1. Motivation

Cloud applications are progressively shifting from monolithic services to graphs with hundreds of single-purpose and loosely-coupled microservices [2, 7, 8, 22, 48, 49]. Several large cloud providers, such as Amazon, Twitter, Netflix, and eBay have already adopted this application model [2, 7, 8].

Microservices offer several advantages, such as rapid integration and deployment, but they also introduce new challenges with respect to resource management, as dependencies between tiers introduce backpressure effects, causing unpredictable performance to propagate through the system [22, 23]. Diagnosing such performance issues empirically is both cumbersome and prone to errors, especially as typical microservices deployments include hundreds or thousands of unique tiers. Similarly, current cloud managers [17, 18, 19, 27, 29, 30, 31, 32, 35, 36, 38, 42, 45, 53, 56] are not expressive enough to account for the impact of microservice dependencies, thus putting more pressure on the need for automated performance debugging systems.

2. Limitations of the State of the Art

With the increased pervasiveness of the cloud, monitoring and performance debugging systems that track and investigate system and application behavior over time have gained attention. Several tools, such as [9, 14, 20], construct causal paths and diagnose performance issues in distributed systems. There are also several production-level distributed tracing systems, including Dapper [46], Zipkin [6], Jaeger [4], and Google-Wide Profiling (GWP) [39]. Dapper, Zipkin and Jaeger record RPC-level traces for sampled requests across the calling stack, while GWP monitors low-level hardware metrics. These systems aim to facilitate locating performance issues, but are not geared towards taking action to resolve them.

On the performance debugging front, there has been increased attention on trace-based methods to analyze [12, 20, 37], diagnose [9, 10, 13, 16, 24, 25, 26, 34, 40, 51, 54, 55], and in some cases anticipate [21, 23, 50] performance issues in cloud services. Autopilot [41], for example, adjusts the number of tasks and CPU/memory limits automatically to reduce resource slack while guaranteeing performance.

While most of these systems target cloud applications, they almost always focus on single-tier services, and even the ones targeting microservices [23] rely on supervised learning and invasive instrumentation to correctly diagnose the root causes of unpredictable performance. This is problematic in real cloud deployments, as labeling training data with the root causes of QoS violations requires manually diagnosing past performance issues, or injecting new ones whose cause is known. This is impractical, as introducing performance issues hurts the availability and user experience of live applications. Similarly, instrumenting the application and kernel is non-trivial, especially in cases of third-party microservices, whose source code may not be available. Therefore, it is important to explore performance debugging techniques that can uncover the impact of dependencies between microservices without the need for data labeling or expensive instrumentation.

3. Key Insights

We present Sage, a root cause analysis system that leverages unsupervised learning to identify the culprit of unpredictable performance in complex graphs of microservices. Specifically, Sage uses Causal Bayesian Networks to capture the dependencies between microservices, and counterfactuals to examine the impact of microservices on end-to-end performance. Sage does not rely on data labeling, hence it can be entirely transparent to both cloud users and application developers, scales well with the number of microservices and machines, and only relies on lightweight tracing that does not require application changes or kernel instrumentation.

The main design principles in Sage are the following:

- **Unsupervised learning**: Sage focuses on unsupervised learning to circumvent the overhead of labeling training data, which is not practical and/or scalable in real deployments. Instead, it shows that low-frequency live traces collected using infrastructure readily available in cloud providers today, coupled with a set of analytical and ML methods are sufficient to correctly identify the culprits of QoS violations in a complex system.

- **Robustness to sampling frequency**: Sage does not require tracking individual requests to detect temporal latency patterns, making it robust to tracing frequency. This is important, as production tracing systems like Dapper [46] employ aggressive request sampling to reduce overheads [15, 43]. In comparison, previous studies [23, 44, 52] collect traces at the granularity of 10s-100s of milliseconds, which can introduce significant monitoring overheads.

- **User-level metrics**: Sage only uses user-level metrics that can be easily obtained through cloud monitoring APIs and service-level traces from distributed tracing frameworks, such as Jaeger [4] or Zipkin [6]. It does not require any kernel-level information, which is expensive, or even inaccessible in many cloud platforms.

- **Partial retraining**: A major design premise of microservices is enabling frequent updates. Retraining the entire debugging system every time the code or deployment of a mi-
microservice changes is prohibitively expensive. Instead Sage implements partial and incremental retraining, whereby only the microservice that changed and its immediate neighbors are retrained, greatly reducing overheads.

- **Fast resolution:** Empirically examining different sources of unpredictable performance is costly in both time and resources, especially due to the ingest delay cloud systems have in consuming monitoring data, causing a change to experience inertia before propagating on recorded traces. Sage models the impact of the different probable root causes concurrently, enabling faster QoS recovery.

4. Main Artifacts

Sage is an ML-driven performance debugging system for interactive cloud microservices. An overview of the ML pipeline and system architecture of Sage can be found in Figures 1 and 7 of the original paper. The main artifacts we present are:

- **ML pipeline:** Sage contributes an unsupervised ML pipeline consisting of a causal Bayesian network (CBN) and a graphical variational auto-encoder (GVAE). The CBN is trained on RPC-level distributed traces [6, 46] to capture the dependencies between microservices, as well as causal relationships between individual microservices and the end-to-end performance. The CBN also captures the latency propagation from the backend to the frontend. Second, Sage uses a graphical variational auto-encoder (GVAE), a deep generative model, to generate hypothetical scenarios (counterfactuals [33]), which tweak the performance and/or usage of individual microservices to values known to meet QoS, and infers whether the change restores QoS. Using these two techniques, Sage determines which set of microservices initiated a QoS violation, and adjusts their deployment or resource allocation accordingly.

- **System design and implementation:** We have designed and implemented the end-to-end debugging system, including the tracing infrastructure, ML pipeline, and performance debugging system. The system uses Jaeger [4], a distributed RPC tracing system for end-to-end execution traces, and the Prometheus Node Exporter [5] to collect hardware/OS metrics, container-level performance metrics, and network latencies. Sage uses a centralized master for trace processing, root cause analysis, and actuation, implemented in approximately 6KLOC of Python, and per-node agents for trace collection and container deployment. It also maintains two hot stand-by copies of the master for fault tolerance. The GVAE model is built in PyTorch, with each VAE’s encoder, decoder, and prior networks using a DNN with 3-5 fully connected layers.

- **Validation methodology & large-scale evaluation:** We validate Sage’s root cause detection accuracy using both synthetic microservice topologies based on Apache Thrift [1, 47], a widely-used RPC framework, and an end-to-end application from the DeathStarBench suite implementing a Social Network [22]. We use wrk2 [3], an open-loop HTTP workload generator, to send requests to the front-ends of all applications. We first validate Sage’s accuracy in a controlled local cluster, and then we demonstrate Sage’s scalability on a large-scale deployment on Google Compute Engine.

5. Key Results

We compare Sage with autoscaling techniques, which are widely used in industry, as well as recent work on performance debugging (CauseInfer [11], Microscope [28], and Seer [23]), targeting both monolithic and microservice applications.

In the dedicated local cluster, we show that Sage achieves 91%-95% root cause detection accuracy, and can quickly take action and restore QoS. It significantly outperforms the autoscaling techniques, by learning the impact of microservice dependencies, instead of memorizing usage thresholds for a particular cluster state. We also show that Sage outperforms prior work on performance debugging, namely CauseInfer and Microscope, which often identify the wrong paths in the dependency graph when searching for root causes. Finally, we show that the unsupervised models in Sage achieve very similar accuracy to the supervised model of Seer, but are significantly more practical and scalable. Specifically, unlike Seer, Sage does not require millisecond-level tracing of queue lengths across the system stack, and it does not need labeling data for training. This makes Sage more portable in datacenter deployments, especially when the application includes libraries or tiers that cannot be instrumented.

We also evaluate Sage’s ability to adjust to changes in application design, which are common in microservices. To adapt to such changes, Sage uses transfer learning and partial retraining to localize the model updates to only neurons affected by a change. We show that transfer learning allows Sage to keep its detection accuracy high, and makes retraining 3-30× faster than when retraining the model from scratch.

Finally we evaluate Sage’s scalability on 188 container instances on Google Compute Engine. Although we deploy 6.7× more containers compared to the local cluster, the training and inference times only increase by 19.4% and 26.5% respectively. The detection accuracy is not impacted by system scale. Finally, we show that in addition to resource-related performance issues, Sage is also able to detect performance problems caused by software bugs, and isolate the microservice where the bug resides.

6. Why ASPLOS

Sage tackles performance predictability in microservices, a challenging problem that many cloud providers face, especially on an increasingly prevalent application model. Performance debugging and cloud systems are popular topics in ASPLOS and involve hardware, operating systems, and application-level innovations. Sage leverages ML for performance debugging, which also fits in the ML for systems topic.

7. Citation for Most Influential Paper Award

For introducing a data-driven approach in cloud performance debugging, and showing the potential and benefits of using unsupervised learning to address the performance challenges of interactive microservices in a scalable and practical manner.
References


