Observability: Logging

Abdul Manan Xiling Zhang
Introduction

- Hard to reproduce and diagnose failures in testing environments
- Still challenging: fault localization
- How to understand the logs
Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs
Latent error prediction and fault localization

MEPFL (Microservice Error Prediction and Fault Localization)

- Support 3 types of microservice application faults
  - i.e., multi-instance faults, configuration faults, and asynchronous interaction faults
- Feature definition
- Trains prediction models at both the trace level and microservice level
  - system trace logs collected from automatic executions of the target application and its faulty versions produced by fault injection
- Evaluation
  - two open-source microservice benchmarks
  - real-world fault cases
Fault Types

- **Monolithic Faults.** Can cause failures even when deployed in a monolithic mode
- **Multi-Instance Faults.** Related to the existence of multiple instances of the same microservice. Mainly caused by inconsistent states
- **Configuration Faults.** Related to the environmental configurations of microservices, such as the resource (e.g., memory and CPU) limits.
- **Asynchronous Interaction Faults.** Related to the asynchronous invocations and may cause failures when invocations are executed or returned in an unexpected order.
## Fault Types

<table>
<thead>
<tr>
<th>Fault</th>
<th>Reporter</th>
<th>Symptom</th>
<th>Root Cause</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>P1 (A1)</td>
<td>Messages are displayed in wrong order</td>
<td>Asynchronous message delivery lacks sequence control</td>
<td>7D</td>
</tr>
<tr>
<td>F2</td>
<td>P2 (A2)</td>
<td>Some information displayed in a report is wrong</td>
<td>Different data requests for the same report are returned in an unexpected order</td>
<td>3D</td>
</tr>
<tr>
<td>F3</td>
<td>P2 (A2)</td>
<td>The system periodically returns server 500 error</td>
<td>JVM configurations are inconsistent with Docker configurations</td>
<td>10D</td>
</tr>
<tr>
<td>F4</td>
<td>P3 (A3)</td>
<td>The response time for some requests is very long</td>
<td>SSL offloading happens in a fine granularity (happening in almost each Docker instance)</td>
<td>7D</td>
</tr>
<tr>
<td>F5</td>
<td>P4 (A4)</td>
<td>A service sometimes returns timeout exceptions for user requests</td>
<td>The high load of a type of requests causes the timeout failure of another type of requests</td>
<td>6D</td>
</tr>
<tr>
<td>F6</td>
<td>P5 (A5)</td>
<td>A service is slowing down and returns error finally</td>
<td>Endless recursive requests of a microservice are caused by SQL errors of another dependent microservice</td>
<td>3D</td>
</tr>
<tr>
<td>F7</td>
<td>P6 (A6)</td>
<td>The payment service of the system fails</td>
<td>The overload of requests to a third-party service leads to denial of service</td>
<td>2D</td>
</tr>
<tr>
<td>F8</td>
<td>P7 (A7)</td>
<td>A default selection on the web page is changed unexpectedly</td>
<td>The key in the request of one microservice is not passed to its dependent microservice</td>
<td>5D</td>
</tr>
<tr>
<td>F9</td>
<td>P7 (A7)</td>
<td>There is a Right To Left (RTL) display error for UI words</td>
<td>There is a CSS display style error in bi-directional</td>
<td>0.5D</td>
</tr>
<tr>
<td>F10</td>
<td>P8 (A8)</td>
<td>The number of parts of a specific type in a bill of material (BOM) is wrong</td>
<td>An API used in a special case of BOM updating returns unexpected output</td>
<td>4D</td>
</tr>
<tr>
<td>F11</td>
<td>P9 (A8)</td>
<td>The bill of material (BOM) tree of a product is erroneous after updates</td>
<td>The BOM data is updated in an unexpected sequence</td>
<td>4D</td>
</tr>
<tr>
<td>F12</td>
<td>P10 (A9)</td>
<td>The price status shown in the optimized result table is wrong</td>
<td>Price status querying does not consider an unexpected output of a microservice in its call chain</td>
<td>6D</td>
</tr>
<tr>
<td>F13</td>
<td>P11 (A9)</td>
<td>The result of price optimization is wrong</td>
<td>Price optimization steps are executed in an unexpected order</td>
<td>6D</td>
</tr>
<tr>
<td>F14</td>
<td>P11 (A9)</td>
<td>The result of the Consumer Price Index (CPI) is wrong</td>
<td>There is a mistake in including the locked product in CPI calculation</td>
<td>2D</td>
</tr>
<tr>
<td>F15</td>
<td>P11 (A9)</td>
<td>The data-synchronization job quits unexpectedly</td>
<td>The spark actor is used for the configuration of actorSystem (part of Apache Spark) instead of the system actor</td>
<td>3D</td>
</tr>
<tr>
<td>F16</td>
<td>P11 (A9)</td>
<td>The file-uploading process fails</td>
<td>The &quot;max-content-length&quot; configuration of spray is only 2 MB, not allowing to upload a bigger file</td>
<td>2D</td>
</tr>
<tr>
<td>F17</td>
<td>P12 (A10)</td>
<td>The grid-uploading process takes too much time</td>
<td>Too many nested &quot;select&quot; and &quot;from&quot; clauses are in the constructed SQL statement</td>
<td>1D</td>
</tr>
<tr>
<td>F18</td>
<td>P13 (A11)</td>
<td>Loading the product-analysis chart is erroneous</td>
<td>One key of the returned JSON data for the UI chart includes the null value</td>
<td>0.5D</td>
</tr>
<tr>
<td>F19</td>
<td>P13 (A11)</td>
<td>The price is displayed in an unexpected format</td>
<td>The product price is not formatted correctly in the French format</td>
<td>1D</td>
</tr>
<tr>
<td>F20</td>
<td>P14 (A12)</td>
<td>Nothing is returned upon workflow data request</td>
<td>The JBoss startup classpath parameter does not include the right DB2 jar package</td>
<td>3D</td>
</tr>
<tr>
<td>F21</td>
<td>P15 (A13)</td>
<td>JAWS (a screen reader) masses reading some elements</td>
<td>The &quot;aria-labelledby&quot; element for accessibility cannot be located by the JAWS</td>
<td>0.5D</td>
</tr>
<tr>
<td>F22</td>
<td>P16 (A13)</td>
<td>The error of SQL column missing is returned upon some data request</td>
<td>The constructed SQL statement includes a wrong column name in the &quot;select&quot; part according to its &quot;from&quot; part</td>
<td>3.5D</td>
</tr>
</tbody>
</table>
Features Definition

3 scopes and 4 categories

- Global: global property of the microservice and is irrelevant to the current trace instance
- Trace: indicates that the feature is an overall property of the microservice in the current trace instance
- Invocation: indicates that the feature is a property of the current microservice invocation
Features Definition

* A trace instance (which records the process of microservice invocations for a user request)
Features Definition

3 scopes and 4 categories

- Configuration: reflect the environmental configurations of the microservice instance
- Resource features: reflect the resource consumptions
- Instance features: reflect the status of the deployment of the microservice and their involvement in the current trace instance.
- Interaction features: reflect the status of interactions (especially asynchronous interactions) with other microservices
### Features Definition

**Table 2: Feature Definition**

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature</th>
<th>Description</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Configuration</strong></td>
<td>ML</td>
<td>Memory limit of the current microservice relative to the node memory limit</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>CL</td>
<td>CPU limit of the current microservice relative to the node CPU limit</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>VS</td>
<td>Whether volume support is enabled for the current microservice</td>
<td>Global</td>
</tr>
<tr>
<td><strong>Resource</strong></td>
<td>MC</td>
<td>Memory consumption of the current microservice instance relative to its memory limit</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>CC</td>
<td>CPU consumption of the current microservice instance relative to its CPU limit</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>NMC</td>
<td>Memory consumption of the current node relative to its memory limit</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>NCC</td>
<td>CPU consumption of the current node relative to its CPU limit</td>
<td>Invocation</td>
</tr>
<tr>
<td><strong>Instance</strong></td>
<td>IN</td>
<td>Number of instances for the current microservice in the whole system</td>
<td>Global</td>
</tr>
<tr>
<td></td>
<td>IIIN</td>
<td>Number of instances (for the current microservice) invoked in the current trace instance</td>
<td>Trace</td>
</tr>
<tr>
<td></td>
<td>SVA</td>
<td>Ratio of the shared variables that are accessed in the current invocation</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>CA</td>
<td>Whether the cache is accessed in the current invocation</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>SA</td>
<td>Whether the third-party storage (e.g., database) is accessed in the current invocation</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>TN</td>
<td>Number of threads of the current microservice instance</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>LT</td>
<td>Life time of the current microservice instance since its creation</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>NIN</td>
<td>Number of microservices instances residing in the current node</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>NMN</td>
<td>Number of microservices whose instances reside in the current node</td>
<td>Invocation</td>
</tr>
<tr>
<td><strong>Interaction</strong></td>
<td>ET</td>
<td>Execution time of the current microservice invocation</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>RSC</td>
<td>HTTP response status code of the current microservice invocation</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>AIT</td>
<td>Time of an asynchronous invocation in the current microservice execution</td>
<td>Invocation</td>
</tr>
<tr>
<td></td>
<td>CEO</td>
<td>Whether the execution order of a pair of asynchronous invocations is consistent with their invocation order</td>
<td>Invocation</td>
</tr>
</tbody>
</table>
Features Definition

Figure 2: An Example of Trace Instance
Prediction Models

- 4 prediction models - classification model
- Trace level prediction models:
  - latent error prediction
  - faulty microservice prediction
  - fault type prediction
- Microservice level prediction:
  - microservice-status prediction
Prediction Models

- 4 prediction models - classification model
- Trace level prediction models:
  - latent error prediction ----> binary-classification model
  - faulty microservice prediction ----> multi-label classification model
  - fault type prediction ----> multi-label classification model
- Microservice level prediction:
  - microservice-status prediction ----> single-label classification model
MEPFL Approach

Figure 3: Approach Overview
Fault Injection

- Each fault injection strategy is parameterized:
  - precondition
  - code transformation method
  - expected failure symptom
Fault Injection

- Multi-Instance Faults
- Configuration Faults
- Asynchronous Interaction Faults

Figure 3: Approach Overview
public OrderTicketsResult preserve(OrderTicketsInfo oti, String accountId,
String loginToken, HttpHeaders headers) {
    List<CompletableFuture<Void>> futures = new ArrayList<>();
    OrderTicketsResult otr = new OrderTicketsResult();
    List<AddAssuranceResult> colist = new ArrayList<>();
    CompletableFuture<Void> crFuture = CompletableFuture.supplyAsync(() -> {
        HttpEntity requestCheckResult = new HttpEntity(info, headers);
        ResponseEntity<CheckResult> reCheckResult = restTemplate.exchange(
            "security_url", HttpMethod.POST,
            requestCheckResult,
            CheckResult.class);
        return reCheckResult.getBody();
    }).thenAccept(retList::add);
    futures.add(crFuture);

    ....

    CompletableFuture<Void> coFuture = CompletableFuture.supplyAsync(() -> {
        HttpEntity requestEntityCreateOrderResult = new HttpEntity(httpHeaders);
        ResponseEntity<CreateOrderResult> reCreateOrderResult = restTemplate.exchange(
            "order_url", HttpMethod.POST,
            requestEntityCreateOrderResult,
            CreateOrderResult.class);
        CreateOrderResult cor = reCreateOrderResult.getBody();
        CheckResult checkResult = crList.get(0);
        if (cor.isStatus() || checkResult.isStatus()) {
            otr.setStatus(false);
            otr.setMessage(cor.getMessage());
            otr.setOrder(null);
        } else {
            otr.setStatus(true);
            otr.setMessage("Success");
            otr.setOrder(cor.getOrder());
        }
        return otr;
    }).thenAccept(retList::add);
    futures.add(coFuture);
    futures.forEach(x -> x.join());
    return colist.get(0);
Execution Control

- Deploys the target application and its faulty versions
  - manipulates the runtime environment of the application
- Executes different application versions with existing automated test cases
  - For each setting, executes the test cases for a given number (e.g., 10) of times.
  - get multiple trace instances under the same setting
- Collects trace logs
Execution Control

- Deploys the target application and its faulty versions
  - manipulates the runtime environment of the application:
    - multi-instance fault: different numbers of microservice instances
    - configuration fault: environmental configurations
    - asynchronous interaction fault: asynchronous interaction sequences
- Executes different application versions with existing automated test cases
  - For each setting, executes the test cases for a given number (e.g., 10) of times.
  - get multiple trace instances under the same setting
- Collects trace logs

Figure 2: An Example of Trace Instance
Preparation of Training Corpus

- For each trace instance
  - Generate a trace-level training sample
- For each microservice in the trace instance
  - Generate a microservice-level training sample
Preparation of Training Corpus

- First extract microservice-level features
- Aggregate these microservice-level features into trace-level features
- Error status - 1 or 0
  - 0 indicates the trace instance has a latent error
Feature selection by correlation analysis. (> 0.3)

Table 3: The Pearson Correlation Coefficient of Microservice Level Features and Latent Errors

<table>
<thead>
<tr>
<th>Trace Types</th>
<th>Features</th>
<th>Configuration</th>
<th>Resource</th>
<th>Instance</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ML</td>
<td>CL</td>
<td>VS</td>
<td>MC</td>
</tr>
<tr>
<td>Trace Type 1</td>
<td>Instance</td>
<td>0.02</td>
<td>0.12</td>
<td>0.22</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>Config</td>
<td>0.61</td>
<td>0.20</td>
<td>0.11</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>Interaction</td>
<td>0.10</td>
<td>0.22</td>
<td>0.03</td>
<td>0.01</td>
</tr>
<tr>
<td>Trace Type 2</td>
<td>Instance</td>
<td>0.04</td>
<td>0.09</td>
<td>0.27</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>Config</td>
<td>0.23</td>
<td>0.67</td>
<td>0.20</td>
<td>0.33</td>
</tr>
<tr>
<td></td>
<td>Interaction</td>
<td>0.22</td>
<td>0.18</td>
<td>0.13</td>
<td>0.21</td>
</tr>
<tr>
<td>Trace Type 3</td>
<td>Instance</td>
<td>0.03</td>
<td>0.17</td>
<td>0.18</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>Config</td>
<td>0.70</td>
<td>0.13</td>
<td>0.13</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>Interaction</td>
<td>0.01</td>
<td>0.09</td>
<td>0.09</td>
<td>0.15</td>
</tr>
<tr>
<td>Trace Type 4</td>
<td>Instance</td>
<td>0.10</td>
<td>0.10</td>
<td>0.15</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>Config</td>
<td>0.39</td>
<td>0.74</td>
<td>0.19</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>Interaction</td>
<td>0.14</td>
<td>0.21</td>
<td>0.21</td>
<td>0.17</td>
</tr>
<tr>
<td>Trace Type 5</td>
<td>Instance</td>
<td>0.13</td>
<td>0.07</td>
<td>0.07</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>Config</td>
<td>0.60</td>
<td>0.16</td>
<td>0.22</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>Interaction</td>
<td>0.05</td>
<td>0.09</td>
<td>0.03</td>
<td>0.36</td>
</tr>
</tbody>
</table>
Training

- Feature selection by correlation analysis. (> 0.3)
- 3 machine learning techniques:
  - Random Forests (RF)
  - K-Nearest Neighbors (KNN)
  - Multi-Layer Perceptron (MLP)
Feature Extraction and Prediction

- The online prediction phase includes 2 steps
  - feature extraction
  - prediction step
Implementation

- 4 main components:
  - the fault injector - JavaParser 3.8
  - execution controller - TestNG 6.8
  - log processor - Kubernetes and Istio
  - Predictor - scikit-learn
Evaluation

- RQ1: How accurate is MEPFL in predicting latent errors, faulty microservices, and fault types of microservice applications?
- RQ2: How does the training coverage of trace types influence the prediction accuracy?
- RQ3: How effective is MEPFL when it is used to predict latent errors caused by real fault cases?
Evaluation

- RQ1: How accurate is MEPFL in predicting latent errors, faulty microservices, and fault types of microservice applications?
- For Sock Shop, inject 32 faults (including 10 multi-instance faults, 10 configuration faults, and 12 asynchronous interaction faults) into 10 trace types.
- For TrainTicket, inject 142 faults (including 46 multi-instance faults, 52 configuration faults, and 44 asynchronous interaction faults) into 86 trace types.
RQ1: How accurate is MEPFL in predicting latent errors, faulty microservices, and fault types of microservice applications?

### Table 4: Accuracy of Intra-Application Error Prediction and Fault Localization

<table>
<thead>
<tr>
<th>Methods</th>
<th>Sock Shop</th>
<th>TrainTicket</th>
<th>Fault Type</th>
<th>Fault Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Latent Error</td>
<td>Rec</td>
<td>Precision</td>
<td>F1</td>
</tr>
<tr>
<td>MEPFL-RF</td>
<td>0.949</td>
<td>0.997</td>
<td>0.973</td>
<td>0.015</td>
</tr>
<tr>
<td>MEPFL-KNN</td>
<td>0.961</td>
<td>0.997</td>
<td>0.978</td>
<td>0.013</td>
</tr>
<tr>
<td>MEPFL-MLP</td>
<td>0.982</td>
<td>0.998</td>
<td>0.990</td>
<td>0.009</td>
</tr>
<tr>
<td>Approach in [40]</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Table 5: Accuracy of Cross-Application Error Prediction and Fault Localization

<table>
<thead>
<tr>
<th>Methods</th>
<th>TrainTicket Model Predicting Sock Shop</th>
<th>Sock Shop Model Predicting TrainTicket</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LE</td>
<td>FM</td>
</tr>
<tr>
<td></td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>MEPFL-RF</td>
<td>0.441</td>
<td>0.510</td>
</tr>
<tr>
<td>MEPFL-KNN</td>
<td>0.461</td>
<td>0.530</td>
</tr>
<tr>
<td>MEPFL-MLP</td>
<td>0.451</td>
<td>0.562</td>
</tr>
</tbody>
</table>
Evaluation

- RQ2: How does the training coverage of trace types influence the prediction accuracy?
- MEPFL-MLP + TrainTicket

**Table 6: Influence of Trace Type Coverage**

<table>
<thead>
<tr>
<th>Coverage</th>
<th>TT</th>
<th>PT</th>
<th>T-Rate</th>
<th>Latent Error</th>
<th>Faulty Microservice</th>
<th>Fault Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Recall</td>
<td>Precision</td>
<td>F1</td>
</tr>
<tr>
<td>10%</td>
<td>8min37s</td>
<td>12s</td>
<td>0.329</td>
<td>0.771</td>
<td>0.807</td>
<td>0.789</td>
</tr>
<tr>
<td>20%</td>
<td>11min58s</td>
<td>12s</td>
<td>0.403</td>
<td>0.795</td>
<td>0.846</td>
<td>0.820</td>
</tr>
<tr>
<td>30%</td>
<td>14min15s</td>
<td>12s</td>
<td>0.591</td>
<td>0.824</td>
<td>0.916</td>
<td>0.868</td>
</tr>
<tr>
<td>40%</td>
<td>16min24s</td>
<td>12s</td>
<td>0.655</td>
<td>0.829</td>
<td>0.945</td>
<td>0.884</td>
</tr>
<tr>
<td>50%</td>
<td>19min58s</td>
<td>12s</td>
<td>0.662</td>
<td>0.884</td>
<td>0.951</td>
<td>0.897</td>
</tr>
<tr>
<td>60%</td>
<td>23min39s</td>
<td>12s</td>
<td>0.766</td>
<td>0.858</td>
<td>0.950</td>
<td>0.902</td>
</tr>
<tr>
<td>70%</td>
<td>27min26s</td>
<td>12s</td>
<td>0.797</td>
<td>0.848</td>
<td>0.954</td>
<td>0.898</td>
</tr>
<tr>
<td>80%</td>
<td>30min42s</td>
<td>12s</td>
<td>0.830</td>
<td>0.857</td>
<td>0.958</td>
<td>0.905</td>
</tr>
<tr>
<td>90%</td>
<td>35min35s</td>
<td>12s</td>
<td>0.813</td>
<td>0.879</td>
<td>0.950</td>
<td>0.913</td>
</tr>
</tbody>
</table>
Evaluation

- RQ3: How effective is MEPFL when it is used to predict latent errors caused by real fault cases?

<table>
<thead>
<tr>
<th>Fault Cases</th>
<th>Type</th>
<th>#TR</th>
<th>#ET</th>
<th>LE Recall</th>
<th>LE Precision</th>
<th>LE F1</th>
<th>LE FPR</th>
<th>FM Top1</th>
<th>FM Top3</th>
<th>FM Top5</th>
<th>FT Recall</th>
<th>FT Precision</th>
<th>FT F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>AI</td>
<td>306</td>
<td>61</td>
<td>0.983</td>
<td>0.803</td>
<td>0.884</td>
<td>0.033</td>
<td>0.951</td>
<td>1.000</td>
<td>1.000</td>
<td>0.819</td>
<td>0.655</td>
<td>0.728</td>
</tr>
<tr>
<td>F2</td>
<td>AI</td>
<td>322</td>
<td>65</td>
<td>0.738</td>
<td>0.984</td>
<td>0.843</td>
<td>0.031</td>
<td>0.907</td>
<td>1.000</td>
<td>1.000</td>
<td>0.815</td>
<td>0.846</td>
<td>0.830</td>
</tr>
<tr>
<td>F3</td>
<td>C</td>
<td>310</td>
<td>48</td>
<td>0.667</td>
<td>0.979</td>
<td>0.793</td>
<td>0.042</td>
<td>0.291</td>
<td>1.000</td>
<td>1.000</td>
<td>0.750</td>
<td>0.771</td>
<td>0.760</td>
</tr>
<tr>
<td>F4</td>
<td>C</td>
<td>312</td>
<td>37</td>
<td>0.783</td>
<td>0.918</td>
<td>0.845</td>
<td>0.026</td>
<td>0.486</td>
<td>0.513</td>
<td>1.000</td>
<td>0.918</td>
<td>0.972</td>
<td>0.944</td>
</tr>
<tr>
<td>F5</td>
<td>C</td>
<td>328</td>
<td>89</td>
<td>0.685</td>
<td>0.971</td>
<td>0.805</td>
<td>0.034</td>
<td>0.977</td>
<td>0.977</td>
<td>1.000</td>
<td>0.943</td>
<td>0.977</td>
<td>0.960</td>
</tr>
<tr>
<td>F7</td>
<td>C</td>
<td>392</td>
<td>34</td>
<td>0.647</td>
<td>0.882</td>
<td>0.746</td>
<td>0.028</td>
<td>0.558</td>
<td>1.000</td>
<td>1.000</td>
<td>0.676</td>
<td>0.823</td>
<td>0.742</td>
</tr>
<tr>
<td>F8</td>
<td>MI</td>
<td>343</td>
<td>75</td>
<td>0.960</td>
<td>0.586</td>
<td>0.728</td>
<td>0.041</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.840</td>
<td>0.826</td>
<td>0.833</td>
</tr>
<tr>
<td>F11</td>
<td>MI</td>
<td>328</td>
<td>40</td>
<td>0.975</td>
<td>0.875</td>
<td>0.922</td>
<td>0.024</td>
<td>0.875</td>
<td>0.975</td>
<td>0.975</td>
<td>0.975</td>
<td>0.975</td>
<td>0.975</td>
</tr>
<tr>
<td>F12</td>
<td>MI</td>
<td>369</td>
<td>37</td>
<td>0.837</td>
<td>0.918</td>
<td>0.876</td>
<td>0.027</td>
<td>0.837</td>
<td>1.000</td>
<td>1.000</td>
<td>0.702</td>
<td>0.648</td>
<td>0.674</td>
</tr>
<tr>
<td>F13</td>
<td>AI</td>
<td>322</td>
<td>64</td>
<td>0.937</td>
<td>0.984</td>
<td>0.960</td>
<td>0.016</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.843</td>
<td>0.703</td>
<td>0.767</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>333</td>
<td>55</td>
<td>0.821</td>
<td>0.891</td>
<td>0.854</td>
<td>0.030</td>
<td>0.788</td>
<td>0.947</td>
<td>0.998</td>
<td>0.828</td>
<td>0.820</td>
<td>0.824</td>
</tr>
</tbody>
</table>
Evaluation - Threats to Validity

- 3 major threats to internal validity
  - correctness of fault injection and trace labeling
  - Unknown problems within the two benchmark applications
  - Uncertainties of automatic deployment and runtime management

- 2 major threats to external validity
  - limitation of two target applications
  - limitation of the fault cases used in the studies
Conclusion

MEPFL

Support 3 types of fault.

Features extract from trace logs.

Offline training

Online prediction

Figure 3: Approach Overview
QA

- More than one fault in invocation?
- Utility in industry?
- What is faulty version?
- Incorporate online training into off-line training?
Advances and Challenges in Log Analysis

Adam Oliner, UC Berkeley; Archana Ganapathi, Splunk; Wei Xu, Google
Why do we need logs?

Many Different Applications

- troubleshooting
- studying traffic patterns to maximize advertising revenue
- detecting a malicious intrusion

But there are many different **CHALLENGES** as well!
1. Debugging

As Brian Kernighan wrote in *Unix for Beginners* in 1979, "The most effective debugging tool is still careful thought, coupled with judiciously placed print statements."

Logs can be used to locate specific messages or patterns

- Difficult to figure out what to search for

Machine-learning techniques, especially anomaly detection, are commonly used

- Log volume can be excessive in a large system.
- Messages from multiple sources, however, may be interleaved.
2. Performance

Understanding how the resources in that system are used.

- Synthesizing logs from multiple sources can be challenging.
- Extensive logging can consume resources.
- Sampling may miss rare events.
3. Security

Logs can be used in detecting breaches or misbehavior.

Intrusion detection often requires reconstructing sessions from logs.

- **But**, an analysis is limited by the information in the logs.

"healthy" log Vs a "suspicious" one.
4. Reporting and Profiling

Log analysis can be used to profile resource utilization, workload, or user behavior.

Profiling can be used for business analytics.

- Statistical techniques like clustering can be used for this
- Classifying log events is often challenging
- How to interpreting profiles?
Logging Infrastructure

Two features:

- Log Generation
- Log Storage

Mostly, logs are just unstructured text

- serializing variables into text is expensive
- Parsing these logs can be complicated
Logging Infrastructure

Choosing the right log-storage solution involves the following tradeoffs:

- Cost
- Total Capacity
- Persistence guarantees
- Write-access characteristics
- Read-access characteristics
- Integration with existing infrastructure

Log-storage solutions with alerting and reporting capabilities.
Summary

System management has become log-centric.

Log-analysis techniques have made much progress

Several challenges remain

● Correlating distributed logs
● Minimize instrumentation overhead while maximizing information content.
● Getting actionable insights
Future Directions

Advances in log visualization techniques

Modifying existing instrumentation techniques

Exploring what logs to use, when, and for what purpose.

Designing better logging standards and best practices.
Three Pillars with Zero Answers: Rethinking Observability with Ben Sigelman
The 3 Pillars

Metrics
Logs
Traces

But, there are some Fatal Flaws!
Metrics

A squiggly line helps in drilling down to the problem.

But,

many real-world tags have thousands or millions of values.
The problem is simple, too expensive!

"Don't log anything anymore"

application transaction rate
all microservices
cost of network and storage
weeks of data retention

way, way too much $$$$
Traces

Tracing - a technology specifically developed to address the above problem with logging systems.

But,

Sampling the right distributed traces and extracting meaningful, accessible insights is a broader challenge.
And what about emulating Google?

they've done some terrific work

But,

Google's technologies are built to scale like crazy

system's scalability ↔ its feature set.
How to think about Observability?

Goals

1. Gradually improving an SLI
2. Rapidly restoring an SLI

Activities

1. Measuring SLIs precisely
   a. Specificity - Fidelity - Freshness
2. Refinement
   a. Identifying variance - explaining variance
Designing an Observability System

We can choose three out of these 4

- High-throughput
- High-Cardinality
- Lengthy retention window
- Avoiding Sampling
Questions?