# Leveraging Deep Learning to Improve Performance Predictability in Cloud Microservices with Seer

Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou
Cornell University
{yg397, yz2297, sh2442, dc924, yh772, mp832, delimitrou}@cornell.edu

## Abstract

*Performance unpredictability is a major roadblock towards cloud adoption, and has performance, cost, and revenue ramifications. Predictable performance is even more critical as cloud services transition from monolithic designs to microservices. Detecting QoS violations after they occur in systems with microservices results in long recovery times, as hotspots propagate and amplify across dependent services.*

*In this work we discuss how machine learning can be used to proactively detect upcoming QoS violations be framing performance debugging as a pattern matching problem. We present Seer, an online cloud performance debugging system that leverages the massive amount of tracing data cloud systems collect to learn spatial and temporal patterns that translate to QoS violations. Seer combines lightweight distributed RPC-level tracing, with detailed low-level hardware monitoring to signal an upcoming QoS violation, and diagnose the source of unpredictable performance. We validate Seer's accuracy in signaling upcoming QoS violations both in controlled local and large-scale public clusters. Adopting data-driven approaches like Seer offers insights on how to better architect microservices to achieve predictable performance, providing practical solutions for systems whose scale make previous empirical solutions impractical.*

## 1. Introduction

The last few years have seen a significant shift in the way cloud services are designed, from monolithic applications that encompass the entire functionality in a single codebase, to large graphs of single-concerned, loosely-coupled microservices [5, 21]. Microservices are appealing for several reasons, including accelerating development and deployment, isolating errors can be isolated in specific tiers, and enabling a rich software ecosystem, as each microservice is written in the language or programming framework that best suits its needs.

At the same time microservices introduce several new system challenges. Specifically, even though the quality-of-service (QoS) requirements of the end-to-end application are similar for microservices and monoliths, the tail latency required for each individual microservice is much stricter than for traditional cloud applications [28, 26, 15, 29, 27, 19, 20, 21]. This puts increased pressure on delivering predictable performance, as dependencies between microservices mean that a single misbehaving microservice can cause cascading QoS violations across the system.

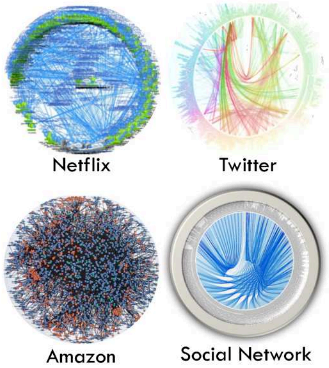

**Figure 1: Microservices graphs in three large cloud providers [5, 1], and our `Social Network` service.**

Fig. 1 shows three real large-scale production deployments of microservices [5, 1]. The perimeter of the circle (or sphere surface) shows the different microservices, and edges show dependencies between them. We also show these dependencies for *Social Network*, one of the large-scale services used in the evaluation of this work (see Sec. 3). Unfortunately the complexity of modern cloud services means that manually determining the impact of each pair-wise dependency on end-to-end QoS, or relying on the user to provide this information is impractical.

Apart from software heterogeneity, datacenter hardware is also becoming increasingly heterogeneous as special-purpose architectures [24, 9] and FPGAs are used to accelerate critical operations [8, 11]. This adds to the existing server heterogeneity from incremental server replacement [27, 14], and further complicates the effort to guarantee predictable performance.

Even though there has been extensive work on tracing, monitoring and performance debugging systems which help detect and correct unpredictable performance, until their actions take effect, performance suffers. For monolithic services this primarily affects the service experiencing the QoS violation itself, and potentially services it is sharing physical resources with. With microservices, however, a posteriori QoS violation detection is more impactful, as hotspots propagate and amplify across dependent services, forcing the system to operate in a degraded state for longer, until all oversubscribed tiers have been relieved, and all accumulated queues have drained. Fig. 2 shows the impact of reacting to a QoS violation after it occurs for the *Social Network* application with several hundred users running on 20 two-socket, high-end servers. Even though the scheduler scales out all oversubscribed tiers once the violation occurs, it takes several seconds for the service to return to nominal operation. There are two reasons for this; first, by the time one tier has been upsized, its neighboring tiers have built up request backlogs, which cause them to saturate in turn. Second, utilization is not always a good proxy for tail latency and/or QoS violations [26, 6, 12]. A common way to address such QoS violations is rate limiting [33], which constrains the incoming load, until hotspots dissipate. This restores performance, but degrades the end user's experience, as a fraction of input requests is dropped.

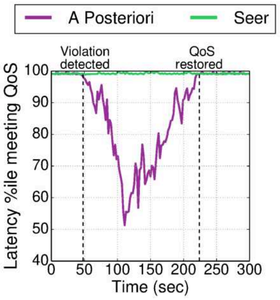

**Figure 2: Performance with a posteriori debugging and with Seer.**

In our recent ASPLOS'19 paper [22] we presented *Seer*, a proactive cloud performance debugging system that leverages practical deep learning techniques to diagnose upcoming QoS violations in a scalable and online manner. First, Seer is proactive to avoid the long recovery periods of a posteriori QoS violation detection. Second, it uses the massive amount of tracing data cloud systems collect over time to learn spatial and temporal patterns that lead to QoS violations early enough to avoid them altogether. Seer includes a lightweight, distributed RPC-level tracing system, based on Apache Thrift's timing interface, to collect end-to-end traces of request execution, and track per-microservice outstanding requests. Seer uses these traces to train a deep neural network to recognize imminent QoS violations, and identify the microservice(s) that initiated them. Once Seer identifies the culprit of a QoS violation, it uses detailed per-node hardware monitoring to determine the reason behind the degraded performance, and provide the cluster scheduler with recommendations on how to avoid it.

We evaluated Seer both in dedicated local clusters of 20 two-socket servers, and on large-scale clusters on Google Compute Engine (GCE) with a set of end-to-end interactive applications built with microservices, including the *Social Network* above. In our local cluster, Seer correctly identifies upcoming QoS violations in 93% of cases, and correctly pinpoints the microservice initiating the violation 89% of the time. To combat long inference times as clusters scale, we offload the DNN pipeline to Google's Tensor Processing Units (TPUs) when running on GCE [24], and show up to orders of magnitude of improvement for both training and inference.

Finally, we deploy Seer in a large-scale installation of the *Social Network* service with several hundred users, and show that Seer not only correctly identifies 90.6% of upcoming QoS violations and avoids 84% of them, but also helps the application's developers identify design bugs, resulting in fewer QoS violations over time. As cloud application and hardware complexity continues to grow, data-driven systems like Seer can offer practical solutions for systems whose scale make empirical approaches intractable.

## 2. Related Work

Performance unpredictability is a well-studied problem in public clouds that stems from platform heterogeneity, resource interference, software bugs and load variation [15, 16, 17, 26].

There is extensive work on monitoring systems that has shown that execution traces can help diagnose performance, efficiency, and even security problems in large-scale systems [18, 10, 32, 29]. For example, X-Trace is a tracing framework that provides a comprehensive view of the behavior of services running on large-scale, potentially shared clusters. X-Trace supports several protocols and software systems, and has been deployed in several real-world scenarios, including DNS resolution, and a photo-hosting site [18]. The Mystery Machine, on the other hand, leverages the massive amount of monitoring data cloud systems collect to determine the causal relationship between different requests [10]. Cloudseer serves a similar purpose, building an automaton for the workflow of each task based on normal execution, and then compares against this automaton at runtime to determine if the workflow has diverged from its expected behavior [34]. Finally, there are several systems, including Dapper [32], GWP [30], and Zipkin [4] which provide the tracing infrastructure for large-scale services at Google and Twitter, respectively. Dapper and Zipkin trace distributed user requests at RPC granularity, while GWP focuses on low-level hardware monitoring.

Root cause analysis of performance abnormalities in the cloud has also gained increased attention over the past few years, as the number of interactive, latency-critical services hosted in cloud systems has increased. Jayathilaka et al. [23], for example, developed Roots, a system that automatically identifies the root cause of performance anomalies in web applications deployed in Platform-as-a-Service (PaaS) clouds. Roots tracks events within the PaaS cloud using a combination of metadata injection and platform-level instrumentation.

Even though this work does not specifically target interactive, latency-critical microservices, or applications of similar granularity, such examples provide promising evidence that data-driven performance diagnostics can improve a large-scale system's ability to identify performance anomalies, and address them to meet its performance guarantees.

## 3. End-to-End Applications with Microservices

We motivate and evaluate Seer with a set of new end-to-end, interactive services built with microservices. Even though there are open-source microservices that can serve as components of a larger application, such as `nginx`, `memcached`, `MongoDB`, `Xapian`, and `RabbitMQ`, there are currently no publicly-available end-to-end microservices applications, with the exception of a few simple architectures, like Go-microservices [2], and Sockshop [3]. We design four end-to-end services implementing a *Social Network*, a *Media Service*, an *E-commerce Site*, and a *Banking System*. Starting from the Go-microservices architecture [2], we also develop

| Service | Communication | Unique | Per-language LoC breakdown |
|---|---|---|---|
| | Protocol | Microservices | (end-to-end service) |
| **Social Network** | **RPC** | 36 | 34% C, 23% C++, 18% Java, 7% node, 6% Python, 5% Scala, 3% PHP, 2% JS, 2% Go |
| **Media Service** | **RPC** | 38 | 30% C, 21% C++, 20% Java, 10% PHP, 8% Scala, 5% node, 3% Python, 3% JS |
| **E-commerce Site** | **REST** | 41 | 21% Java, 16% C++, 15% C, 14% Go, 10% JS, 7% node, 5% Scala, 4% HTML, 3% Ruby |
| **Banking System** | **RPC** | 28 | 29% C, 25% Javascript, 16% Java, 16% node.js, 11% C++, 3% Python |
| **Hotel Reservations [2]** | **RPC** | 15 | 89% Go, 7% HTML, 4% Python |

**Table 1: Characteristics and code composition of each end-to-end microservices-based application.**

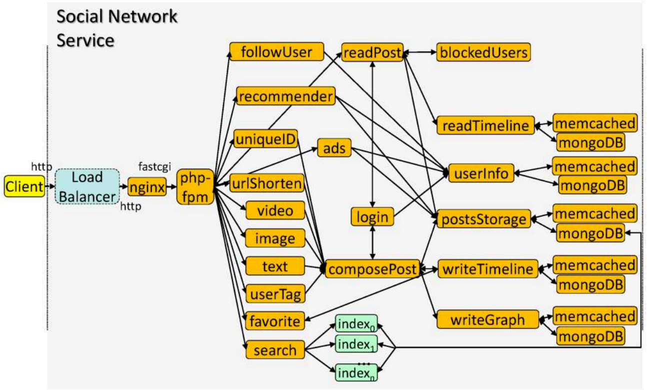

**Figure 3: Dependency graph between the microservices of the end-to-end *Social Network* application.**

an end-to-end *Hotel Reservation* system. Services are designed to be representative of frameworks used in production systems, modular, and easily reconfigurable. The applications and tracing infrastructure are described in more detail in [21].

Table 1 briefly shows the characteristics of each end-to-end application, including its communication protocol, the number of unique microservices it includes, and its breakdown by programming language and framework. Unless otherwise noted, all microservices are deployed in Docker containers. Below, we briefly describe the scope and functionality of the Social Network applications; the remaining services also have similar structure and granularity.

### 3.1. Social Network

**Scope:** The end-to-end service implements a broadcast-style social network with uni-directional follow relationships.

**Functionality:** Fig. 3 shows the architecture of the end-to-end service. Users (`client`) send requests over `http`, which first reach a load balancer, implemented with `nginx`, which selects a specific webserver is selected, also in `nginx`. Users can create posts embedded with text, media, links, and tags to other users, which are then broadcasted to all their followers. Users can also read, favorite, and share posts, as well as reply publicly, or send a direct message to another user. The application also includes machine learning plugins, such as ads and user recommender engines [7], a search service using `Xapian`, and microservices that allow users to follow, unfollow, or block other accounts. Inter-microservice messages use Apache Thrift RPCs. The service's backend uses `memcached` for caching, and `MongoDB` for persistently storing posts, user profiles, media, and user recommendations. This service is broadly deployed at Cornell and elsewhere, and currently has several hundred users. We use this installation to test the effectiveness and scalability of Seer in Section 6.

## 4. Seer Design

### 4.1. Overview

Fig. 4 shows the high-level architecture of the system. Seer is an online performance debugging system for cloud systems hosting interactive, latency-critical services. Even though we are focusing our analysis on microservices, where the impact of QoS violations is more severe, Seer is also applicable to general cloud services, and traditional multi-tier or Service-Oriented Architecture (SOA) workloads.

First, Seer uses a lightweight, distributed RPC-level tracing system, which collects end-to-end execution traces for each user request, including per-tier latency and outstanding requests, associates RPCs belonging to the same end-to-end request, and aggregates them to a centralized Cassandra database (TraceDB). Traces are used to train Seer to recognize patterns in space (between microservices) and time that lead to QoS violations. At runtime, Seer consumes real-time streaming traces to infer whether there is an imminent QoS violation.

Once problematic microservices have been isolated, Seer uses its lower tracing level, to identify the reason behind the QoS violation. It also uses this information to provide the cluster manager with recommendations on how to avoid the degraded performance altogether. Using two specialized tracing levels instead of collecting detailed low-level traces for all active microservices ensures that the distributed tracing is lightweight enough to track all active services in the system, and that detailed low-level hardware tracing is only used on-demand, for microservices likely to cause disruptions.

### 4.2. Distributed Tracing

A major challenge with microservices is that one cannot simply rely on the client to report performance, as with traditional client-server applications. We have developed a distributed tracing system for Seer, similar in design to Dapper [32] and Zipkin [4] that records per-microservice latencies, using the Thrift timing interface. We additionally track the number of requests queued in each microservice (*outstanding requests*), since queue lengths are highly correlated with performance and QoS violations [25]. In all cases, the overhead from tracing without request sampling is negligible, less than 0.1% on

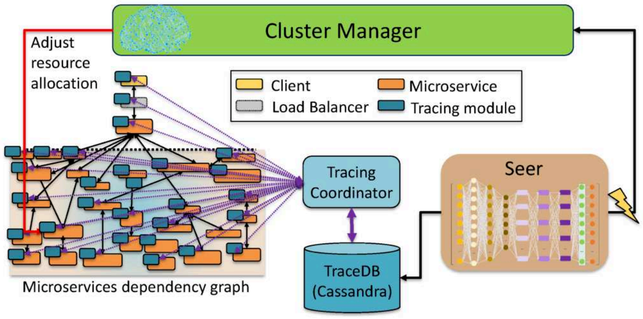

**Figure 4: Overview of *Seer*'s operation.**

end-to-end latency, and less than 0.15% on throughput (QPS), which is tolerable for such systems [32, 10, 30].

### 4.3. Deep Learning in Performance Debugging

A popular way to model performance in cloud systems, especially when there are dependencies between tasks, are *queueing networks*. Although queueing networks are a valuable tool to model how bottlenecks propagate through the system, they require in-depth knowledge of application semantics and structure, and can become overly complex as applications and systems scale. They additionally cannot easily capture all sources of contention, such as the OS and network stack.

Instead in Seer, we take a data-driven approach that assumes no information about the structure and characteristics of a service, making it robust to unknown and changing applications. Seer relies on practical learning techniques that identify patterns that lead to QoS violations. This includes both *spatial* patterns, such as dependencies between microservices, and *temporal* patterns, such as input load, and resource contention. The key idea in Seer is that conditions that led to QoS violations in the past can be used to recognize unpredictable performance in the near future. Below we describe the structure of the neural network, and how Seer adapts to changes in application structure online.

**Configuring the DNN:** The choice of DNN architecture is also instrumental to its accuracy. The number of input and output neurons correspond to the number of active microservices ordered according to their dependencies from back- to front-end to capture spatial patterns between neighboring microservices. Signaling QoS violations in a large cluster requires both spatial recognition, namely identifying problematic clusters of microservices and discarding noisy but non-critical microservices, for which CNNs are well-suited, and temporal recognition, namely using past QoS violations to anticipate future ones, for which LSTM networks are well-fit. We compare three network designs, a CNN, an LSTM, and a hybrid network that combines the two, using the CNN first to reduce the dimensionality and filter out microservices that do not affect end-to-end performance, and then an LSTM with a *SoftMax* final layer to infer the probability for each microservice to initiate a QoS violation. The architecture of the hybrid network is shown in Fig. 5. Each network is configured using hyperparameter tuning to avoid overfitting. We train each network on a week's worth of trace data collected on a 20-server cluster running all end-to-end services and test it on traces collected on a different week, after the servers had been patched, and the OS had been upgraded.

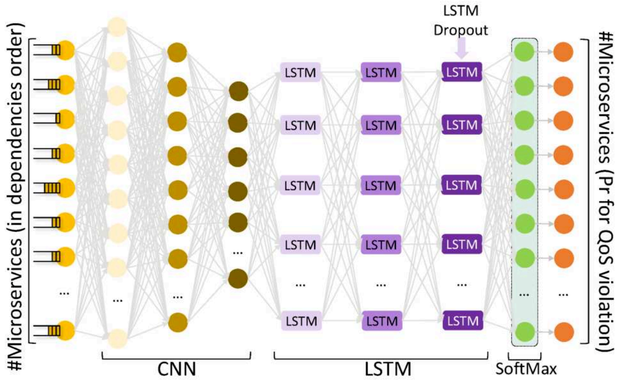
#Microservices (in dependencies order)
LSTM Dropout
LSTM
CNN
LSTM
SoftMax
#Microservices (Pr for QoS violation)

**Figure 5: The DNN in Seer, consisting of a set of convolution layers followed by a set of long short-term memory layers. Each input and output neuron corresponds to a microservice, ordered in topological order, from back-end to front-end.**

The quantitative comparison of the three networks is shown in Fig. 6a. The CNN is by far the fastest, but also the worst performing, since it is not designed to recognize patterns in time. The LSTM, on the other hand, is especially effective at capturing load patterns over time, but is less effective at reducing the dimensionality of the original dataset, which makes it prone to false positives due to non-critical microservices with many outstanding requests. Finally, Seer correctly anticipates 93.45% of violations, outperforming both networks, for a small increase in inference time compared to the LSTM. Given that most resource partitioning decisions take effect after a few 100ms, the inference time for Seer is within the window of opportunity the cluster manager has to take action. More importantly it attributes the QoS violation to the correct microservice, simplifying the cluster manager's task. QoS violations missed by Seer included four random load spikes, and a network switch failure which caused high packet drops.

**Incremental retraining:** Training happens once from scratch, and can be time consuming, taking several hours up to a day for week-long traces collected on clusters with hundreds of servers. However, one of the main advantages of microservices is that they simplify frequent application updates, with old microservices often swapped out and replaced by newer modules, or large services progressively broken down to microservices. If the application (or underlying hardware) changes significantly, Seer's detection accuracy can be impacted. To adjust to changes in the execution environment, Seer retrains incrementally in the background, using zero padding and the *transfer learning*-based approach in [31], which allows for weights from previous training rounds to be stored in disk, resuming training from where the model last left off.

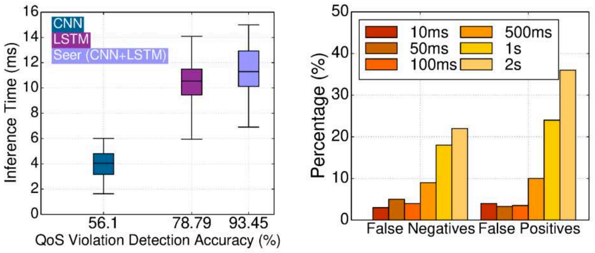

**Figure 6: (a) Comparison of DNN architectures, (b) False negatives and false positives as we vary the inference window.**

### 4.4. Hardware Monitoring

Once a QoS violation is signaled and a culprit microservice is pinpointed, Seer uses low-level monitoring to identify the reason behind the QoS violation. In private clusters, we use performance counters and utilization monitors to distinguish between bottlenecked and non-bottlenecked resources. In public clouds, where access to performance counters is limited, we use a set of contentious microbenchmarks to identify resources that impact the performance of an active microservice [13]. Upon identifying the problematic resource(s), Seer notifies the cluster manager, which uses resource partitioning and isolation to adjust the allocation of the problematic resource.

### 4.5. Implementation

Seer is implemented in 12KLOC of C,C++, and Python. It runs on Linux and OSX and supports applications in various languages, including all frameworks the end-to-end services are designed in. Furthermore, we provide automated patches for the instrumentation probes for many popular microservices, including `NGINX`, `memcached`, `MongoDB`, `Xapian`, and all Sockshop and *Go-microservices* applications. Seer is a centralized system; we use master-slave mirroring to improve fault tolerance, with two hot fail-over masters. The trace database is also replicated in the background.

## 5. Seer Evaluation

### 5.1. Methodology

**Server clusters:** First, we use a dedicated local cluster with 20, 2-socket 40-core servers with 128GB of RAM each. Each server is connected to a 40Gbps ToR switch over 10Gbe NICs. Second, we deploy the Social Network service to Google Compute Engine (GCE) and Windows Azure clusters with hundreds of servers to study the scalability of Seer.
**Applications:** We use all five end-to-end services of Table 1. Services are driven by open-loop workload generators, with the exception of the study in Section 6, where we examine a large-scale deployment of the *Social Network*; in that case the input load is driven by real user traffic.

### 5.2. Validation

**False negatives & false positives:** Fig. 6b shows the percentage of false negatives and false positives as we vary the prediction window. When Seer tries to anticipate QoS violations that will occur in the next 10-100ms both false positives and false negatives are low, since Seer uses a very recent snapshot of the cluster state to anticipate performance unpredictability. However, given that applying corrective actions takes 10-100s of milliseconds to take effect, such short windows do not allow enough time to avoid the QoS violation. At the other end, predicting far into the future results in significant false negatives, and especially false positives. This is because many QoS violations are caused by very short, bursty events that do not have an impact on queue lengths until a few milliseconds before the violation occurs. Unless otherwise specified we use a 100ms prediction window.

## 6. Large-Scale Cloud Study

### 6.1. Seer Scalability

We now deploy our *Social Network* service on a 112-server dedicated cluster on Google Compute Engine (GCE), and use it to service real user traffic. The application has 582 registered users, with 165 daily active users, and has been deployed for a two-month period. The cluster on average hosts 386 single-concerned containers (one microservice per container), subject to resource scaling actions based on Seer's feedback.

Accuracy remains high, consistent with the small-scale experiments. Inference time, however, increases substantially from 11.4ms for the 20-server cluster to 54ms. Even though this is sufficient for many allocation decisions, as the application scales further, Seer's ability to find QoS violations early enough diminishes.

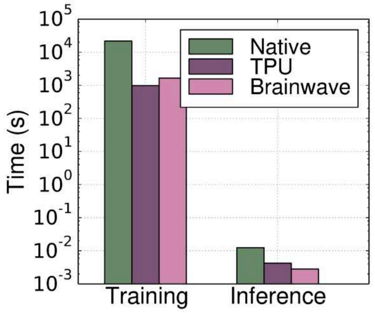

**Figure 7: Seer training and inference with hardware acceleration.**

Over the past year multiple public cloud providers have exposed hardware acceleration offerings for DNN training and inference, either using a special-purpose design like the Tensor Processing Unit (TPU) from Google [24], or using reconfigurable FPGAs, like Project Brainwave from Microsoft [11]. We offload Seer's DNN logic to both systems, and quantify the impact on training and inference time, and detection accuracy [1]. Fig. 7 shows this comparison for a 200GB training dataset. Both the TPU and Project Brainwave dramatically outperform our local implementation, by up to two orders of magnitude. Between the two accelerators, the TPU is more effective in training, consistent with its design objective [24], while Project Brainwave achieves faster inference. For the

[1] Before running on TPUs, we reimplemented our DNN in Tensorflow. We similarly adjust the DNN to the currently-supported designs in Brainwave.

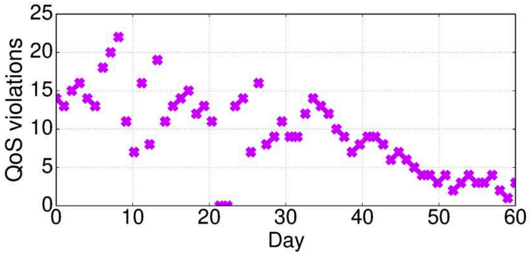

**Figure 8: Number of QoS violations in Social Network, when Seer is employed over a two month period.**

remainder of the paper, we run Seer on TPUs, and host the *Social Network* service on GCE.

### 6.2. Seer's Long-Term Impact on Application Design

Seer has now been deployed in the *Social Network* cluster for over two months, and during this time it has detected 536 upcoming QoS violations (90.6% accuracy) and avoided 495 (84%). Furthermore, by detecting recurring patterns that lead to QoS violations, Seer has helped the application developers better understand bugs and design decisions that lead to hotspots, such as microservices with a lot of back-and-forth communication, or microservices forming cyclic dependencies, or using blocking primitives. This has led to a decreasing number of QoS violations over the two months (seen in Fig. 8), as the application progressively improves. There are still some spikes in QoS violations, coinciding with major redesigns of the application, and in days 22, 23 there was a cluster outage, resulting in zero reported violations. Systems like Seer can be used not only to improve performance predictability, but also to better understand the design challenges of microservices, as more services transition to this application model.

## 7. Conclusions

Cloud services increasingly move away from complex monolithic designs, and adopt the model of specialized, loosely-coupled microservices. For such services traditional performance debugging is insufficient, as dependencies between microservices lead to backpressure effects and prolonged degraded performance. We presented Seer, a proactive, data-driven performance debugging system that leverages practical learning techniques, and the massive amount of tracing data cloud systems collect to anticipate and avoid QoS violations. We have validated Seer's accuracy in controlled environments, and evaluated its scalability on large-scale clusters on public clouds. In all scenarios, Seer accurately detects upcoming QoS violations, improving responsiveness and performance predictability. As more services transition to the microservices model, systems like Seer provide practical solutions to help navigate the increasing complexity of the cloud.

## References

[1] Decomposing twitter: Adventures in service-oriented architecture. http://tiny.cc/rg0k6y.

[2] Golang microservices example. https://github.com/harlow/go-micro-services.

[3] Sockshop: A microservices demo application. http://tiny.cc/5c0k6y.

[4] Zipkin. http://zipkin.io.

[5] The evolution of microservices. http://tiny.cc/ka0k6y, 2016.

[6] Luiz Barroso and Urs Hoelzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 2009.

[7] Robert Bell, Yehuda Koren, and Chris Volinsky. The bellkor 2008 solution to the netflix prize. Technical report, 2007.

[8] Adrian Caulfield, Eric Chung, and et al. A cloud-scale acceleration architecture. In *MICRO*, 2016.

[9] Tianshi Chen, Zidong Du, and et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proc. of ASPLOS*, 2014.

[10] Michael Chow, David Meisner, Jason Flinn, Daniel Peek, and Thomas F. Wenisch. The mystery machine: End-to-end performance analysis of large-scale internet services. In *Proceedings of OSDI*, 2014.

[11] Eric S. Chung, Jeremy Fowers, and et al. Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro*, 38(2), 2018.

[12] Jeffrey Dean and Luiz Andre Barroso. The tail at scale. In *CACM, Vol. 56 No. 2*.

[13] Christina Delimitrou and Christos Kozyrakis. iBench: Quantifying Interference for Datacenter Workloads. In *IISWC*. 2013.

[14] Christina Delimitrou and Christos Kozyrakis. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Proc. of ASPLOS*. 2013.

[15] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *ASPLOS*. 2014.

[16] Christina Delimitrou and Christos Kozyrakis. HCloud: Resource-Efficient Provisioning in Shared Cloud Systems. In *Proceedings of ASPLOS*, April 2016.

[17] Christina Delimitrou, Daniel Sanchez, and Christos Kozyrakis. Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters. In *Proceedings of SOCC*, August 2015.

[18] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. X-trace: A pervasive network tracing framework. In *Proceedings of NSDI*, 2007.

[19] Yu Gan and Christina Delimitrou. The Architectural Implications of Cloud Microservices. In *CAL, vol.17, iss. 2*, Jul-Dec 2018.

[20] Yu Gan, Meghna Pancholi, Dailun Cheng, Siyuan Hu, Yuan He, and Christina Delimitrou. Seer: Leveraging Big Data to Navigate the Complexity of Cloud Debugging. In *Proc. of HotCloud*, 2018.

[21] Yu Gan, Yanqi Zhang, and et al. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems. In *Proceedings of ASPLOS*, April 2019.

[22] Yu Gan, Yanqi Zhang, Kelvin Hu, Yuan He, Meghna Pancholi, Dailun Cheng, and Christina Delimitrou. Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices. In *Proceedings of ASPLOS*, April 2019.

[23] Hiranya Jayathilaka, Chandra Krintz, and Rich Wolski. Performance monitoring and root cause analysis for cloud-hosted web applications. In *Proceedings of WWW*, 2017.

[24] Norman P. Jouppi, Cliff Young, Nishant Patil, and et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017.

[25] Harshad Kasture and Daniel Sanchez. Ubik: Efficient Cache Sharing with Strict QoS for Latency-Critical Workloads. In *ASPLOS*, 2014.

[26] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In *Proc. of ISCA*. 2015.

[27] Jason Mars and Lingjia Tang. Whare-map: heterogeneity in "homogeneous" warehouse-scale computers. In *Proceedings of ISCA*. 2013.

[28] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. In *Proceedings of ISCA*, pages 319–330, 2011.

[29] Charles Reiss, Alexey Tumanov, Gregory Ganger, Randy Katz, and Michael Kozych. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of SOCC*. 2012.

[30] Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, and Robert Hundt. Google-wide profiling: A continuous profiling infrastructure for data centers. *IEEE Micro*, pages 65–79, 2010.

[31] S. Sarwar, A. Ankit, and K. Roy. Incremental learning in deep convolutional neural networks using partial network sharing. In *arXiv:1712.02719*.

[32] Benjamin H. Sigelman, Luiz André Barroso, and et al. Dapper, a large-scale distributed systems tracing infrastructure. Tech report, 2010.

[33] Lalith Suresh, Peter Bodik, Ishai Menache, Marco Canini, and Florin Ciucu. Distributed resource management across process boundaries. In *Proceedings of SOCC*. 2017.

[34] Xiao Yu, Pallavi Joshi, Jianwu Xu, Guoliang Jin, Hui Zhang, and Guofei Jiang. Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs. In *Proceedings of ASPLOS*, 2016.