CARL VON OSSIETZKY universität OLDENBURG

Automatic Failure Diagnosis based on Timing Behavior Anomaly Detection in Distributed Java Web Applications

Diploma Thesis Presentation

Nina S. Marwede

Abteilung Software Engineering
Fakultät II – Department für Informatik

August 26, 2008

First examiner	Prof. Dr. Wilhelm Hasselbring
Second examiner	MIT Matthias Rohr
Advisor	Dipl.-Inform. André van Hoorn
Advisor	MIT Matthias Rohr

Contents

Motivation for Automatic Failure Diagnosis

- Software systems are practically never free of faults
- Software failures have great influence on our lives
- Large effort for manual diagnosis and debugging
- Automated processes are required
  1. Failure detection
  2. **Fault localization**
  3. Fault removal

Motivation for Automatic Failure Diagnosis

- Software systems are practically never free of faults
- Software failures have great influence on our lives
- Large effort for manual diagnosis and debugging
- Automated processes are required
  1. Failure detection
  2. **Fault localization**
  3. Fault removal

Monitoring of System Behavior

- Log files
- User interfaces
- Resources
- Control flow
- *Timing behavior*

→ Instrumentation of hardware/software

Kieker [Rohr et al., 2008]

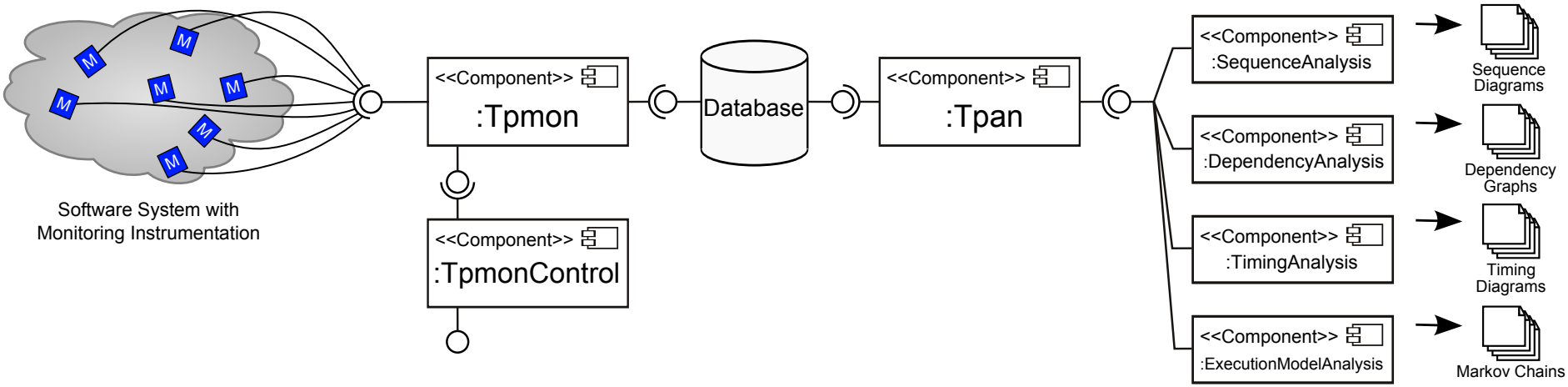
Monitoring of System Behavior

- Log files
- User interfaces
- Resources

- Control flow
- *Timing behavior*

→ Instrumentation of hardware/software

Kieker [Rohr et al., 2008]



Failure Diagnosis

- Model checking: explicit messages
- Timing behavior: throughput, latency, *response times*
- Anomaly detection: statistical analysis
- Correlation: connection of information from different sources
- Goal: cause instead of symptoms
- Visualization

Goals

1. Design of an approach for fault localization
   - Timing behavior anomaly detection [Rohr, 2008]
   - Calling dependencies between components *(dependency graphs)*
   - Focus on event correlation

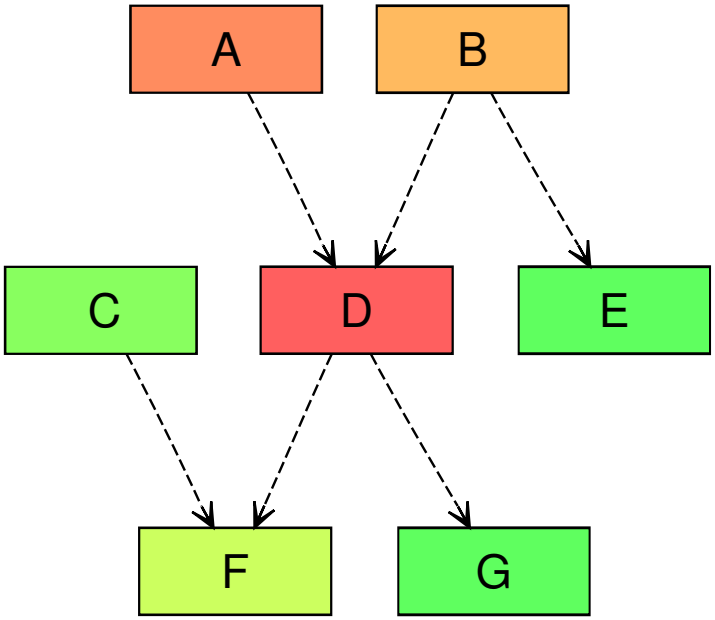
   $\Rightarrow$ "Anomaly Correlator"

2. Evaluation: Case Study
   - Java Web Application: iBATIS JPetStore
   - Workload Generation: Markov4JMeter [van Hoorn et al., 2008]
   - Fault Injection

Contents

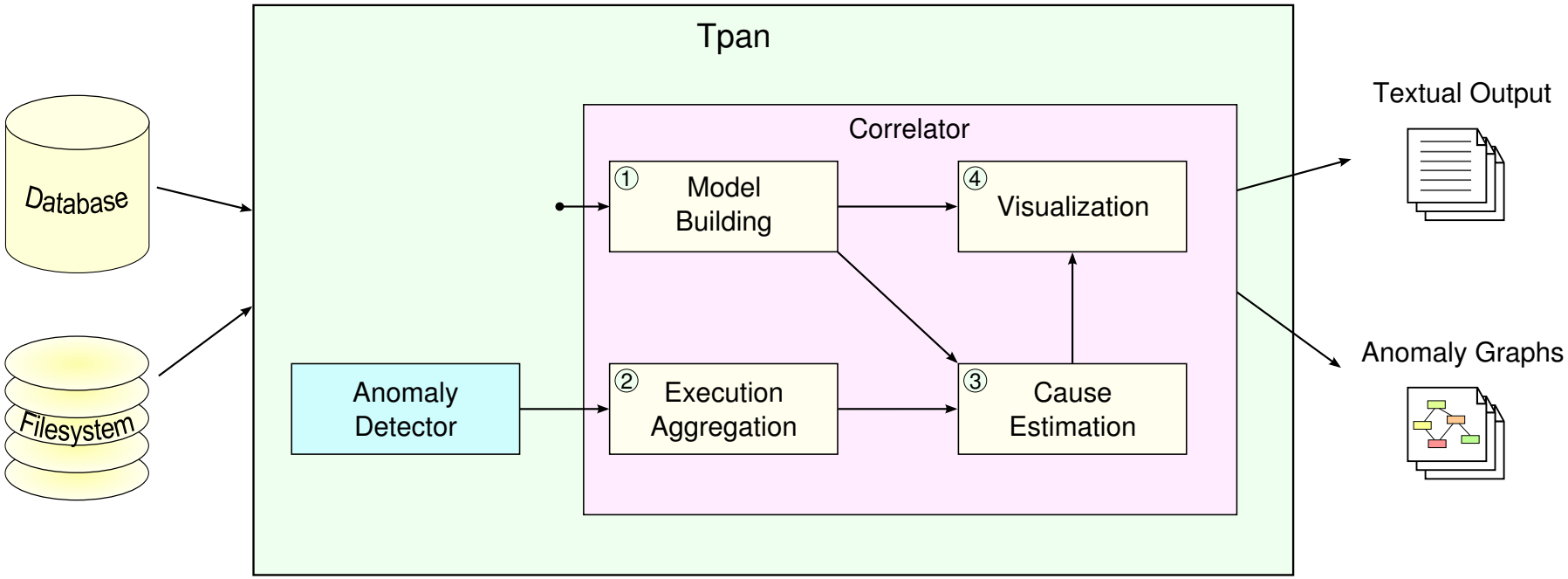
Solution Idea

- Correlation: Draw conclusions from the arrangement of the anomalies in the calling dependency graph



Implementation

- Extension to existing software "Kieker" [Rohr et al., 2008]
- *Tpmon* stores monitoring data, *Tpan* with its plug-ins analyzes it
- Correlator: Plug-in for Tpan

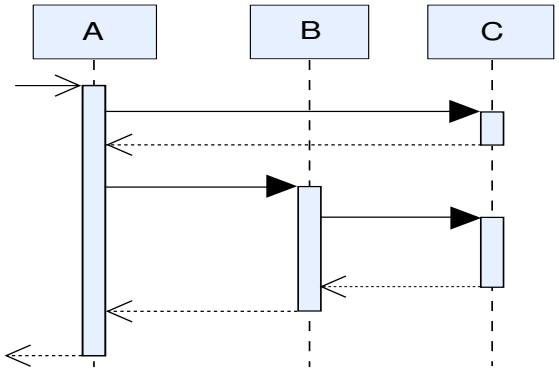


Assumptions

- Correct failure detection
- Correct anomaly scoring
- Failure has distinct cause
- Exactly one failure in the observation period
- Anomaly *propagation*

Input Data

1. Calling dependencies between operations

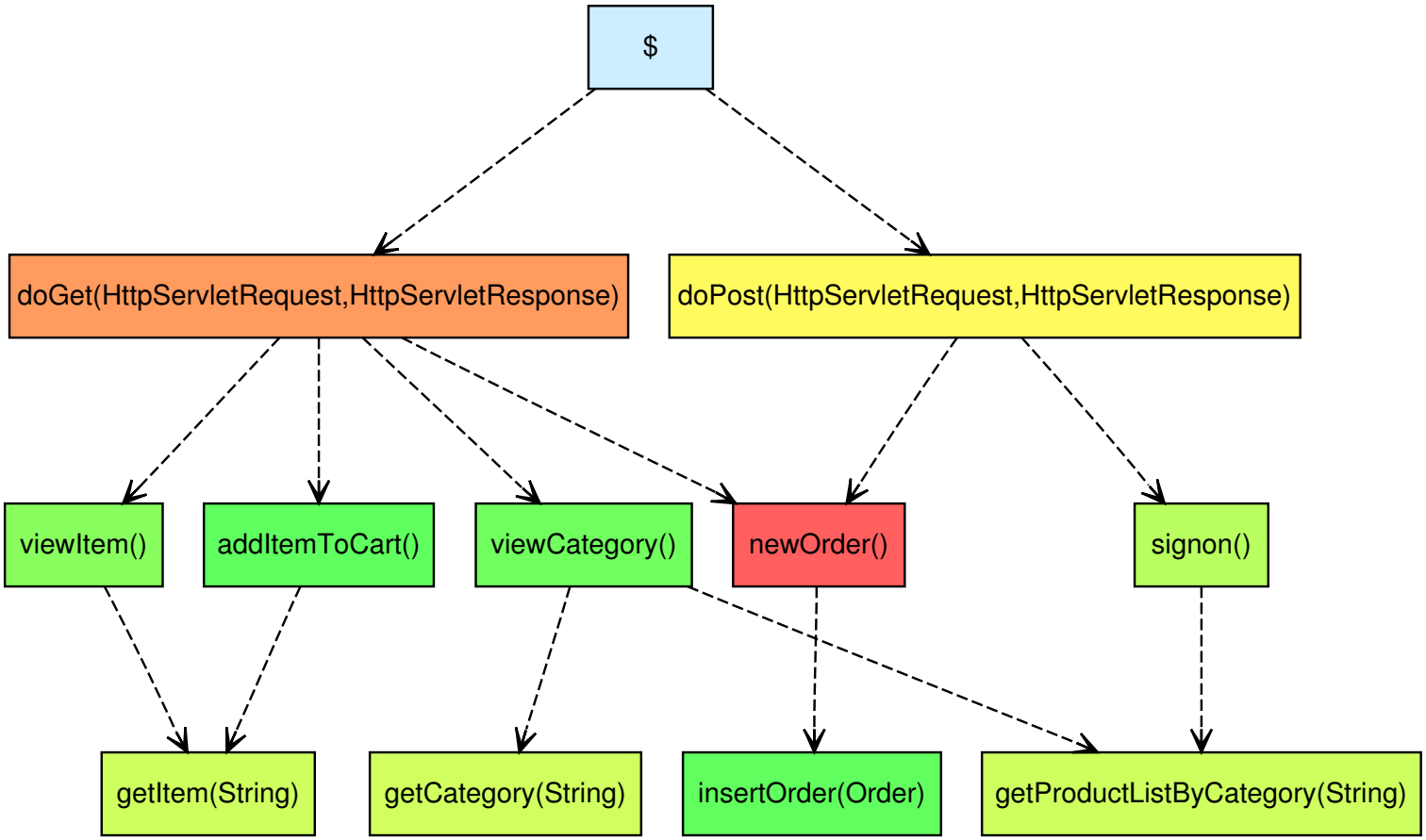


2. Anomalies in the timing behavior of executions

Comp	VM	Start	RT	Anomaly
...				
A	X	0001	8	0.6
C	Y	0002	1	–0.2
B	X	0004	4	0.9
C	Y	0006	2	0.3
...				

Step 1: Preparation of Data Structures

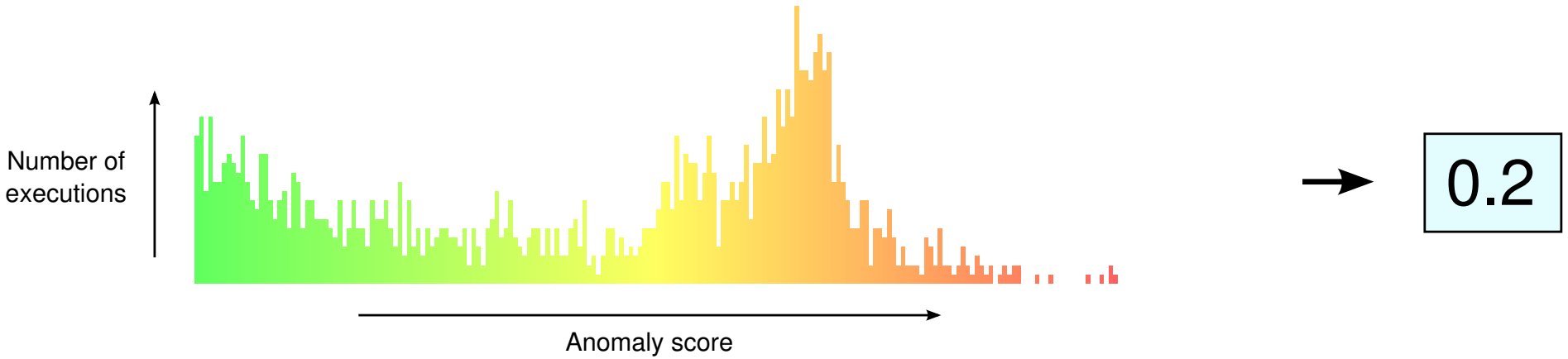
- Generation of calling dependency graphs from traces
- Connection of anomalies with software architecture



Challenges (1/2)

Aggregation:
How to aggregate a number of anomaly scores into one value?

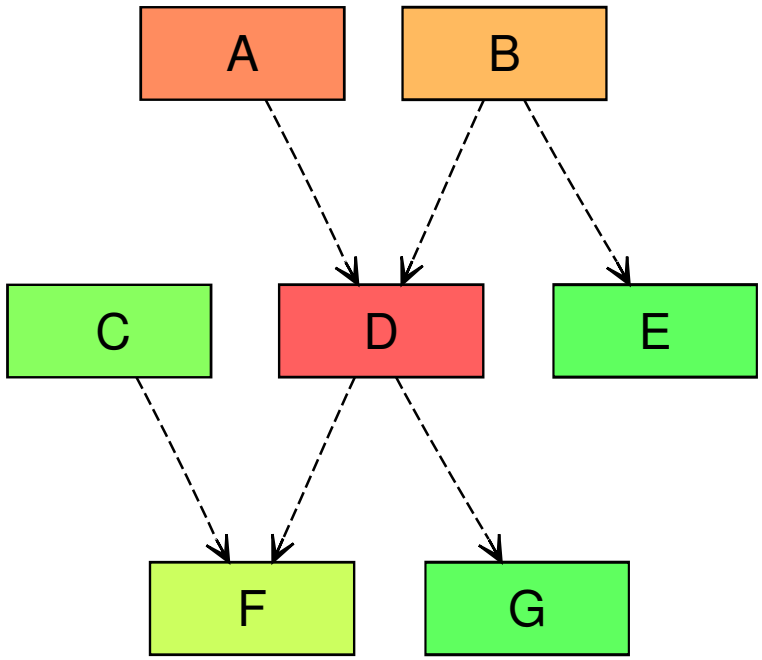
- Four places are involved:
  Three architectural levels, and neighbors on operation level
- Five methods are evaluated:
  Median, power mean (three exponents), maximum



Challenges (2/2)

Correlation:
How to recognize the propagation of an anomaly?

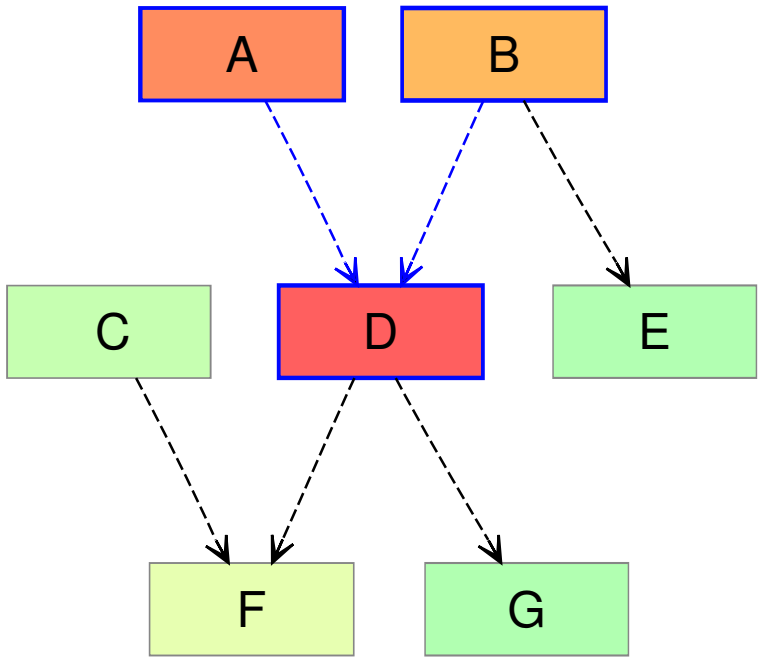
- Consider the perspective of each component
- Three algorithms are evaluated:
  Trivial, simple, advanced



Challenges (2/2)

Correlation:
How to recognize the propagation of an anomaly?

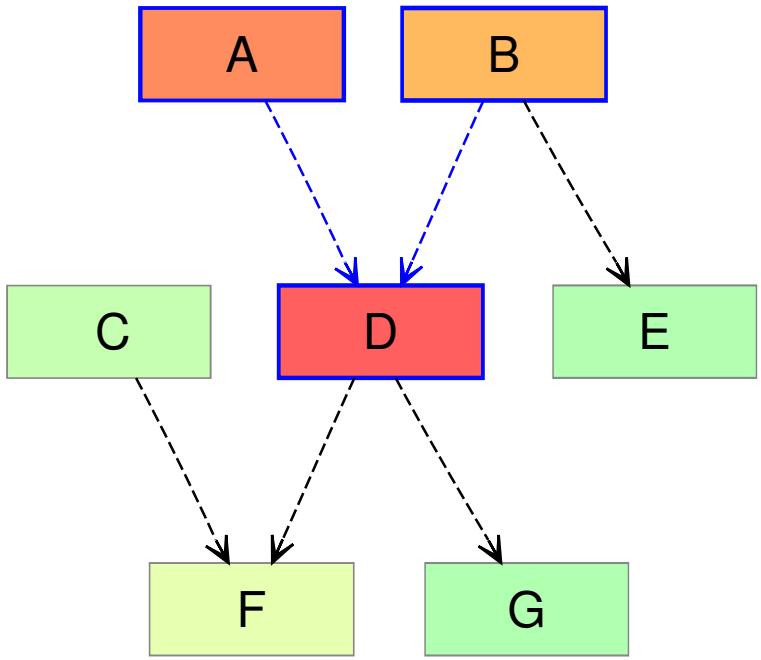
- Consider the perspective of each component
- Three algorithms are evaluated:
  Trivial, simple, advanced



Challenges (2/2)

Correlation:
How to recognize the propagation of an anomaly?

- Consider the perspective of each component
- Three algorithms are evaluated:
  Trivial, simple, advanced



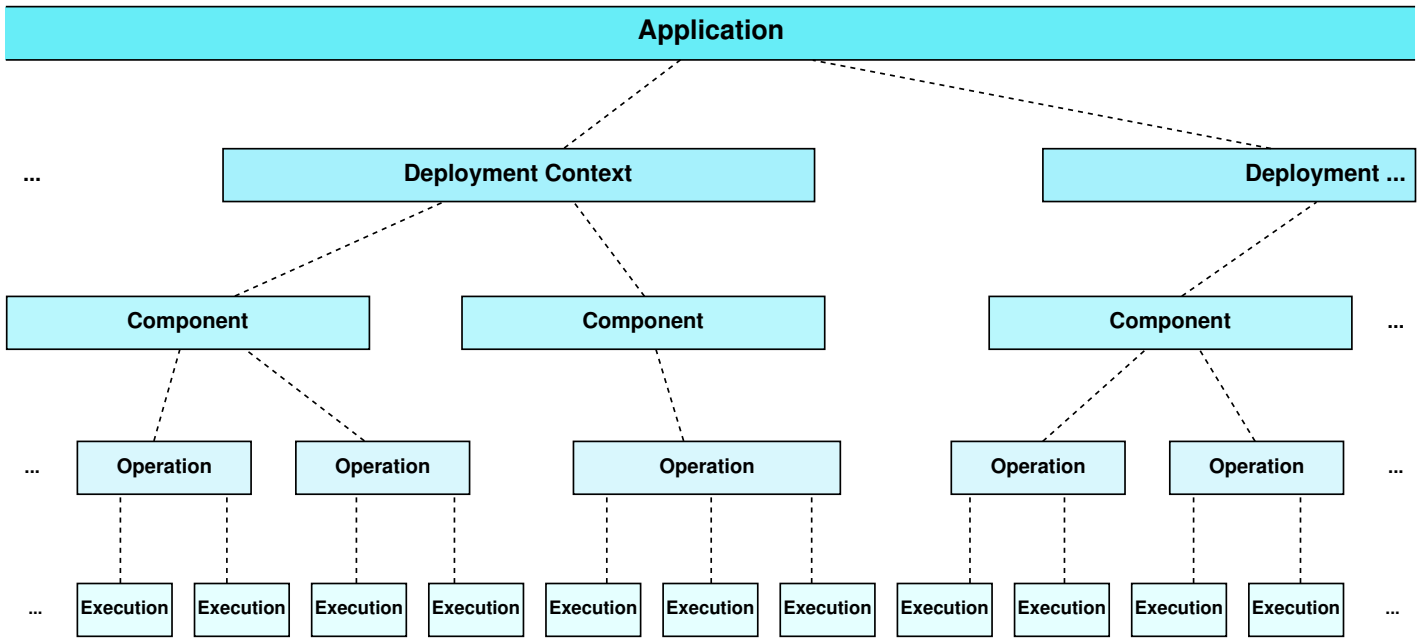
Step 2: Processing of Anomaly Scores

Three algorithms

1. Trivial:
   Simple aggregation, no correlation
2. Simple:
   Simple aggregation, “pessimistic” correlation
3. Advanced:
   Weighted configurable aggregation, “optimistic” correlation

Trivial Algorithm

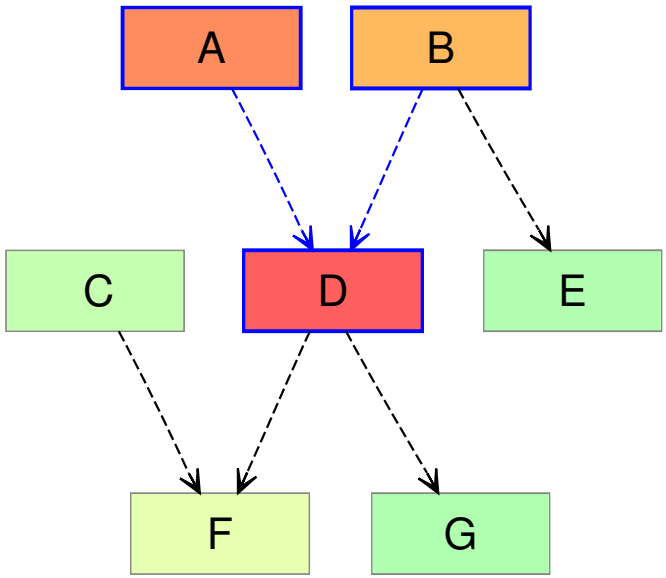
- Aggregation: Unweighted arithmetic mean on each level
- Correlation: None



Simple Algorithm

1. Rule 1:
   Mean of anomaly ratings of directly connected **callers** ...
   relative high? $\Rightarrow$ Increase rating

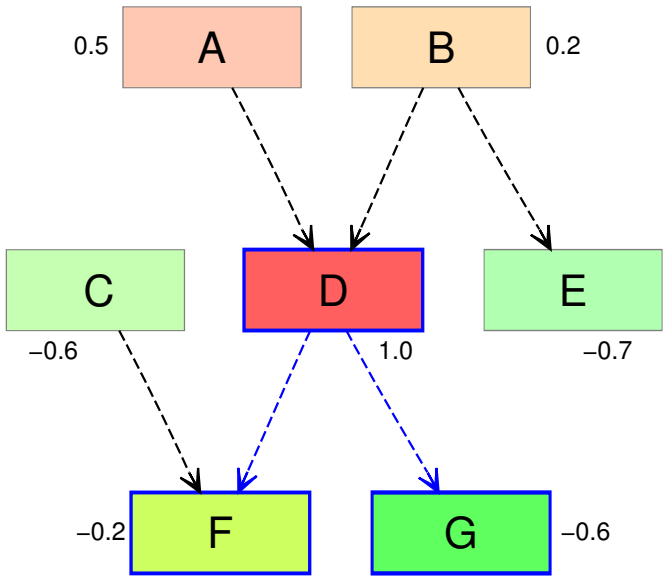
2. Rule 2:
   Maximum of anomaly ratings of directly connected **callees** ...
   relative high? $\Rightarrow$ Decrease rating



Simple Algorithm

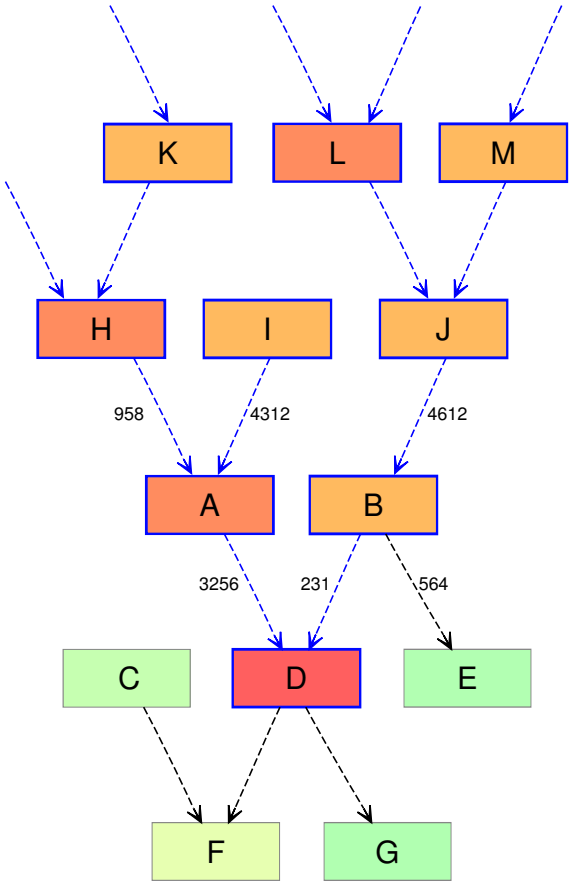
1. Rule 1:
   Mean of anomaly ratings of directly connected **callers** ...
   relative high? ⇒ Increase rating

2. Rule 2:
   Maximum of anomaly ratings of directly connected **callees** ...
   relative high? ⇒ Decrease rating



Advanced Algorithm

- Aggregation
  - In addition to arithmetic mean: median, power mean, maximum
- Correlation
  - Consideration of call frequencies (edges in CDG)
  - Transitive closure of callers
  - Transitive closure of callees



Step 3: Output

1. Text output

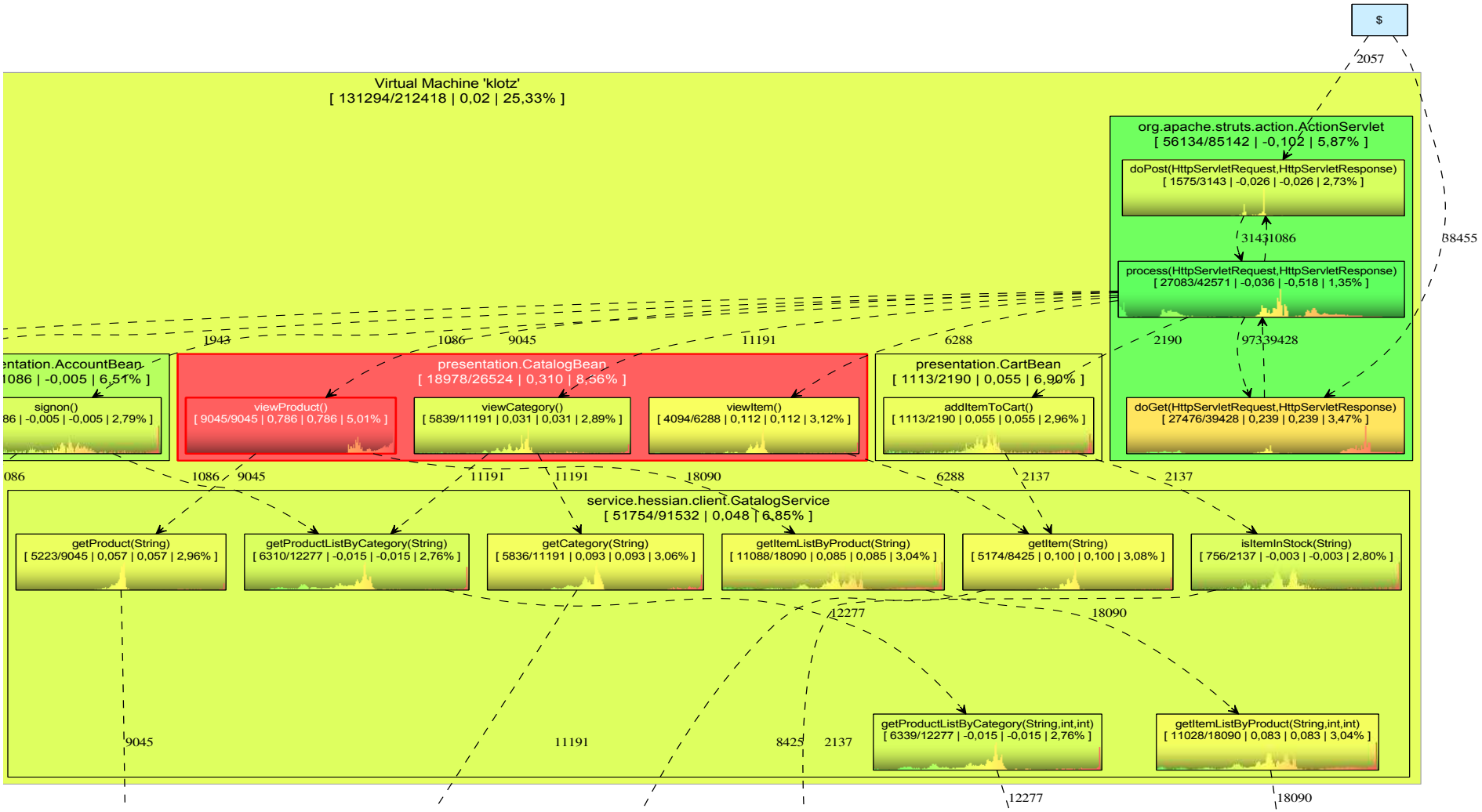
```
Components sorted by cause rating in descending order:
==================
8.89% persistence.sqlmapdao.ItemSqlMapDao
7.78% service.hessian.server.CatalogService
6.91% persistence.sqlmapdao.ProductSqlMapDao
6.83% presentation.OrderBean
6.71% org.apache.struts.action.ActionServlet
6.03% persistence.sqlmapdao.AccountSqlMapDao
...
```

Cause rating for:

- Deployment contexts (e.g., virtual machines)
- Components (e.g., Java classes)
- Operations (e.g., Java methods)

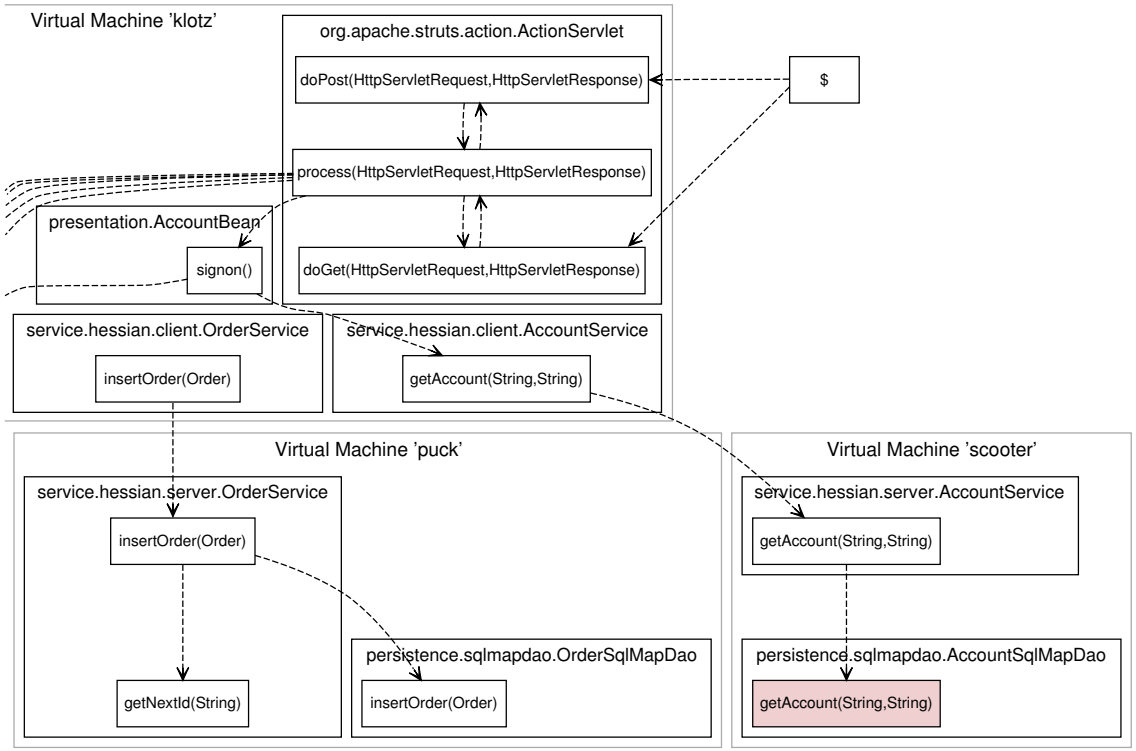
Step 3: Output (cont'd)

2. Visualization of the application hierarchy structure



Visualization Parameters

- Hierarchy levels
- Node and edge annotations
- Color shade spectrum
- Embedded anomaly score histograms
- Additional explanations, caption, legend, . . .



Contents

Goals & Metrics

Goals

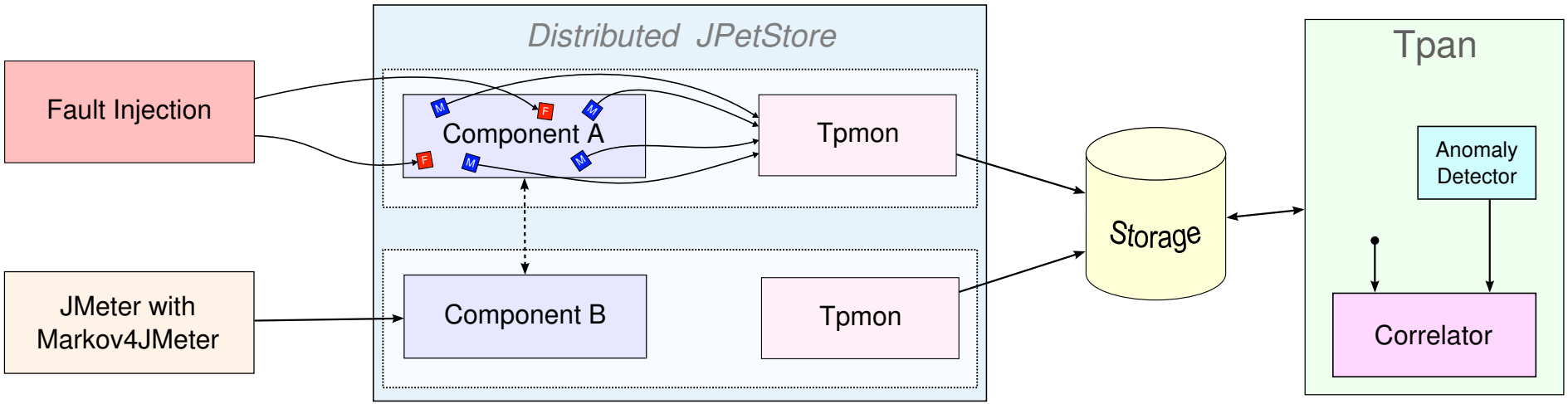
- Proof of concept
- Quantitative evaluation of the Correlator
- Appropriate visualization

Metrics

- Success rating:
  Percent value comparing the highest rated element to the element where the fault injection happened.
- Clearness rating:
  Reflects the visual impression of contrast.

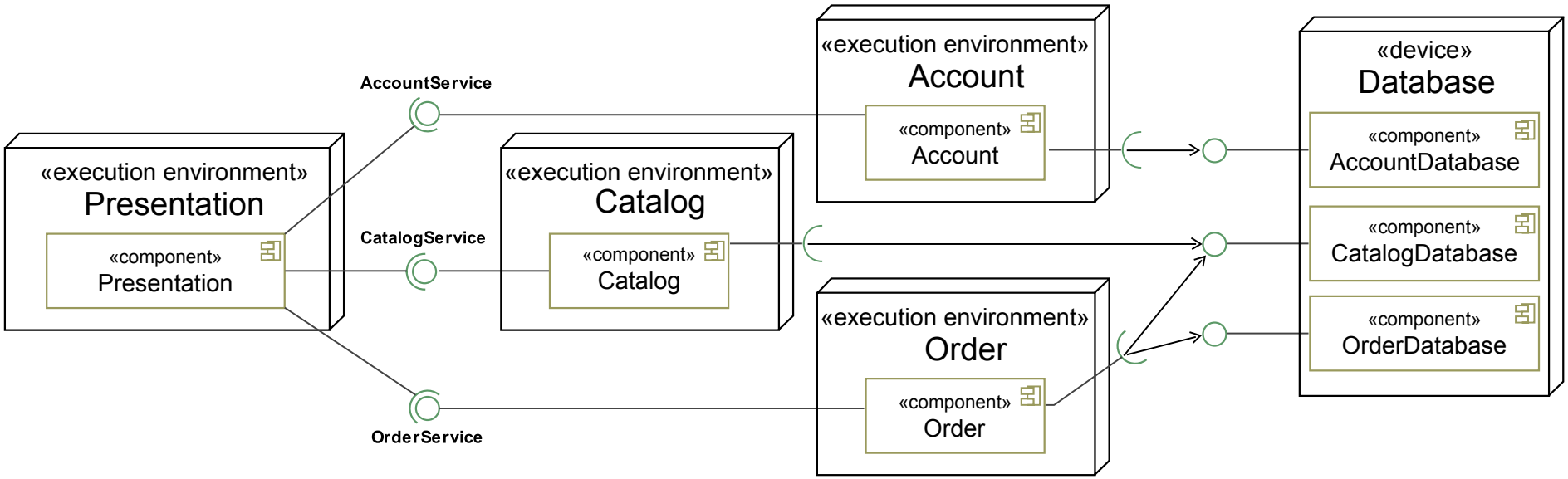
Experiment Setup

- Non-trivial software system: JPetStore
- Application distributed to 4 machines
- Workload generation
  - Probabilistic user behavior
  - Constant number of users
- Fault injection
- Monitoring using Tpmon



Distributed JPetStore

- 4 deployment contexts + DBMS
- 34 operations are instrumented with monitoring probes

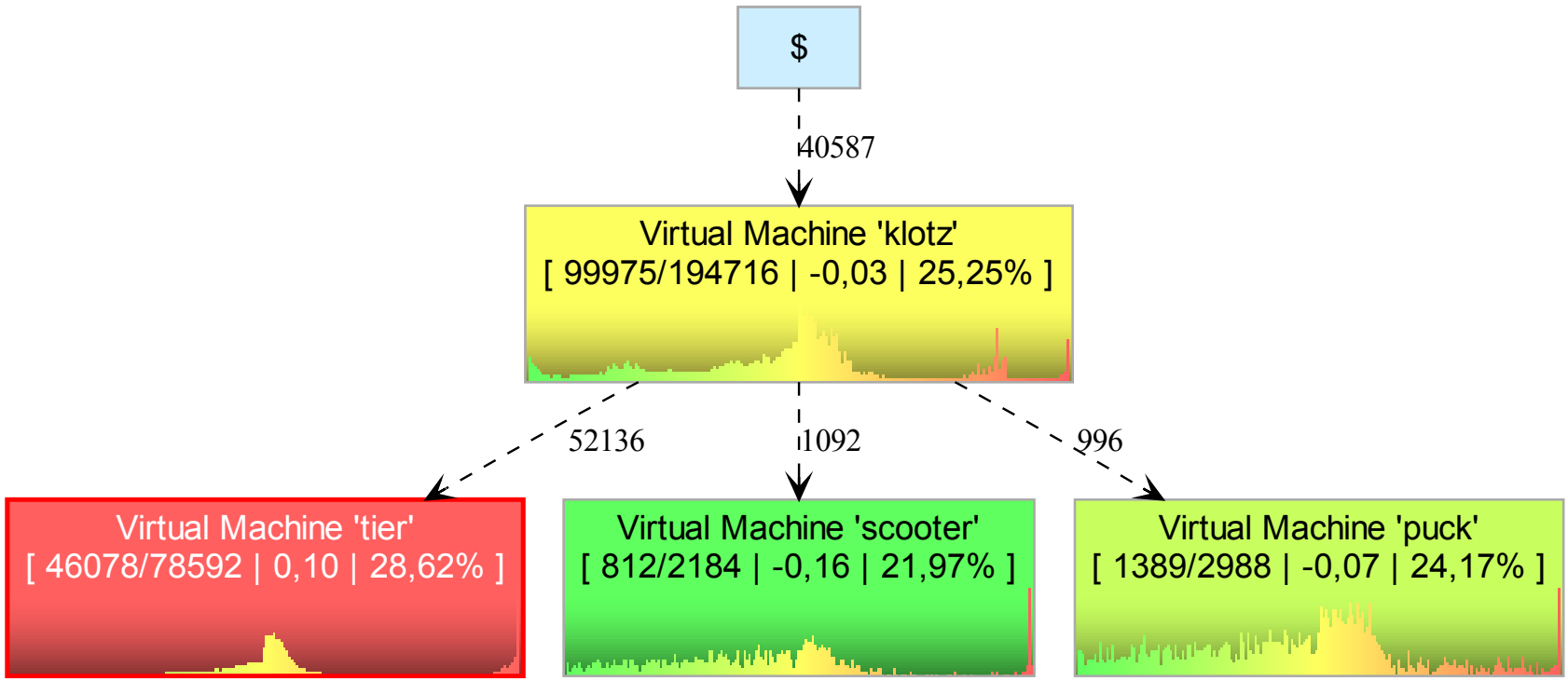


Fault Injection

1. Programming faults
   - Duplicated code execution
   - Empty DB query result set
2. Database connection slowdown
   - `Thread.sleep(10)`
3. Hard disk misconfiguration
   - `hdparm -X mdma1 /dev/hda`
4. Resource intensive processes
   - "Reiner's Fork Bomb"
5. CPU throttling
   - To simulate a broken CPU cooling system
   - `cpufreq-set -f 800`
   - Clock duty cycle of 50%

Experiment Statistics

- 42 experiment runs + 3 fault-free runs
- 20 hours total experiment time
- 16 million monitored executions
- 100 MB data per experiment run



Correlator Configuration

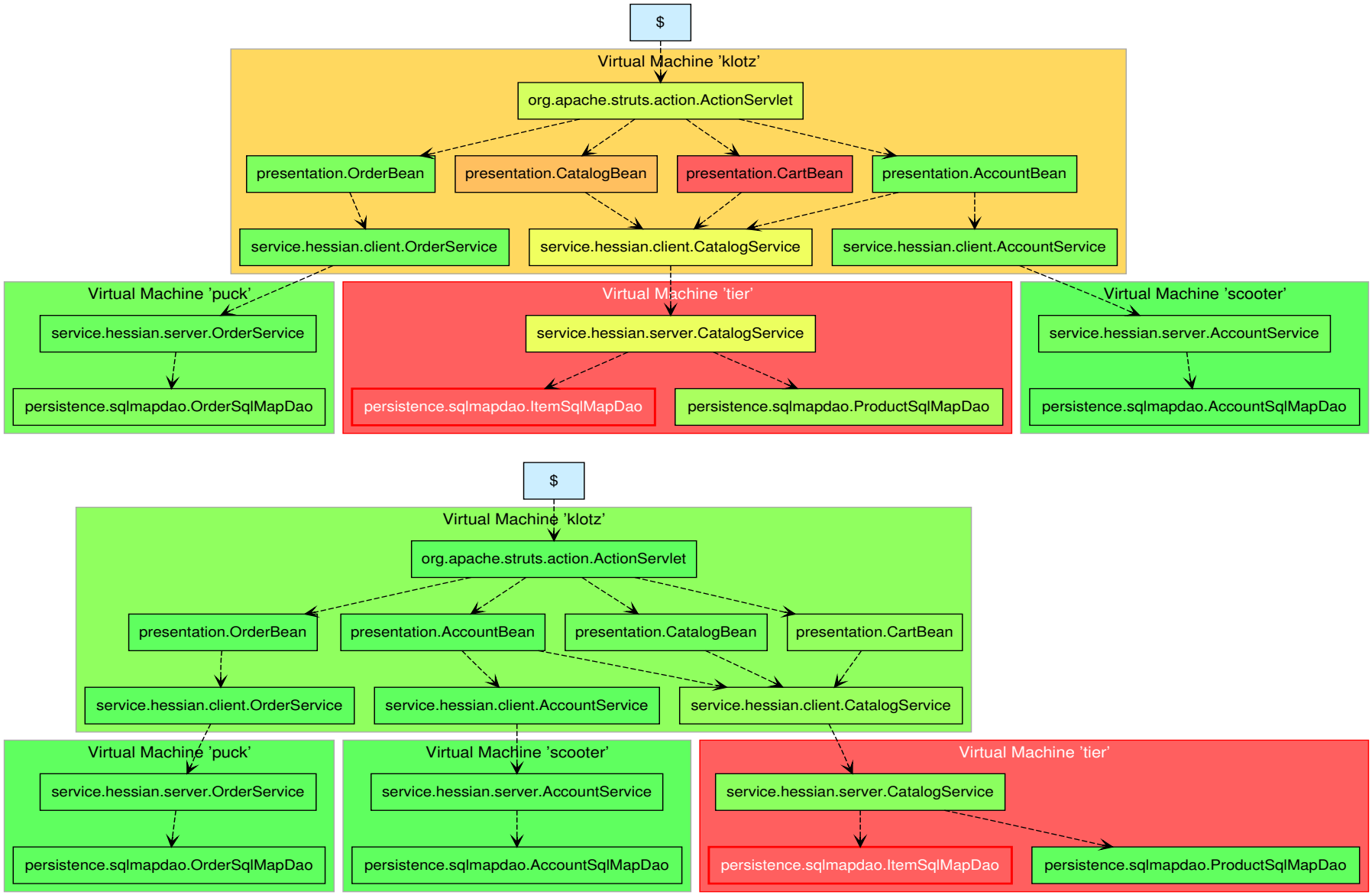
- Algorithm selection – 3 implemented; extendable
- Algorithm parameters – 11 variables for advanced algorithm
- Result export detail level and file names – 7 variables
- Visualization parameters – 12 feature switches, 9 color selections, 5 font settings, 4 others (30 total)

$\longrightarrow$ Java properties

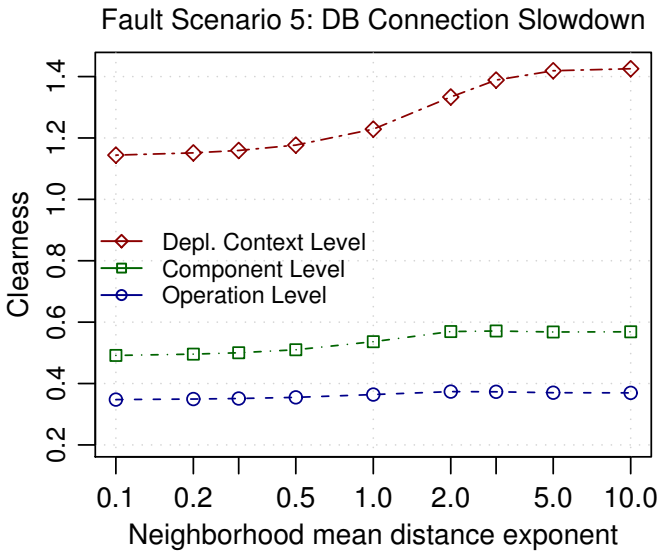
Results: Quality of the Correlation Algorithms

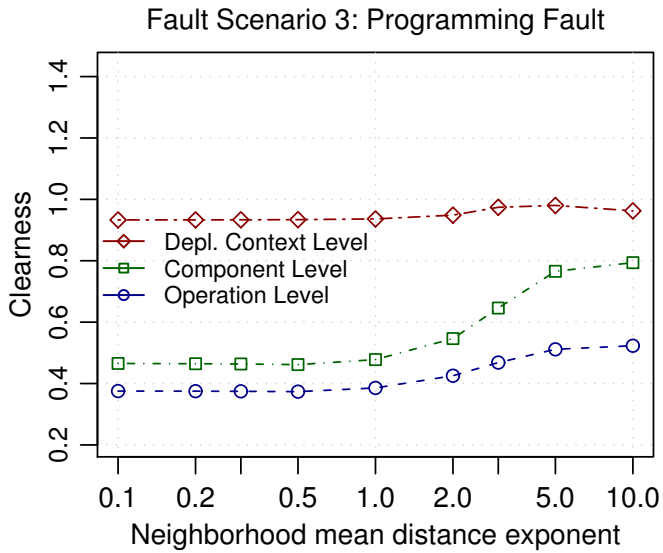
No.	Injection Variant	Trivial	Simple	Advanced	Optimized
1	Programming fault 1	+	+	- -	o
2	Programming fault 2	+	+	o	++
3	Programming fault 3	-	-	+	++
4	DB conn. slowdown 1	+	++	o	++
5	DB conn. slowdown 2	+	++	+	++
6–14	other	- -			
1–5	Averages	2.4	2.0	2.8	1.4

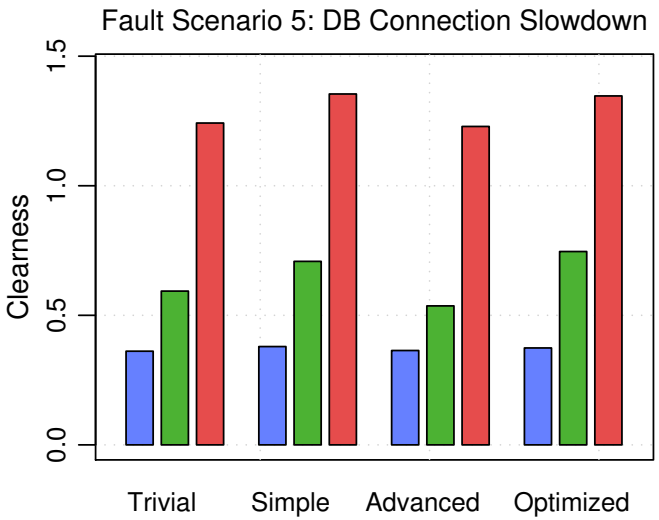
Results: Visualization Clearness – Trivial vs. Optimized

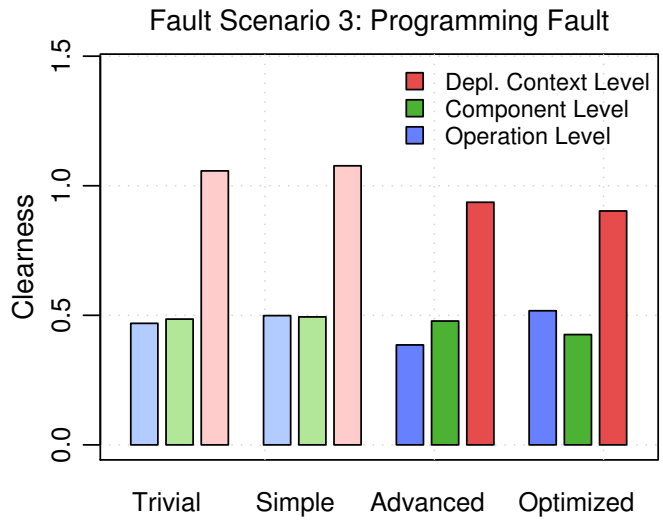


Results


Fault Scenario 5: DB Connection Slowdown
Clearness
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Depl. Context Level
Component Level
Operation Level
0.1
0.2
0.5
1.0
2.0
5.0
10.0
Neighborhood mean distance exponent


Fault Scenario 3: Programming Fault
Clearness
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Depl. Context Level
Component Level
Operation Level
0.1
0.2
0.5
1.0
2.0
5.0
10.0
Neighborhood mean distance exponent


Fault Scenario 5: DB Connection Slowdown
Clearness
0.0
0.5
1.0
1.5
Trivial
Simple
Advanced
Optimized


Fault Scenario 3: Programming Fault
Clearness
0.0
0.5
1.0
1.5
Depl. Context Level
Component Level
Operation Level
Trivial
Simple
Advanced
Optimized

Summary & Conclusions

Summary

- New approach for failure diagnosis
- Implementation and evaluation of correlation algorithms
- Visualization of the results (vector graphic export)

Conclusions

- Good chance of pointing to the right cause
- Small risk of denoting false positives
- At least large parts are declared as *not* containing a fault
- Simpler algorithms show more the effect, less the cause

Future Work

- Continuous analysis and visualization (in progress)
  ("Software-Betriebsleitstand") [Giesecke et al., 2006]
- Application to industry system data (in progress)
- Recognition of known anomaly patterns learned from training data
- Improved interactive user interface

Bibliography

Algirdas Anthony Avižienis, Jean-Claude Laprie, Brian Randell, and Carl E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

Simon Giesecke, Matthias Rohr, and Wilhelm Hasselbring. Software-Betriebs-Leitstände für Unternehmensanwendungslandschaften. In *Proceedings of the Workshop "Software-Leitstände: Integrierte Werkzeuge zur Softwarequalitätssicherung"*, volume P-94 of *Lecture Notes in Informatics*, pages 110–117. Gesellschaft für Informatik, October 2006.

Matthias Rohr. *Workload-sensitive Timing Behavior Anomaly Detection for Automatic Software Failure Diagnosis*. PhD thesis, Department of Computing Science, University of Oldenburg, Oldenburg, Germany, 2008. *work in progress*.

Matthias Rohr, André van Hoorn, Jasminka Matevska, Nils Sommer, Lena Stoever, Simon Giesecke, and Wilhelm Hasselbring. Kieker: Continuous monitoring and on demand visualization of Java software behavior. In Claus Pahl, editor, *Proceedings of the IASTED International Conference on Software Engineering 2008 (SE 2008)*, pages 80–85, Anaheim, February 2008. ACTA Press.

André van Hoorn, Matthias Rohr, and Wilhelm Hasselbring. Generating probabilistic and intensity-varying workload for web-based software systems. In Samuel Kounev, Ian Gorton, and Kai Sachs, editors, *Performance Evaluation – Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop (SIPEW '08)*, volume 5119 of *Lecture Notes in Computer Science (LNCS)*, pages 124–143, Heidelberg, June 2008. Springer.

Thanks for your attention.

Requirements for Fault Injection

- Noticeable effect
- No administrative messages
- Diversity in position
- All hierarchy levels
- Realistic, and repeatable
- Increasing & decreasing the response times