Sage: Leveraging ML To Diagnose Unpredictable Performance in Cloud Microservices

Yu Gan*
Cornell University

Sundar Dev
Google

David Lo
Google

Christina Delimitrou
Cornell University

Abstract

Cloud applications are increasingly shifting from large monolithic services, to complex graphs of loosely-coupled microservices. Despite their advantages, microservices also introduce cascading QoS violations in cloud applications, which are difficult to diagnose and correct.

We present *Sage*, a ML-driven root cause analysis system for interactive cloud microservices. Sage leverages unsupervised learning models to circumvent the overhead of trace labeling, determines the root cause of unpredictable performance online, and applies corrective actions to restore performance. On experiments on both dedicated local clusters and large GCE clusters we show that Sage achieves high root cause detection accuracy and predictable performance.

1 Introduction

Cloud computing has reached proliferation by offering *resource flexibility*, *cost efficiency*, and *fast deployment* [13, 15, 18, 20, 26, 37]. As the cloud scale and complexity increased, cloud services have undergone a major shift from large monolithic designs to complex graphs of single-concerned, loosely-coupled microservices. This shift is becoming increasingly pervasive, with large cloud providers, such as Amazon, Twitter, Netflix, and eBay having already adopted this application model [2, 11, 12]. Microservices are appealing for several reasons, such as facilitating development, promoting elasticity, and enabling software heterogeneity.

Despite their advantages, microservices also complicate resource management, as dependencies between tiers introduce backpressure, causing unpredictable performance to propagate through the system [23, 25]. Diagnosing such issues empirically is cumbersome and prone to errors, especially as typical microservices deployments include hundreds/thousands of tiers. Similarly, current cluster managers [19, 20, 32–34, 36, 37, 42, 43, 47, 52, 53]. are not expressive enough to account for the impact of dependencies, putting more pressure on the need for automated root cause analysis systems.

Over the past few years, there has been increased attention on trace-based methods to analyze [21], diagnose, and in some cases anticipate [22, 25] performance issues in cloud services. While most of these systems target cloud applications, the only one focusing on microservices is Seer [25], a DL-based system that anticipates cases of unpredictable performance by inferring the impact of outstanding requests on end-to-end performance. Despite its high accuracy, Seer leverages supervised learning to anticipate QoS violations, which require offline and online trace labeling. In a production system, this is non-trivial, as it involves injecting resource contention in live applications, hurting user experience.

We present Sage, a root cause analysis system for interactive microservices that leverages unsupervised learning to identify the culprit of unpredictable performance in complex graphs of microservices. 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 instrumentaion. We have evaluated Sage both on dedicated local clusters and large GCE settings and showed high root cause detection accuracy and improved performance predictability.

2 ML for Performance Debugging

2.1 Overview

Sage is a performance debugging and root cause analysis system for large-scale cloud applications. While the design centers around interactive microservices, where dependencies between tiers are more impactful, Sage is also applicable to traditional monolithic or SOA services. Sage relies on two broad techniques, each of which is described in detail below; first, it uses Causal Bayesian Networks (CBN) to model the RPC-level dependencies between microservices, and the latency propagation from the backend to frontend. Second, it uses a graphical variational auto-encoder (GVAE) to generate examples of counterfactuals [40], and infer the hypothetical end-to-end latency had some occurring events not happened. Using these two techniques, Sage determines which set of microservices initiated an end-to-end QoS violation, and moves to adjust deployment and/or resource allocation to correct it.

2.2 Microservice Latency Propagation

Multiple RPCs between microservices form a tree of nested traces in a distributed monitoring system. Figure 1 shows an example of the RPC dependency graph containing five services, four RPCs, and its corresponding latency traces.

The server-side latency of any non-leaf RPC is determined by the processing time of the RPC itself and the waiting time

*This work was not done at Google.

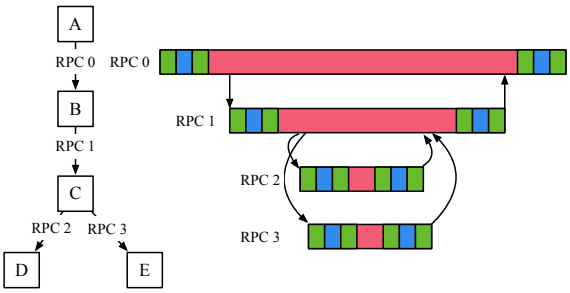

Figure 1. The dependency graph and traces of nested RPCs.

(i.e., client-side latency) of its child RPCs. The latency of any RPC will propagate through the RPC dependency graph to the frontend and impact the end-to-end latency. Since the latency of a child RPC cannot propagate to its parent without impacting its own latency, the RPC latency propagation follows a *local Markov property*, where each latency variable is conditionally independent on its non-ancestor RPC latencies, given its child RPC latencies [31]. For instance, the latency of `RPC0` is conditionally independent on the latency of `RPC2` and `RPC3`, given the latency of `RPC1`.

2.3 Modeling Microservice Dependency Graphs

Causal Bayesian Networks (CBN) are a common tool to model causal relationships [44, 46]. A CBN is a directed acyclic graph (DAG), where the nodes are different random variables and the edges indicate their conditional dependencies. More specifically, each edge represents a causal relationship, directed from the cause to the effect. We define three types of nodes in the Bayesian network:

- **Metric nodes** (X): The metric nodes contain primarily resource-related metrics of all services and network channels, such as CPU and memory utilization and network bandwidth, collected via the in-place monitoring tools [14, 17, 48].
- **Latency nodes** (Y): The latency nodes include client-side latency (Y^c), server-side latency (Y^s), and request/response network delay (Y^{req} and Y^{resp}) of all RPCs.
- **Latent variables** (Z): The latent variables contain the unobserved and immeasurable factors that are responsible for latency stochasticity. They are critical to generate the counterfactual latencies Sage relies on to diagnose root causes of QoS violations.

We can construct the CBN among the three-node classes of all RPCs based on the inherent causal relationships and latency propagation observations obtained via a distributed tracing system, such as Dapper or Zipkin.

Figure 2 shows an example of the CBN constructed for a three-microser-vice chain based on the RPC dependency graph. The nodes with solid lines (X and Y) are observed, while the nodes with dashed lines (Z) are latent variables that need to be inferred. The arrows in the RPC dependency graph and CBN have opposite directions because the latency of one RPC is determined by the latency of its child RPCs.

2.4 Counterfactual Queries

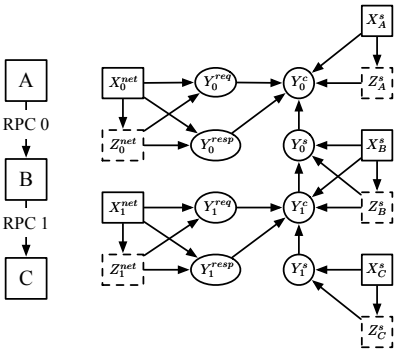

Figure 2. Latency propagation CBN.

In a typical cloud environment, site reliability engineers (SREs) can verify if a suspected root cause is correct by reverting a microservice's configuration to a state known to be safe, while keeping the remaining microservices unchanged. If the problem is resolved, the suspected culprit is causally related to the QoS violation. Sage uses a similar process, where "suspected root causes" are generated using counterfactuals, which determine the causal effect by asking what the outcome would be if the state of a microservice had been different [27, 38, 41].

To avoid impacting the performance of live services during this process, Sage leverages historical tracing data to generate realistic counterfactuals [41, 46], taking into account that the exact situation may not have occurred in the past. If the probability that the end-to-end tail latency meets QoS after intervention is greater than a threshold, then those services are the root cause of the performance issue.

Conditional deep generative models, such as the conditional variational autoencoders (CVAE) [50] and conditional generative adversarial nets [39], are common tools to generate new data of a class from an original distribution. Generally, they compress a high-dimensional target (Y) and tag (X) into low-dimensional latent space variables (Z), and use the latent space variables and tags to generate new target data. Recent studies have showed that these techniques can also be used to generate counterfactuals for causal inference [35, 55].

To generate counterfactuals, we build a network of CVAEs according to the structure of the CBN. Although using one CVAE for the entire microservice graph would be simple, it has several drawbacks. First, it lacks the CBN's structural information which is useful in terms of avoiding ineffectual counterfactuals based on spurious correlations. Second, it prohibits partial retraining and transfer learning, which is essential given the frequent update cadence of microservices. Finally, the black-box model is less explainable since it cannot reveal any information on how the latency of a problematic service propagates to the frontend. Therefore, we construct one lightweight CVAE per microservice, and connect the different CVAEs according to the structure of the CBN to form the graphical variational autoencoder (GVAE).

The encoders and prior networks take the observed metrics as inputs, and are trained in parallel. The decoders require the outputs of the parent decoders in the CBN as inputs, and are

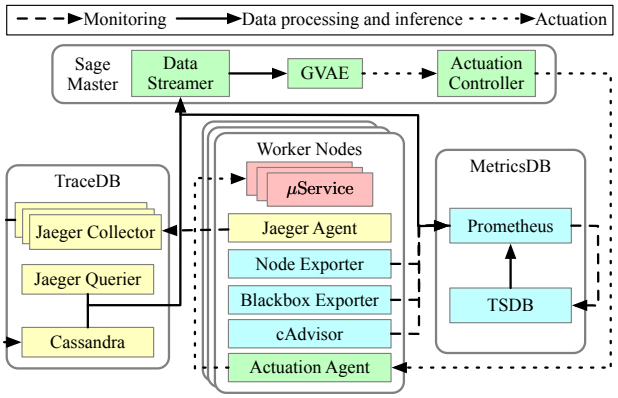

Figure 3. Overview of Sage's system design.

trained serially. The maximum depth of the CBN determines the maximum number of serially-cascaded decoders.

3 Sage Design

Sage is a root cause analysis system for complex graphs of interactive microservices. Sage relies on RPC-level tracing to compose the Causal Bayesian network of the microservice topology, and per-node tracing to track the per-tier latency distribution. Below we discuss the training and inference pipeline (Sec. 3.1), Sage's actuation system once a root cause has been identified (Sec. 5.2), and the way Sage handles changes in application design (Sec. 3.3).

Fig. 3 shows an overview of Sage. The system uses Jaeger [5], a distributed RPC-level tracing system to collect end-to-end request execution traces. Jaeger uses sampling to reduce tracing oveheads. Sage also uses the Prometheus Node Exporter [8], Prometheus Blackbox Exporter [7], and cAdvisor [4] to collect machine- and container-level hardware/OS metrics, and network latencies. Each metric's timeseries is stored in the Prometheus TSDB [6]. At runtime, Sage queries Jaeger and Prometheus periodically to obtain real-time latency and usage metrics. The GVAE then uses this data to infer the root cause of any QoS violation(s). Once a root cause is diagnosed, Sage's actuator takes action to restore performance by adjusting the offending microservice's resources.

Sage uses a centralized master for trace processing and root cause analysis, and per-node agents for trace collection and container deployment. It also maintains two hot stand-by copies of the Sage master for fault tolerance.

3.1 Root Cause Analysis

Sage first uses the Data Streamer to fetch and pre-process the latency and usage statistics. Sage initializes and trains the GVAE model offline with all initially available data. It then periodically retrains the model, even when there are no changes in application design, to account for changes in user behavior [16, 28, 45, 54]. Online learning models are prone to *catastrophic forgetting*, where the model forgets previously-learned knowledge upon learning new information [30, 45]. To avoid this, we interleave the current and previous data in the training batches. In addition, to avoid *class imbalance*, i.e., cases where the datapoints that meet QoS are significantly more than those which violate it, the model oversamples the minority classes to create a more balanced training dataset.

At runtime, Sage uses the latest version of the GVAE model to infer the root cause of QoS violations. Sage first calculates the medians of usage and performance metrics where QoS is met, and labels them as *normal values*. If at any point QoS is not met, the GVAE will generate counterfactuals by replacing a microservice's metrics with their respective *normal values*. The service whose counterfactual would resolve the QoS violation is identified as the culprit behind it.

Sage implements a two-level approach to locate the root cause of a QoS violation. It first uses service-level counterfactuals to locate the culprit microservices, and then uses metric-level counterfactuals of the offending service to identify the underlying reason that caused it to become the culprit.

3.2 Actuation

Once Sage determines the root cause of a QoS violation it takes action. Depending on which resource is identified by the GVAE as the one instigating the QoS violation, Sage will dynamically adjust the CPU frequency, scale up or scale out the problematic microservices, limit the rate of the collocated interference jobs, partition the last level cache (LLC) with Intel Cache Allocation Technology (CAT), and partition the network bandwidth with Linux traffic control queuing disciplines. Sage first tries to resolve the performance issue by only adjusting resources on the offending node, and only when that is insufficient it scales out the problematic microservice on new nodes and/or migrate it.

3.3 Handling Microservice Updates

Training the complete model from scratch for large clusters takes tens of minutes to hours, so it is impractical to happen for every change in application design/deployment. Sage instead implements *selective partial retraining* and *incremental retraining* with a dynamically reshapable GVAE similar to [54], thanks to VAE's ability to be decomposed using the CBN. On the one hand, with selective partial retraining, we only retrain the neurons corresponding to the updated nodes and their descendents in the CBN, because the causal relationships guarantee that all other nodes are not affected by the change. On the other hand, with incremental retraining, we initialize the parameters of the network to those of the previous model, while adding/removing/reshaping the corresponding networks if microservices are added/dropped/updated. The combination of these two *transfer learning* approaches reduces retraining time by more than 10×, especially when there is large fanout in the RPC dependency graph.

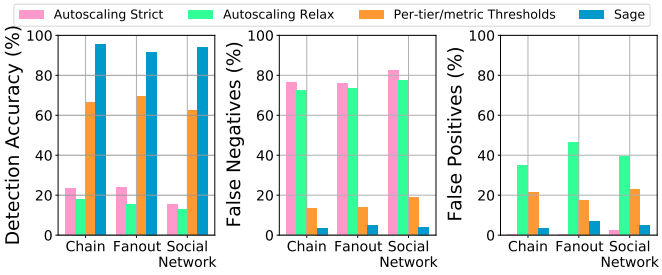

Figure 4. Detection accuracy and false negatives/positives.

4 Methodology

4.1 Cloud Services

Generic Thrift microservices: *Apache Thrift* [1, 49] is a popular RPC framework. We implement a code generator for composable Thrift microservices, and evaluate two topologies; a *Chain* and a *Fanout*. In Chain, each microservice receives a request from its parent, sends it to its downstream service, and responds to its parent once it gets the results from its child. In Fanout, the root service broadcast each request to all leaf services and returns to the client only after all children have responded. Most real microservice architectures are combinations of these two topologies [24, 29, 51].

Social Network: One of the end-to-end microservice in the DeathStarBench suite [24] that implements a broadcast-style social network with uni-directional follow relationships.

4.2 Systems

Local Cluster: We use a dedicated local cluster with five 2-socket 40-core servers with 128GB RAM each, and two 2-socket 88-core servers with 188GB RAM each. Each server is connected to a 40Gbps ToR switch over 10Gbe NICs.

Google Compute Engine: We also deploy the Social Network service to Google Compute Engine (GCE) with 84 nodes in `us-central1-a` to study Sage's scalability. All nodes are dedicated, so there is no interference from external jobs.

4.3 Training Dataset for Validation

We use wrk2 [3] as the open-loop HTTP workload generator for all three applications. To verify the ground truth during validation, we use `stress-ng` [9] and `tc-netem` [10] to inject CPU-, memory-, disk-, and network-intensive microbenchmarks to different, randomly-chosen subsets of microservices.

5 Evaluation

5.1 Sage Validation

Fig. 4 shows the accuracy of Sage in the local cluster across services, compared to two autoscaling techniques, and an oracular scheme that sets thresholds for each tier and metric offline, beyond which point resources are upscaled. *Autoscale Strict* increases resource allocation when the utilization of a microservice exceeds 50% and *Autoscale Relaxed* when it exceeds 70% (on par with AWS's autoscaling policy). Sage significantly outperforms the other methods, even the offline oracular one, by learning the impact of dependencies between neighboring microservices. Similarly, Sage's false negative and false positive rates are marginal, which avoids QoS violations and resource inefficiencies respectively.

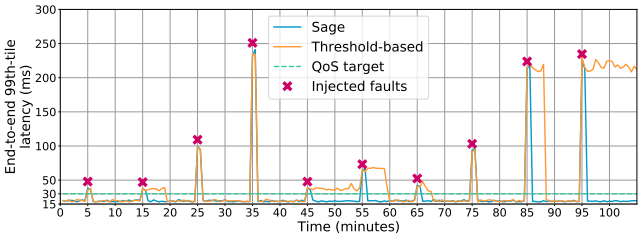

Figure 5. End-to-end latency for Social Network in the presence of QoS violations.

5.2 Actuation

Fig. 5 shows the impact of root cause analysis on the end-to-end tail latency of the Social Network, with Sage and the offline oracular technique. To create unpredictable performance whose source is known, we inject resource intensive kernels in a randomly-selected subset of microservices. While there are cases that the threshold-based scheme identifies correctly, more often performance takes a long time to recover. In comparison, Sage immediately identifies the correct root cause, and applies corrective action to restore performance.

We have also introduced changes to several microservices and have validated that the transfer learning in Sage reduces training time by at least an order of magnitude compared to retraining from scratch, without impacting accuracy.

5.3 Scalability

We now evaluate Sage's accuracy when deploying the Social Network on 188 instances on GCE using Docker Swarm. We replicate the stateless microservices and shard the caching systems and databases across instances during periods of higher load. Each service has 1-10 replicas, depending on the maximum single-process throughput. Accuracy on GCE is within 1% of the detection accuracy on the local cluster, while the difference in false positives and false negatives is also marginal. We also evaluate the difference in training and inference time between the local cluster and GCE. We use two Intel Xeon 6152 processors with 44 cores in total for training and inference. Training from scratch takes 124 minutes for the local cluster and 148 minutes for GCE. Inference takes 49ms on the local cluster and 62ms on GCE. Although we deploy 6.7x more containers on GCE than on the local cluster, the training and inference time only increase by 19.4% and 26.5% respectively.

6 Conclusions

We have presented Sage, an ML-driven root cause analysis system for interactive, cloud microservices. Sage leverages

entirely unsupervised models to detect the sources of unpredictable performance, removing the need for empirical diagnosis or expensive data labeling. Sage adapts to frequent design changes, and takes action to restore QoS. We shows that Sage achieves high root cause detection accuracy and improved performance. Given the increasing complexity of cloud services, data-driven systems like Sage can improve performance predictability without sacrificing efficiency.

References

[1] "Apache thrift," https://thrift.apache.org.

[2] "Decomposing twitter: Adventures in service-oriented architecture," https://www.slideshare.net/InfoQ/decomposing-twitter-adventures-in-serviceoriented-architecture.

[3] "giltene/wrk2," https://github.com/giltene/wrk2.

[4] "google/cadvisor," https://github.com/google/cadvisor.

[5] "Jaeger: open source, end-to-end distributed tracing," https://www.jaegertracing.io/.

[6] "Prometheus," https://prometheus.io/.

[7] "prometheus/blackbox_exporter," https://github.com/prometheus/blackbox_exporter.

[8] "prometheus/node_exporter," https://github.com/prometheus/node_exporter.

[9] "stress-ng," https://wiki.ubuntu.com/Kernel/Reference/stress-ng.

[10] "tc-netem(8) - linux manual page," http://man7.org/linux/man-pages/man8/tc-netem.8.html.

[11] "The evolution of microservices," https://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference, 2016.

[12] "Microservices workshop: Why, what, and how to get there," http://www.slideshare.net/adriancockcroft/microservices-workshop-craft-conference.

[13] "Amazon ec2," http://aws.amazon.com/ec2/.

[14] M. Azure, *Azure Monitor documentation*, 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-monitor/

[15] L. Barroso and U. Hoelzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. MC Publishers, 2009.

[16] Z. Chen and B. Liu, "Lifelong machine learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 12, no. 3, pp. 1–207, 2018.

[17] G. Cloud, *Cloud Monitoring documentation*, 2020. [Online]. Available: https://cloud.google.com/monitoring/docs/apis

[18] J. Dean and L. A. Barroso, "The tail at scale," in *CACM, Vol. 56 No. 2*.

[19] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Houston, TX, USA, 2013.

[20] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-Efficient and QoS-Aware Cluster Management," in *Proc. of ASPLOS*. Salt Lake City, 2014.

[21] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "X-trace: A pervasive network tracing framework," in *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, ser. NSDI'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 20–20.

[22] Y. Gan, M. Pancholi, D. Cheng, S. Hu, Y. He, and C. Delimitrou, "Seer: Leveraging Big Data to Navigate the Complexity of Cloud Debugging," in *Proceedings of the Tenth USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, July 2018.

[23] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems," in *Proceedings of the Twenty Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, April 2019.

[24] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems," in *Proceedings of the Twenty Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, April 2019.

[25] Y. Gan, Y. Zhang, K. Hu, Y. He, M. Pancholi, D. Cheng, and C. Delimitrou, "Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices," in *Proceedings of the Twenty Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, April 2019.

[26] "Google container engine," https://cloud.google.com/container-engine.

[27] M. Höfler, "Causal inference based on counterfactuals," *BMC medical research methodology*, vol. 5, no. 1, p. 28, 2005.

[28] S. C. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," *arXiv preprint arXiv:1802.02871*, 2018.

[29] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3302424.3303958

[30] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[31] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[32] C.-C. Lin, P. Liu, and J.-J. Wu, "Energy-aware virtual machine dynamic provision and scheduling for cloud computing," in *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing (CLOUD)*. Washington, DC, USA, 2011. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2011.94

[33] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceedings of the 41st Annual International Symposium on Computer Architecuture (ISCA)*. Minneapolis, MN, 2014.

[34] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proc. of the 42Nd Annual International Symposium on Computer Architecture (ISCA)*. Portland, OR, 2015.

[35] C. Louizos, U. Shalit, J. M. Mooij, D. Sontag, R. Zemel, and M. Welling, "Causal effect inference with deep latent-variable models," in *Advances in Neural Information Processing Systems*, 2017, pp. 6446–6456.

[36] J. Mars and L. Tang, "Whare-map: heterogeneity in "homogeneous" warehouse-scale computers," in *Proceedings of ISCA*. Tel-Aviv, Israel, 2013.

[37] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *Proceedings of the 38th annual international symposium on Computer architecture*, 2011, pp. 319–330.

[38] P. Menzies, "Counterfactual theories of causation," *Stanford Encyclopedia of Philosophy*, 2008.

[39] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014.

[40] M. Moore, "Causation in the law," in *The Stanford Encyclopedia of Philosophy*, winter 2019 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2019.

[41] S. L. Morgan and C. Winship, *Counterfactuals and causal inference*. Cambridge University Press, 2015.

[42] R. Nathuji, C. Isci, and E. Gorbatov, "Exploiting platform heterogeneity for power efficient data centers," in *Proceedings of ICAC*. Jacksonville, FL, 2007.

[43] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for qos-aware clouds," in *Proceedings of EuroSys*. Paris,France, 2010.

[44] R. E. Neapolitan *et al.*, *Learning bayesian networks*. Pearson Prentice Hall Upper Saddle River, NJ, 2004, vol. 38.

[45] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, 2019.

[46] J. Pearl *et al.*, "Causal inference in statistics: An overview," *Statistics surveys*, vol. 3, pp. 96–146, 2009.

[47] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proceedings of EuroSys*. Prague, 2013.

[48] A. W. Services, *Amazon CloudWatch User Guide Document History*, 2020. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring

[49] M. Slee, A. Agarwal, and M. Kwiatkowski, "Thrift: Scalable cross-language services implementation," *Facebook White Paper*, vol. 5, no. 8, 2007.

[50] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 3483–3491.

[51] A. Sriraman and T. F. Wenisch, "μ suite: A benchmark suite for microservices," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018, pp. 1–12.

[52] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[53] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: precise online qos management for increased utilization in warehouse scale computers," in *Proceedings of ISCA*. 2013.

[54] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=Sk7KsfW0-

[55] J. Yoon, J. Jordon, and M. van der Schaar, "GANITE: Estimation of individualized treatment effects using generative adversarial nets," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=ByKWUeWA-