Seer: Leveraging Big Data to Navigate The Increasing Complexity of Cloud Debugging

Yu Gan, Meghna Pancholi, Dailun Cheng, Siyuan Hu, Yuan He and Christina Delimitrou
Cornell University
Executive Summary

- **Microservices puts more pressure on performance predictability**
  - Microservices dependencies → propagate & amplify QoS violations
  - Finding the culprit of a QoS violation is difficult
  - Post-QoS violation, returning to nominal operation is hard

- **Anticipating QoS violations & identifying culprits**

- **Seer: Data-driven Performance Debugging for Microservices**
  - Combines lightweight RPC-level distributed tracing with hardware monitoring
  - Leverages scalable deep learning to signal QoS violations with enough slack to apply corrective action
From Monoliths to Microservices
Advantages of microservices:
- Ease & speed of code development & deployment
- Security, error isolation
- PL/framework heterogeneity

Challenges of microservices:
- Change server design assumptions
- Complicate resource management → dependencies
- Amplify tail-at-scale effects
- More sensitive to performance unpredictability
- No representative end-to-end apps with microservices
An End-to-End Suite for Cloud & IoT Microservices

- 4 end-to-end applications using popular open-source microservices → ~30-40 microservices per app
  - Social Network
  - Movie Reviewing/Renting/Streaming
  - E-commerce
  - Drone control service

- Programming languages and frameworks:
  - node.js, Python, C/C++, Java/Javascript, Scala, PHP, and Go
  - Nginx, memcached, MongoDB, CockroachDB, Mahout, Xapian
  - Apache Thrift RPC, RESTful APIs
  - Docker containers
  - Lightweight RPC-level distributed tracing
Resource Management Implications

- Challenges of microservices:
  - Dependencies complicate resource management
  - Dependencies change over time → difficult for users to express
  - Amplify tail@scale effects

Netflix
Twitter
Amazon
Movie Streaming
The Need for Proactive Performance Debugging

- Detecting QoS violations after they occur:
  - Unpredictable performance propagates through system
  - Long time until return to nominal operation
  - Does not scale
### Performance Implications

<table>
<thead>
<tr>
<th>Queue</th>
<th>CPU</th>
<th>Mem</th>
<th>Net</th>
<th>Disk</th>
</tr>
</thead>
</table>

[Image: A complex diagram with various icons and patterns representing different performance metrics.]

8
Seer: Data-Driven Performance Debugging

- Leverage the massive amount of traces collected over time

1. Apply online, practical data mining techniques that identify the culprit of an *upcoming* QoS violation

2. Use per-server hardware monitoring to determine the cause of the QoS violation

3. Take corrective action to prevent the QoS violation from occurring

- Need to predict 100s of msec – a few sec in the future
RPC level tracing
- Based on Apache Thrift
- Timestamp start-end for each microservice
- Store in centralized DB (Cassandra)
- Record all requests → No sampling
- Overhead: <0.1% in throughput and <0.2% in tail latency
Deep Learning to the Rescue

- Why?
  - Architecture-agnostic
  - Adjusts to changes in dependencies over time
  - High accuracy, good scalability
  - Inference within the required window

[Deep Learning Diagram]
DNN Configuration

Input signal

- Container utilization
- Latency
- Queue depth

Output signal

Which microservice will cause a QoS violation in the near future?
DNN Configuration

**Input signal**
- Container utilization
- Latency
- Queue depth

**Output signal**
Which microservice will cause a QoS violation in the near future?
DNN Configuration

- **Training** once: slow (hours - days)
  - Across load levels, load distributions, request types
  - Distributed queue traces, annotated with QoS violations
  - Weight/bias inference with SGD
  - Retraining in the background

- **Inference** continuously: streaming trace data

  93% accuracy in signaling upcoming QoS violations
  91% accuracy in attributing QoS violation to correct microservice
DNN Configuration

- **Challenges:**
  - In large clusters inference too slow to prevent QoS violations
  - Offload on TPUs, 10-100x improvement; 10ms for 90\textsuperscript{th} \%ile inference
  - Fast enough for most corrective actions to take effect (net bw partitioning, RAPL, cache partitioning, scale-up/out, etc.)
Experimental Setup

- 40 dedicated servers
- ~1000 single-concerned containers
- Machine utilization 80-85%
- Inject interference to cause QoS violation
  - Using microbenchmarks (CPU, cache, memory, network, disk I/O)
Restoring QoS

- Identify cause of QoS violation
  - Private cluster: performance counters & utilization monitors
  - Public cluster: contentious microbenchmarks

- Adjust resource allocation
  - RAPL (fine-grain DVFS) & scale-up for CPU contention
  - Cache partitioning (CAT) for cache contention
  - Memory capacity partitioning for memory contention
  - Network bandwidth partitioning (HTB) for net contention
  - Storage bandwidth partitioning for I/O contention
Restoring QoS

- Post-detection, baseline system → dropped requests
- Post-detection, Seer → maintain nominal performance
Demo
Challenges Ahead

- Security implications of data-driven approaches
- Fall-back mechanisms when ML goes wrong
- Not a single-layer solution → Predictability needs vertical approaches

Data-driven approaches offer practical solutions to problems whose scale makes previous approaches intractable.

Thank you!