SAGE: PRACTICAL AND SCALABLE ML-DRIVEN PERFORMANCE DEBUGGING IN MICROSERVICES

Yu Gan¹, Mingyu Liang¹, Sundar Dev², David Lo², Christina Delimitrou¹

¹Cornell University, ²Google

ASPLOS 2021 - Session 4: Microservices
@ 4:30 – 4:45 PM PDT, April 19th, 2021
EXECUTIVE SUMMARY

Motivation
- Microservices become increasingly popular in cloud systems
- Service-level objectives (SLOs) govern interactive microservices

Challenges in microservice performance debugging
- ML outperforms traditional heuristics

Sage: Root cause analysis system using unsupervised learning
- Use Causal Bayesian Networks for causal relationships among microservices
- Use counterfactuals to detect root causes (services and resources) of SLO violations
**BACKGROUND: MICROSERVICES**

- **Microservices**
  - Fine-grained, loosely-coupled, and single-concerned
  - Communicate with RPCs or RESTful APIs
  - SLOs: tail latency, availability, …

- **Pros**
  - Agile development
  - Better modularity & elasticity
  - Testing and debugging in isolation

- **Cons**
  - Different hardware & software constraints
  - Dependencies → complicate cluster management

BACKGROUND: MICROSERVICES
CHALLENGES OF MICROSERVICE PERF DEBUGGING

- Microservices are more sensitive to performance unpredictability\(^1\)
- Complex network dependencies\(^1\)
  - Hotspots can propagate
  - Difficulty in locating the root cause
- Complex tracing and monitoring
  - Requires end-to-end tracing and aggregation
  - Millions of timeseries over a long period of time
  - Complicates performance debugging, but makes data-driven methods possible

\(^1\) Yu Gan et al. "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems", ASPLOS 2019
Previous Studies

- **Previous work**
  - CauseInfer\[^1\] [INFOCOM’14]
  - Microscope\[^2\] [ICSOC’18]
  - Seer\[^3\] [ASPLOS’19]: Proactive root cause detection system

- **Limitations:**
  - PC-algorithm: Poor scalability, prone to statistical errors
  - Seer: Requires data labeling, high-precision time series & kernel-level tracing

---


DESIgn PRINcIpLES OF SAGE

- No need to label data
  - Challenge: correlation does not imply causation
  - Requires a causal model

- Robust to sampling frequency
  - Suitable for instrumentation in production
  - Not using temporal patterns for inference

- No need for kernel-level tracing

- Practical adjustment to service updates

- Focuses on resource provisioning-related performance issues
OVERVIEW OF TECHNIQUES

**Approach:**
- Causal Bayesian network (CBN) modeling
- Causal inference with generated counterfactuals

Client

Frontend

Logic tier

Backend

RPC dependency graph

Causal structure

Input latency & metrics

Counterfactual latency

Generate

Root cause services & resources
Causal Bayesian Network (CBN)
• A probabilistic graphical model where edges indicate causal relationships

Reason for using CBN modeling
• A tool for structural causal inference
• Interpretable and explainable
**Nodes in the CBN**

- **Service, node and network metrics** (*X* nodes)
  - Service and node metrics: CPU, memory, disk
  - Network metrics

- **RPC and network latency** (*Y* nodes)
  - Client- & server-side latency, request and response network delay

- **Latent variables** (*Z* nodes)
  - Unobservable or immeasurable
  - Assumed multivariate Gaussian distribution

---

CBN of two services

---

**Diagram:**
- Nodes and edges represent different metrics and relationships in the CBN model.
Causal Inference with Counterfactuals

- **Counterfactual queries**
  - Queries of hypothetical end-to-end latency if some metrics had been “normal”
  - Root causes: metrics that hypothetically solve the end-to-end performance issue

- **Generating counterfactuals with generative models**
**CONDITIONAL VARIATIONAL AUTOENCODER (CVAE)**

- **Prior network**: Learn prior distribution $p_{\psi}(Z \mid X)$
- **Encoder**: Learn posterior distribution $q_{\theta}(Z \mid X, Y)$
- **Decoder**: Reconstruct input SLI data by $p_{\phi}(Y \mid X, Z)$ with $Z$ sampled from posterior distribution
- **Loss function**: $L_{\text{CVAE}} = -\mathbb{E}_{Z \sim q_{\theta}(Z \mid X, Y)} \left[ \log p_{\phi}(Y \mid X, Z) \right] + \beta \cdot D_{\text{KL}}[q_{\theta}(Z \mid X, Y) \| p_{\psi}(Z \mid X)]$
**GRAPHICAL VARIATIONAL AUTOENCODER (GVAE)**

- **GVAE - factorizing CVAE according to the CBN model**
  - Factorization of the loss function: \( L_{GVAE} = \sum L_{CVAE} \)
  - One encoder and prior network for each service & network channel
  - One decoder for each RPC
  - Decoder connections are determined by the **information flow** in the CBN

- **Benefits of using GVAE**
  - Connection pruning to enforce the network to follow the causal model
  - Better interpretability
  - Faster retraining upon microservice updates
ROOT CAUSE DETECTION WITH GVAE

- Learn the latent variables ($Z$) from the encoder
- Calculate “normal” values of metrics and latent variables
  - Median value among normal traces
- Two-level intervention for root cause detection
  - Locate culprit services
  - Locate culprit resource
Incremental & Partial Retraining

- Microservices updated frequently
  - Services added, removed & updated

- Incremental & partial retraining
  - Only retrain upstreaming services affected by the updates
**INCREMENTAL & PARTIAL RETRAINING**

- **Microservices updated frequently**
  - Services added, removed & updated

- **Incremental & partial retraining**
  - Only retrain upstreaming services affected by the updates

![Diagram showing retraining process with nodes A, B, and C, and VAE A, B, and C.]
**INCREMENTAL & PARTIAL RETRAINING**

- **Microservices updated frequently**
  - Services added, removed & updated

- **Incremental & partial retraining**
  - Only retrain upstreaming services affected by the updates

---

**Diagram:**

- Node A
  - Connects to VAE A
  - Connects to VAE B
  - Connects to VAE C

- Node B
  - Connects to VAE A

- Node C
  - Connects to VAE B
  - Connects to VAE C

- VAE C is updated twice, indicated by two separate update arrows.
**SYSTEM DESIGN**

- **Monitoring**
  - Jaeger and Prometheus for collecting traces & performance metrics

- **Data collection**
  - Preprocessing, normalization

- **GVAE model**
  - Implemented with PyTorch

- **Actuation**
  - Scale up/out, CAT, network BW partitioning
EVALUATION

- **Methodology**
  
  - **Applications**
    - Synthetic Thrift chain and fanout services
    - DeathstarBench\[1\]
  
  - **Systems**
    - Local cluster: 2-socket 40-core servers with 128GB RAM and 2-socket 88-core servers with 188GB RAM each
    - Google Compute Engine: 84 nodes with 4-64 cores, 4-64GB RAM and 20-128GB SSD
  
  - **Baselines and prior work**
    - Autoscaling and Offline Oracle
    - CauseInfer\[2\] and Microscope\[3\]
    - Seer\[4\]

### Accuracy of detecting root cause

- Sage has 88%-95% accuracy across five applications.
- CauseInfer and Microscope have low accuracy due to errors in finding causal relationships with PC-algorithm.
- Seer has similar accuracy, but Sage needs less information.
EVALUATION

- Actuation
  - Sage resolves SLO violations fast
  - Because of false negatives, other methods cannot always resolve the issue
**Incremental & partial retraining**

- Less accuracy drop & faster convergence
- Incremental retraining: reusing neural network parameters
- Partial retraining: updating subset of neurons

![Graph showing detection accuracy over time](image)

- A: One service added at frontend
- B: One service updated
- C: One service removed
- D: One service added at backend
- E: Multiple services added, updated, and removed
- F: More services added, updated, and removed
EVALUATION

- **Scalability on GCE**

  - 84 nodes with 4-64 cores, 4-64GB RAM and 20-128GB SSD
  - 6.7x more containers
  - Comparable accuracy with local runs
  - 19.4% increase in training time and 26.5% increase in inference time
    » Collecting distributional data across replicas
CONCLUSIONS

- Performance debugging for microservice is challenging
- Sage: Root cause detection system based on unsupervised learning
  - Causal Bayesian network for modeling causal relationships
  - Counterfactual queries for root cause detection
- Evaluation with representative microservices
  - Accurate detection and fast actuation
  - Fast convergence upon service updates
  - Scales well to large clusters on GCE
- Future work
  - More types of issues: design bugs, security issues, …
Thank you!

Questions are welcome at Session 4 Q&A Panel
@ 4:45 – 5:00 PM PDT, April 19th, 2021