Automated Debugging of SLO Violations in Enterprise Systems

Sangameshwar Patil,
Maitreyya Natu, Vaishali Sadaphal, Harrick Vin
Detection and Debugging of SLO Violations is Critical

• SLOs are generally defined with respect to
  - End-to-end latencies
  - Throughput
  - Availability
  - Reliability

• Rapid detection and debugging of SLO violations is critical

• Increasing scale and complexity of today’s enterprise systems demands automated debugging of SLO violations
Automated Debugging of SLO Violations

• Identifying root-cause of the SLO violations involves two steps:
  - Identify the component most likely responsible for SLO violations
    • Typically involves analysis based on graph theory and queuing theory
  - Further analyze the operations of the component and determine the root-cause of SLO violation
Automated Debugging of SLO Violations

- Identifying root-cause of the SLO violations involves two steps:
  - Identify the component most likely responsible for SLO violations
    - Typically involves analysis based on graph theory and queuing theory
  - Further analyze the operations of the component and determine the root-cause of SLO violation

Given a component identified as the cause of SLO violation, identify the bottleneck resources that are the root-causes of the observed SLO violation in near real time
Scale Presents Two Main Challenges

• Large number of metrics are monitored for each component:
  - Workload, latency, CPU, memory, IO and network utilization, cache hit/miss rates, etc.

• Large number of data-points for each metric
  - Metrics are monitored at fine time-scales (e.g., every few seconds)
Key Observations

• Only a few metrics are sufficient to explain an SLO violation
• Analysis of data points only around the time-periods representing SLO violations is sufficient to explain an SLO violation
Given a time series of the performance metrics observing SLO violation and a set of time series of the *all* component-level metrics, detect the root-causes of the observed violation.
Given a time series of the performance metrics observing SLO violation and a set of time series of the selected $k$ component-level metrics, detect the root-causes of the observed violation.
Given a *time window* of the performance metrics observing SLO violation and a set of time series of the *selected k* component-level metrics, detect the root-causes of the observed violation.

**Temporal Pruning:** Remove irrelevant and redundant metrics using CARTs.
Proposed Solution

<table>
<thead>
<tr>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance metrics</td>
</tr>
<tr>
<td>Workload metrics</td>
</tr>
<tr>
<td>System metrics</td>
</tr>
<tr>
<td>Middleware metrics</td>
</tr>
<tr>
<td>Application metrics</td>
</tr>
</tbody>
</table>

- **SLO Definitions**
- **SLO Violation Detection**
- **Feature Selection**
- **CART Model**
- **Violation Windows**
- **Temporal Selection**
- **Change Points**

Analyse the performance metrics and likely cause metrics using change-point correlation.

**Hypotheses Report**
A Running Example

• **Input**
  - Time series of 17 component-level metrics F0 through F16
  - Latency of the requests served by the component

• **SLO violation instance:**
  - Actual cause: metrics F3 and F4
Given a time series of the performance metrics observing SLO violation and a set of time series of the *selected k* component-level metrics, detect the root-causes of the observed violation.
Feature Selection: CARTs

• **CARTs: Classification and Regression Trees**

• Special class of decision trees where the target metric values can be categorical or real numbers

• Leaves represent classification of target metric

• Paths from root to leaves represent various if-then conditions to infer relationships between target metric and observation metrics
Feature Selection: Running Example

- Using CARTs for feature selection
  - The root node of a CART provides best classification of the target metric
  - While choosing the root node metric, classification accuracy of other likely root-node candidates is also computed
  - Select the root node and its competitors as the most likely as the features of interest that should be considered for further analysis

Selected features: \{F0, F3, F4, F7\}
Temporal Selection

Given a *time window* of the performance metrics observing SLO violation and a set of time series of the *selected k* component-level metrics, detect the root-causes of the observed violation.
Temporal Selection: Running Example

- In the performance metric observing SLO violation, detect a change-point in the vicinity of SLO violation.
- Define a temporal region of interest consisting of data-points before and after the change-point within a window of time.

Selected temporal regions:
[160s to 260s] and [580s to 680s]
Change Point Correlation

Analyze the performance metrics and likely cause metrics using change-point correlation.
Change Point Correlation: Running Example

- In the time-series of identified features in the temporal regions
  - Detect the presence of change points

Likely causes: F3 and F4
Application on a real-life example

• Batch processing system

• Available data:
  - Latency
  - Wait time
  - CPU time
  - File IO time
  - Job run count
  - Workload1, Workload2, Workload3

• SLO definition: Latency <= 200 ms
Feature Selection

Selected features: \{Workload1, Workload2, Workload3, CPU time\}
Temporal Selection

![Graph showing latency over time with identified temporal regions of interest, change points, and SLO threshold.]

- Temporal regions of interest
- Change points
- SLO threshold

**Latency (in sec)**

**Time (in sec)**
Identified causes: {Workload1, Workload2, Workload3}
Experimental Evaluation

• Techniques used for comparison
  - Time series correlation
  - Bayesian networks
  - Change-point correlation
  - Classification and regression trees (CARTs)

• Experiment setup
  - Simulated a component using CSIM simulator and measured
    • Workload
    • Latency
    • Component-level metrics such as file IO time, wait time, CPU time, etc. that contribute to the overall latency
  - Faults
    • Increase in one or more component-level metrics that result in overall increase in component-level latency
  - Result of each scenario is an average of 10 runs
Experimental Evaluation

• Experiments Performed
  - Effect of increasing number of faults
    • Demonstrates the localization accuracy in the presence of multiple failures
  - Effect of increasing number of metrics
    • Demonstrates that there is a fundamental trade-off between accuracy and execution time
Experimental Evaluation

Effect of increasing number of faults

- Correlation is inaccurate
- Bayesian network is compute-intensive
- CART+CP outperforms CART and CP
Experimental Evaluation

Effect of increasing number of metrics

- Tradeoff between accuracy and execution time
- Bayesian network is compute-intensive
- CART+CP outperforms CART and CP
Conclusion

• We addressed the problem of automated debugging of SLO violations

• To address large volume of data, there is a need to intelligently prune the search space
  - We perform feature selection to remove irrelevant and redundant metrics (using CARTs)
  - We identify temporal regions of interest (using Change Point detection)
  - We then use change-point correlation to identify the root-causes of SLO violations

• We demonstrate through experimental evaluation that the proposed approach not only reduces the execution time but also increases the debugging accuracy
Thanks!
Additional Slides
CART sample example: Training data

<table>
<thead>
<tr>
<th>Car Type</th>
<th>Driver Age</th>
<th>Children</th>
<th>Lives in Suburb?</th>
</tr>
</thead>
<tbody>
<tr>
<td>sedan</td>
<td>23</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>sports</td>
<td>31</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>sedan</td>
<td>36</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>truck</td>
<td>25</td>
<td>2</td>
<td>no</td>
</tr>
<tr>
<td>sports</td>
<td>30</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>sedan</td>
<td>36</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>sedan</td>
<td>25</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>truck</td>
<td>36</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>sedan</td>
<td>30</td>
<td>2</td>
<td>yes</td>
</tr>
<tr>
<td>sedan</td>
<td>31</td>
<td>1</td>
<td>yes</td>
</tr>
<tr>
<td>sports</td>
<td>25</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>sedan</td>
<td>45</td>
<td>1</td>
<td>yes</td>
</tr>
<tr>
<td>sports</td>
<td>23</td>
<td>2</td>
<td>no</td>
</tr>
<tr>
<td>truck</td>
<td>45</td>
<td>0</td>
<td>yes</td>
</tr>
</tbody>
</table>
CART sample example

Figure 1.1: Example of classification tree for training data in Table 1.1