Towards Reliable Cloud Microservices with Intelligent Operations

Ph.D. Oral Defense of Tianyi Yang
Supervised by Prof. Michael Lyu
Wednesday 17 August 2022
Online Cloud Services Are Everywhere

To-Consumer services

- facebook
- NETFLIX
- bilibili

To-Business services

- salesforce
- slack
- SAP

Cloud services

- AWS
- Azure
- Alibaba Cloud
- HUAWEI CLOUD
Online Cloud Services’ Reliability Is Crucial

-- to both **service providers** and **end users**!
Real-world Examples

Facebook outage: what went wrong and why did it take so long to fix after social platform went down? | Facebook | The Guardian

Extended AWS outage disrupts services across the globe | Fierce Telecom

Towards Reliable Cloud Microservices with Intelligent Operations
Service Outage!
Reliability management of online services is important, but challenging, due to the increasing complexity and distributed nature of online services.
CONTENTS

1. Background and Contributions
2. Predicting the Intensity of Dependency
4. Empirical Study on Alerting and Logging
5. Conclusion and Future Work
Background and Contributions
Microservices architecture is an approach in which a single application is composed of many \textit{loosely coupled} and \textit{independently deployable} small programs.
Online Service Systems Shift to Microservices

• Microservices collectively comprise multiple cloud services.
  • *Online services*: provide high-level APIs.
  • *Microservices*: collectively handle the external request via multiple chained invocations.

• Minor anomalies may magnify impact and escalate into system outages!

Loosely-coupled nature makes failure diagnosis difficult.
Microservices Generates a Variety of Data

- **Application Layer**
  - Application
  - Microservice
  - Function

- **Platform Layer**
  - Container
  - Orchestration
  - Database

- **Infrastructure Layer**
  - Compute
  - Networking
  - Storage
  - Virtual Machine
  - Physical Machine

**Logs**

**Metrics**

**Traces (Topology)**

**Alerts**

Towards Reliable Cloud Microservices with Intelligent Operations
Traces

- Tracks the processing of each request.
- Terminologies
  - Span log (abbr. span): a log recording the contextual information of each service invocation.
  - Trace log (abbr. trace): all the spans that serve for the same request.

![Service invocations for a request.](image)

<table>
<thead>
<tr>
<th>Span ID</th>
<th>e22f30bdfbd09134</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Span ID</td>
<td>b42a04bf1897d5d</td>
</tr>
<tr>
<td>Name</td>
<td>ts-preserve-service</td>
</tr>
<tr>
<td>Timestamp (µs)</td>
<td>1618580908705000</td>
</tr>
<tr>
<td>Duration (µs)</td>
<td>1126</td>
</tr>
<tr>
<td>Result</td>
<td>SUCCESS</td>
</tr>
<tr>
<td>Trace ID</td>
<td>e0d17d481f47b9d9</td>
</tr>
<tr>
<td>Additional Logs</td>
<td>...</td>
</tr>
</tbody>
</table>

A span generated by the train-ticket benchmark.

A trace with 6 spans.
Monitoring Metrics

• Monitoring Metrics
  • Observes real-time statuses of microservice systems.
  • Timestamped data with fixed intervals.

• Terminologies
  • System performance metrics.
    • E.g., CPU usage, memory usage, NIC send/receive rate.
  • Business metrics.
    • E.g., Request latency, request error rate, and throughput.
Logs & Alerts

• Logs
  • Semi-structured text printed by logging statements (e.g., `printf()`, `logger.info()`).

```
1 2008-11-09 20:55:54 PacketResponder 0 for block blk_321 terminating
2 2008-11-09 20:55:54 Received block blk_321 of size 67108864 from /10.251.195.70
3 2008-11-09 20:55:54 PacketResponder 2 for block blk_321 terminating
4 2008-11-09 20:55:54 Received block blk_321 of size 67108864 from /10.251.126.5
6 2008-11-10 03:58:04 Verification succeeded for blk_321
7 2008-11-10 10:36:37 Deleting block blk_321 file /mnt/ hadoop/dfs/data/current/subdir1/blk_321
8 2008-11-10 10:36:50 Deleting block blk_321 file /mnt/ hadoop/dfs/data/current/subdir51/blk_321
```

• Alerts
  • Structured text notifications to call for immediate human intervention upon system anomalies.

<table>
<thead>
<tr>
<th>No.</th>
<th>Severity</th>
<th>Time</th>
<th>Service</th>
<th>Alert Title</th>
<th>Duration</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Major</td>
<td>2021/05/18 06:36</td>
<td>Block Storage</td>
<td>Failed to allocate new blocks, disk full</td>
<td>10 min</td>
<td>Region=X;DC=1;...</td>
</tr>
<tr>
<td>2</td>
<td>Critical</td>
<td>2021/05/18 06:38</td>
<td>Database</td>
<td>Failed to commit changes ...</td>
<td>2 min</td>
<td>Region=X;DC=1;...</td>
</tr>
<tr>
<td>3</td>
<td>Critical</td>
<td>2021/05/18 06:39</td>
<td>Database</td>
<td>Failed to commit changes ...</td>
<td>5 min</td>
<td>Region=X;DC=1;...</td>
</tr>
</tbody>
</table>

Alerts need to be promptly dealt with, but logs do not.
Thesis Contribution

Towards Reliable Cloud Microservices with Intelligent Operations

➢ The first empirical study on the intensity of dependency.
➢ The first method to quantify the intensity of microservice dependencies.
➢ Release an industrial dataset for reuse.

[ASE’21]

➢ The first empirical study on the failures of resilient and unresilient microservices.
➢ The first self-adaptive resilience testing framework.

[ICSE’23]*

➢ Identify six antipatterns of alerts in a production cloud.
➢ Identify four postmortem reactions to antipatterns.
➢ Survey the current practice of logging for reliability.
➢ Propose directions on improving the quality of alerts and logs.

[DSN’22, WWW’21]

* Under review by ICSE’23
Thesis Contribution

➢ The first empirical study on the intensity of dependency.
➢ The first method to quantify the intensity of microservice dependencies.
➢ Release an industrial dataset for reuse.

[ASE’21]

Chapter 3

Prediction of the Intensity of Dependency

Intelligent Operations for Reliable Microservices

Traces

Metrics

Self-adaptive Resilience Testing

Alerts

Quality of Alerts

Logs

Logging Practices

➢ The first empirical study on the failures of resilient and unresilient microservices.
➢ The first self-adaptive resilience testing framework.

➢ Identify six antipatterns of alerts in a production cloud.
➢ Identify four postmortem reactions to antipatterns.
➢ Survey the current practice of logging for reliability.
➢ Propose directions on improving the quality of alerts and logs.
The first empirical study on the intensity of dependency.
➢ The first method to quantify the intensity of microservice dependencies.
➢ Release an industrial dataset for reuse.

The first empirical study on the failures of resilient and unresilient microservices.
The first self-adaptive resilience testing framework.

Identify six antipatterns of alerts in a production cloud.
Identify four postmortem reactions to antipatterns.
Survey the current practice of logging for reliability.
Propose directions on improving the quality of alerts and logs.

Chapter 4

Intelligent Operations for Reliable Microservices

Traces
Metrics
Alerts
Logs

Self-adaptive Resilience Testing
Quality of Alerts
Logging Practices

[ICSE’23]*

* Under review by ICSE’23
Thesis Contribution

➢ The first empirical study on the intensity of dependency.
➢ The first method to quantify the intensity of microservice dependencies.
➢ Release an industrial dataset for reuse.

➢ The first empirical study on the failures of resilient and unresilient microservices.
➢ The first self-adaptive resilience testing framework.

Towards Reliable Cloud Microservices with Intelligent Operations

[DSN’22, WWW’21] [CSUR’21]
2 Predicting the Intensity of Dependency
A Survey of the Outages in AWS

AWS Post-Event Summaries

The following is a list of post-event summaries from major service events that impacted AWS service availability:

- Summary of the Amazon Kinesis Event in the Northern Virginia (US-EAST-1) Region, November, 25th 2020
- Summary of the Amazon EC2 and Amazon EBS Service Event in the Tokyo (AP-NORTHEAST-1) Region, August 23, 2019
- Summary of the Amazon EC2 DNS Resolution Issues in the Asia Pacific (Seoul) Region (AP-NORTHEAST-2), November 24, 2018.
- Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region, February 28, 2017.
- Summary of the AWS Service Event in the Sydney Region, June 8, 2016.
- Summary of the Amazon DynamoDB Service Disruption and Related Impacts in the US-East Region, September 20, 2015.
- Summary of the Amazon EC2, Amazon EBS, and Amazon RDS Service Event in the EU West Region, August 7, 2014.
- Summary of the Amazon SimpleDB Service Disruption, June 13, 2014.
- Summary of the December 17th event in the South America Region (SA-EAST-1), December 20, 2013.
- Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region, April 29, 2011.

5 out of 13 Amazon Web Service (AWS) outages are related to service dependency!
AWS Kinesis Event on Nov 25\textsuperscript{th}, 2020

Summary of the Amazon Kinesis Event in the Northern Virginia (US-EAST-1) Region

- Upgrade Failure (dep)
- AWS CloudWatch (Customer Dashboard) Failure (dep)
- AWS Cognito (API Usage Analysis) Failure (bug & dep)

Reduced dependency can accelerate failure recovery.

[Northern Virginia (US-EAST-1) Region]
Drawbacks of Current Failure Diagnosis Methods

Current practice is inefficient and dependent on the human experience.

* OCE: On-Call Engineer

Because each team only have a local view of the whole system.
The intensity of dependency between $A \to B$ is higher than the intensity of dependency between $A \to C$, due to

- Functionality
- Fault tolerance
Intensity of Service Dependency

We define the intensity of dependency between two services as how much the status of the callee service influences the status of the caller service.

• Intensity is inherently determined by the program logic of services.
• Manual maintenance of intensity is hard due to the fast-evolving nature.
• But we could predict the intensity of dependency from traces.
AID: Predicting the Aggregated Intensity of Dependency

- Raw Traces → Candidate Selection
- Service Status Generation
- Status Series of Services
- Candidate Dependency List
- Intensity Prediction
- Dependency Graph with Intensity
**AID: Candidate Selection**

- **Objective**
  - Select the candidate invocation pairs \((\text{caller}, \text{callee})\) from raw traces where \text{caller} directly invokes \text{callee}.

- **Method**
  - Iterate over all spans to get the invocation pairs.
  - Get the invocation pairs if the cloud system have a centralized database of invocation.
AID: Service Status Series Generation

- Three aspects of indicators of service status
  - Number of Invocations
  - Durations of Invocations
  - Error of Invocations

- Method: calculate the number of invocations, average duration, and error rate of all spans of a service in a fixed time interval (e.g., 1 minute).
AID: Intensity Prediction

- Idea: the more similar two services’ status series are, the higher the intensity is.

- Method
  - Dynamic Status Warping.
  - Similarity Normalization & Aggregation.

Algorithm 1: Dynamic Status Warping

**Input:** The status series of caller service and callee service $status^C$, $status^C$; duration series of callee $dur^C$, estimated round trip time $\delta_{rtt}$, max time drift $\delta_d$

**Output:** The similarity between two status series

1. Set the warping window $w = \max(dur^C) + \delta_{rtt}$
2. $K = \text{length}(status^C)$
3. $N = \text{length}(status^P)$
4. Initialize the cost matrix $C \in \mathbb{R}^{K \times N}$, set the initial values as $+\infty$
5. $C_{1,1} = (status^P_1 - status^C_1)^2$
6. for $i = 2 \ldots \min(\delta_d, K)$ do // Initialize the first column
   7. $C_{i,1} = C_{i-1,1} + (status^P_i - status^C_1)^2$
7. end
8. for $j = 2 \ldots \min(w + \delta_d, N)$ do // Initialize the first row
   9. $C_{1,j} = C_{1,j-1} + (status^P_1 - status^C_j)^2$
9. end
10. for $i = 2 \ldots K$ do
    11. for $j = \max(2, i - \delta_d) \ldots \min(N, i + w + \delta_d)$ do
        12. $C_{i,j} = \min(C_{i-1,j-1}, C_{i-1,j}, C_{i,j-1}) + (status^P_i - status^C_j)^2$
    13. end
14. end
15. end
16. return $C_{K,N}$
Overview

Experiment Settings

- **Dataset**
  - *TT*: Simulated traces by the Train-Ticket benchmark.

- **Manual labeling**
  - *Industry*: By engineers of Huawei Cloud.
  - *TT*: By two PhD students familiar with the benchmark.

### Dataset Statistics

<table>
<thead>
<tr>
<th>Dataset</th>
<th>TT</th>
<th>Industry</th>
</tr>
</thead>
<tbody>
<tr>
<td># Microservices</td>
<td>25</td>
<td>192</td>
</tr>
<tr>
<td># Spans</td>
<td>17,471,024</td>
<td>About 1.0e10</td>
</tr>
<tr>
<td># Strong</td>
<td>18</td>
<td>67</td>
</tr>
<tr>
<td># Weak</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>

---

1 We only labeled 75 dependencies that the engineers are familiar with.
2 FudanSELab/train-ticket: Train Ticket - A Benchmark Microservice System (github.com)
Effectiveness of Intensity Prediction

Performance Comparison of Different Methods on Two Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>Metric</th>
<th>CE</th>
<th>MAE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>Pearson</td>
<td>0.6872</td>
<td></td>
<td>0.3305</td>
<td>0.4388</td>
</tr>
<tr>
<td></td>
<td>Spearman</td>
<td>0.7512</td>
<td></td>
<td>0.3735</td>
<td>0.4697</td>
</tr>
<tr>
<td></td>
<td>Kendall</td>
<td>0.6464</td>
<td></td>
<td>0.3749</td>
<td>0.4577</td>
</tr>
<tr>
<td></td>
<td>AID</td>
<td><strong>0.4562</strong></td>
<td></td>
<td>0.3435</td>
<td><strong>0.3859</strong></td>
</tr>
<tr>
<td>Industry</td>
<td>Pearson</td>
<td>0.6076</td>
<td></td>
<td>0.4524</td>
<td>0.4563</td>
</tr>
<tr>
<td></td>
<td>Spearman</td>
<td>0.6030</td>
<td></td>
<td>0.4501</td>
<td>0.4537</td>
</tr>
<tr>
<td></td>
<td>Kendall</td>
<td>0.6258</td>
<td></td>
<td>0.4636</td>
<td>0.4656</td>
</tr>
<tr>
<td></td>
<td>AID</td>
<td><strong>0.3270</strong></td>
<td></td>
<td><strong>0.1751</strong></td>
<td><strong>0.3044</strong></td>
</tr>
</tbody>
</table>

- **Parameter Settings**
  - Bin size $\tau = 1 \text{ min}$
  - Estimated round trip time $\delta_{rtt} = 0$
  - Max time drift
    - $\delta_d = 1 \text{ min}$ (for Industry dataset)
    - $\delta_d = 0 \text{ min}$ (for TT dataset)
Ablation Study

The impact of different similarity measures

<table>
<thead>
<tr>
<th>Dataset /Bin size</th>
<th>Method</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT /1min</td>
<td>$AID_{DSW}$</td>
<td>0.4562</td>
</tr>
<tr>
<td></td>
<td>$AID_{DTW}$</td>
<td>0.4494</td>
</tr>
<tr>
<td>Industry /1min</td>
<td>$AID_{DSW}$</td>
<td>0.3270</td>
</tr>
<tr>
<td></td>
<td>$AID_{DTW}$</td>
<td>0.3584</td>
</tr>
</tbody>
</table>
• Mitigation of Cascading Failures
  • Limit the traffic to critical cloud services.
  • Recover the dependencies marked as “strong” first.

• Optimization of Dependencies
  • Dependency management system detects strong dependencies and reminds engineers.
  • Discovered more than ten unnecessary dependencies within four months.
Summary of Chapter 3

**First** to identify the concept of aggregated intensity of dependency for failure diagnosis and failure recovery.

First method to quantify the intensity of dependencies between different services.

Experiments on simulated & industrial datasets show its effectiveness and efficiency.

Successfully deployed in Huawei Cloud.
CONTENTS

1. Background and Contributions
2. Predicting the Intensity of Dependency
4. Empirical Study on Alerting and Logging
5. Conclusion and Future Work
Self-adaptive Microservice Resilience Testing
Resilience: the ability to maintain performance at an acceptable level and recover the service back to normal under service failures.
Current Practice for Resilience Testing

- Set rules
- Inject failures
- Evaluate results

<table>
<thead>
<tr>
<th>Failure type</th>
<th>Network jam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metrics to monitor</td>
<td>Rx_bytes, tx_bytes, throughput</td>
</tr>
<tr>
<td>Passing criteria</td>
<td>Request throughput recover within 5 minutes</td>
</tr>
</tbody>
</table>

An example rule set

Monolith vs. Microservices

Normal Period  Failure Injection Period
Issues of Current Practice: Scalability

• Scalability Issue
  • Manual identification of the failure rule sets relies heavily on domain expertise.
  • Fast-evolving nature of microservices requires frequent updates of failure rule sets.

Manual identification of failure rule sets does not scale.
Issues of Current Practice: Adaptivity

• Adaptivity Issue
  • PASS/FAIL cannot depict the subtle difference in an online service’s resilience.
  
• Reasons
  • The impact of a failure is diversiform in a microservice system.
  • Online services can be in a gray-failure status instead of fail as a whole.

Defining fixed failure rule sets for evaluating resilience is inadaptive.
Characteristics of Resilient Microservices

• Inject failures into two deployments of the same microservice benchmark system.
  • One with common resilience measures
  • One without common resilience measures

• Compare the manifestation of failures on the two deployments.
Characteristics of Resilient Microservices

• Service degradation manifests the impact of the injected failures.
  • Measured by the performance difference between the normal period and the fault-injection period.

• Insight
  • The less degradation propagation from system performance metrics to business metrics,
  • The higher the resilience.

<table>
<thead>
<tr>
<th>Failure</th>
<th>Degradation w/o resilience mechanisms</th>
<th>Degradation w/ resilience mechanisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container CPU overload</td>
<td>High container CPU usage, slow response speed</td>
<td>Decreased but acceptable response speed</td>
</tr>
<tr>
<td>Container TCP disconnection</td>
<td>Connection error within container</td>
<td>Return to normal response speed shortly</td>
</tr>
<tr>
<td>Container instance killed</td>
<td>Instance offline, unresponsive microservice endpoint</td>
<td>Response normally after some time</td>
</tr>
</tbody>
</table>

(More in the thesis) ……
AVERT: A Self-adaptive Resilience Testing Framework

- **Failure Execution**
  - Load Generation
  - Failure Injection
  - Metric Lattice Construction
  - Failure Clearance

- **Degradation-based Metric Lattice Search**
  - Degradation-based Metric Selection
  - Metric Lattice Search
  - Ranked Metrics

- **Resilience Indexing**
  - Resilience Indexing

**Steps**
- Collect monitoring metrics.
- Rank the metrics by degradation.
- Index the resilience.
• Two phases for each type of failure.
  • Failure injection & Failure clearance.

• Data collected
  • Two types of metrics
    • Business metrics $B$
    • System performance metrics $P$

• Denote all metrics as $M$
  $$M = B \cup P = \{m_1, m_2, ..., m_M\}$$
  $$\exists i, m_i \in B \vee m_i \in P$$
AVERT: Degradation-based Metric Lattice Search

- Construct the metric lattice from the power set of $M$.
  - Each node is a subset of $M$.
  - Ordered by the subset-superset relation.

![Diagram of Degradation-based Metric Lattice Search]

- Metric Lattice Construction
- Degradation-based Metric Selection
- Ranked Metrics
AVERT: Degradation-based Metric Lattice Search

- Idea
  - Depth-first search from the upmost node to the bottommost node.
  - Select the metric that contributes most to the overall service degradation.

Please check the detailed algorithm in the thesis.
AVERT: Resilience Indexing

- **Idea**
  - If the degradation of system performance metrics cannot *propagate* to the degradation of business metrics, the resilience is higher.

- **Approach**
  - Calculate the degradation contributed by $B$ and $P$.
    \[
    D_P = \sum_{m_i \in P} \frac{c_i}{\log_2(rank(m_i; L) + 1)}
    \]
    \[
    D_B = \sum_{m_i \in B} \frac{c_i}{\log_2(rank(m_i; L) + 1)}
    \]
  - Calculate the propagation.
    \[
    r = \frac{1}{1 + e^{D_B - D_P}}
    \]
Experiment Settings

• Dataset
  • $TT^1$
    • The Train-Ticket benchmark
    • Env: Kubernetes
    • No. of failures: 24
  • $SN^2$
    • The Social-Network benchmark
    • Env: docker compose
    • No. of failures: 10

• Manual labeling of resilience
  • Done by two PhD students.
  • Verified by experienced engineers of Huawei.

<table>
<thead>
<tr>
<th>Dataset</th>
<th></th>
<th></th>
<th>#Microservices</th>
<th>#Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train-Ticket</td>
<td>30</td>
<td>209</td>
<td>15</td>
<td>24</td>
</tr>
<tr>
<td>Social-Network</td>
<td>50</td>
<td>325</td>
<td>25</td>
<td>10</td>
</tr>
</tbody>
</table>

1 FudanSELab/train-ticket: Train Ticket - A Benchmark Microservice System (github.com)
2 delimitrou/DeathStarBench: Open-source benchmark suite for cloud microservices (github.com)
**Effectiveness**

Table 4.3: Performance Comparison of AVERT on Two Datasets

<table>
<thead>
<tr>
<th>Method</th>
<th>Train-Ticket</th>
<th></th>
<th>Social-Network</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CE</td>
<td>MAE</td>
<td>RMSE</td>
<td>CE</td>
<td>MAE</td>
</tr>
<tr>
<td>SVC</td>
<td>0.8864</td>
<td>0.4875</td>
<td>0.5594</td>
<td>0.7483</td>
<td>0.4426</td>
</tr>
<tr>
<td>RF</td>
<td>0.6973</td>
<td>0.4259</td>
<td>0.5005</td>
<td>0.5646</td>
<td>0.3787</td>
</tr>
<tr>
<td>ET</td>
<td>0.8766</td>
<td>0.4682</td>
<td>0.5470</td>
<td>0.6546</td>
<td>0.4199</td>
</tr>
<tr>
<td>AVERT</td>
<td>0.1775</td>
<td>0.1572</td>
<td>0.1842</td>
<td>0.1159</td>
<td>0.1078</td>
</tr>
</tbody>
</table>

Table 4.4: Ablation Study of AVERT on Two Datasets

<table>
<thead>
<tr>
<th>Method</th>
<th>Train-Ticket</th>
<th></th>
<th>Social-Network</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CE</td>
<td>MAE</td>
<td>RMSE</td>
<td>CE</td>
<td>MAE</td>
</tr>
<tr>
<td>AVERT-euc</td>
<td>0.3379</td>
<td>0.2735</td>
<td>0.3067</td>
<td>0.1874</td>
<td>0.1655</td>
</tr>
<tr>
<td>AVERT-corr</td>
<td>0.2320</td>
<td>0.1985</td>
<td>0.2296</td>
<td>0.2532</td>
<td>0.2148</td>
</tr>
<tr>
<td>AVERT-cid</td>
<td>0.1784</td>
<td>0.1589</td>
<td><strong>0.1810</strong></td>
<td>0.3131</td>
<td>0.2542</td>
</tr>
<tr>
<td>AVERT-dtw</td>
<td><strong>0.1775</strong></td>
<td><strong>0.1572</strong></td>
<td>0.1842</td>
<td><strong>0.1159</strong></td>
<td><strong>0.1078</strong></td>
</tr>
</tbody>
</table>
Use Cases of AVERT

- Automatic Resilience Indexing
- Selection of the Vulnerable Metrics
Summary of Chapter 4

An empirical study to demonstrate the feasibility of self-adaptive resilience testing for microservice systems.

First self-adaptive resilience testing framework, AVERT, that can automatically index the resilience of a microservice system to different failures.

AVERT measures the degradation propagation from system performance metrics to business metrics. The higher the propagation, the lower the resilience.

Evaluation on two open-source benchmark microservice systems indicates the effectiveness and efficiency.
CONTENTS

1. Background and Contributions

2. Predicting the Intensity of Dependency


4. Empirical Study on Alerting and Logging

5. Conclusion and Future Work
Empirical Study on Alerting and Logging
Why the Quality of Alerts and Logs Matters?

• Logs and alerts are important for reliability assurance.

• But the generation and processing of alerts are highly empirical.
Anti-patterns of Alerts in Cloud Systems

Quantitative inspection of 4 million alerts in 2 years + Interviews with 18 OCEs.

• Individual anti-patterns
  • [A1] Unclear Name or Description.
  • [A3] Improper and Outdated Generation Rule.

• Collective anti-patterns

Impact of Anti-patterns
An example Standard Operation Procedure:

<table>
<thead>
<tr>
<th>Description</th>
<th>CPU usage of nginx instance is higher than 80%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generation Rule</td>
<td>Continuously check the CPU usage of nginx instance, generate the alert when usage is higher than 80%.</td>
</tr>
<tr>
<td>Potential Impact</td>
<td>Affects the forwarding of all requests.</td>
</tr>
<tr>
<td>Possible Causes</td>
<td>a) The workload is too high. b) ........</td>
</tr>
<tr>
<td>Steps to Diagnose</td>
<td>Step 1: execute command <code>top -bn1</code> in the instance. Step 2: ........</td>
</tr>
</tbody>
</table>

Answers to “Overall Helpfulness” regarding OCEs’ working experience.
Reactions to Anti-patterns

• Reactions
  • [R1] Alert Blocking.
  • [R2] Alert Aggregation.
Automatic Evaluation of the Quality of Alerts

- Criteria to measure the quality of alerts
  - Indicativeness
  - Precision
  - Handleability

- Incorporating human knowledge and machine learning to evaluate the three aspects of alerts
Mechanism of Logging

```java
public void setTemperature(Integer temperature) {
    // ...
    logger.debug("Temperature set to {}. Old temperature was {}.", t, oldT);
    if (temperature.intValue() > 50) {
        logger.info("Temperature has risen above 50 degrees.");
    }
}
```

Output:
```
0 [setTemperature] DEBUG Wombat - Temperature set to 61. Old temperature was 42.
0 [setTemperature] INFO Wombat - Temperature has risen above 50 degrees.
```
Challenges for Logging

Challenges

Where to log

What to log

How to log

Meaning

Determining the appropriate location of logging statements.

Providing sufficient and concise *verbosity level, static text, and dynamic content.*

The systematical design pattern and maintenance of logging statements.

Aspects

Each challenge exhibits one or more aspects.

Diagnosability

Maintenance

Performance
Where to log

Aspects

- Diagnosability
- Performance

Objectives

- Minimize or reduce overhead
- Suggest appropriate placement of logging statements into source code
- Study logging practice in industry
What to log

**Aspects**

- **Diagnosability**
  - Enhance existing logging code to aid debugging
  - Suggest proper variables and text description in log
  - Determine whether a logging statement is likely to change in the future
- **Maintenance**
  - Study logging practice in industry
- **Performance**
  - Study the performance overhead and energy impact of logging in mobile app
  - Automatically change the log level of a system in case of anomaly

**Objectives**
How to log

Aspects

Diagnosability

- Characterize the anti-patterns in the logging code
- Optimize the implementation of logging mechanism to facilitate failure diagnosis
- Determine whether a logging statement is likely to change in the future
- Characterize and detect duplicate logging code
- Characterize and detect the anti-patterns in the logging code
- Characterize and prioritize the maintenance of logging statements
- Study the relationship between logging characteristics and the code quality
- Propose new abstraction or programming paradigm of logging

Maintenance

Performance

Objectives
Improving the Quality of Logs

- Prospective Directions
  - Analysis-Oriented Logging
  - Automated Generation of Logging Statements

- Best Practices for Logging
  - Always follow the logging standards
  - Keep proper quantity of log messages
First empirical study on characterizing and mitigating anti-patterns of alerts in an industrial cloud system.

Identify four individual anti-patterns, two collective anti-patterns, and four postmortem reactions.

Propose directions on improving the quality of alerts and logs.
Predicting the Intensity of Dependency

Self-adaptive Microservice Resilience Testing

Empirical Study on Alerting and Logging

Conclusion and Future Work
Conclusion and Future Work
Conclusion

Towards Reliable Cloud Microservices with Intelligent Operations

➢ The first empirical study on the intensity of dependency.
➢ The first method to quantify the intensity of microservice dependencies.
➢ Release an industrial dataset for reuse.

➢ The first empirical study on the failures of resilient and unresilient microservices.
➢ The first self-adaptive resilience testing framework.

➢ Identify six antipatterns of alerts in a production cloud.
➢ Identify four postmortem reactions to antipatterns.
➢ Survey the current practice of logging for reliability.
➢ Propose directions on improving the quality of alerts and logs.

[ASE'21] [ICSE'23]* [DSN'22, WWW'21] [CSUR'21]

* Under review by ICSE'23
Future Work

Multiple data type

- Traces
- Metrics
- Alerts
- Logs

Fusing Multiple Data Sources for Intelligent Operations

Single data type

- Trace Compression based on Service Topology
- Human-in-the-loop Auto QoA Evaluation
- Automated Generation of Logging Statements
- Analysis-Oriented Logging

WHAT'S NEXT


• (Under review, 1st author) AVERT: A Self-adaptive Resilience Testing Framework for Microservice Systems

• (Under review, 1st author) Managing Service Dependency for Cloud Reliability: The Industrial Practice

• (Under review, 2nd author) Eadro: Integrating Anomaly Detection and Root Cause Localization on Multi-source Monitoring Data for Microservice

• (Under review, 2nd author) HADES: Heterogeneous Anomaly Detection for Software Systems via Attentive Multi-modal Learning

• (Under review, 4th author) ScaleStore: Scalable and Fault Tolerant Key-Value Store on Disaggregated Memor
Thank you!
Categorization of the Research Thesis

- **Metrics**: AVERT (§4)
- **Traces**: AID (§3)
- **Alerts**: Alerting (§5)
- **Logs**: Logging (§5)

Proactive | Reactive | Retrospective
--- | --- | ---
• Why data-driven?
  - **Adaptivity**: Data-driven approach can adapt to various types of online services with different programming languages.
  - **Practicality**: Non-intrusive, like a plug-in module for online services.

• Why to use such types of monitoring data?
  - Such data types are universal in microservice architectures.
Evaluation Metrics

\[ CE = \frac{1}{N} \sum_{i=1}^{N} -[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)] \]

\[ MAE = \frac{\sum_{i=1}^{N} |y_i - p_i|}{n} \]

\[ RMSE = \sqrt{\frac{\sum_{i=1}^{N} (y_i - p_i)^2}{N}} \]

The smaller, the better.