KubeKlone: A Digital Twin for Simulating Edge and Cloud Microservices

Ayush Bhardwaj
Brown University
ayush_bhardwaj@brown.edu

Theophilus A. Benson
Brown University
tab@cs.brown.edu

ABSTRACT

Microservices are terraforming the computing landscape with web-scale infrastructures (e.g., Facebook, Google, Amazon) and telecom infrastructures (e.g., ATT, Ericsson) adopting them. At it's core, the microservices paradigm promotes a decoupling of applications into multiple services – a decoupling that promotes better scalability, fault-tolerance, and deployability. Unfortunately, this decoupling significantly increases the space of configuration options and performance problems, rendering traditional approaches to management ineffective. Recent efforts to address this problem embrace Artificial Intelligence for IT Operations (AIOps). However, training effective AI models requires significant amounts of data and, in some instances, a framework for quickly exploring or analyzing model performance. Digital twins, or simulators, have effectively enabled AI-based management frameworks within other domains (e.g., manufacturing, industrial and automotive).

In this paper, we propose the design of KubeKlone, the first comprehensive and opensource digital twin for modeling cloud-native microservices applications. KubeKlone is motivated by our need for accurate, efficient, and general model training. KubeKlone satisfies these goals by decoupling the simulation of microservices from the training of machine learning (ML) models while simultaneously ensuring efficiency and simplifying model design. In particular, KubeKlone introduces a queue-based simulator that abstracts infrastructure details and focuses on modeling, with queues, resource contentions across host and network components. To simplify model design, KubeKlone provides interfaces that hide simulator details and provides wrappers around popular ML packages. To illustrate the strengths of KubeKlone, we validate it against a deployment on Google Cloud Engine and implement several AIOps resource management algorithms (including PARTIES).

ACM Reference Format:
Ayush Bhardwaj and Theophilus A. Benson. 2022. KubeKlone: A Digital Twin for Simulating Edge and Cloud Microservices. In *6th Asia-Pacific Workshop on Networking (APNet 2022), July 1–2, 2022, Fuzhou, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3542637.3542642

APNet 2022, July 1–2, 2022, Fuzhou, China

ACM ISBN 978-1-4503-9748-3/22/07…$15.00
https://doi.org/10.1145/3542637.3542642

1 INTRODUCTION

The recent shifts to microservice-based architectures [2, 12, 54] radically disrupt the landscape of infrastructure management tasks. While the adoption of microservices improves scalability, utilization, fault-tolerance, and deployment speeds, they do not themselves provide strong management primitives. In fact, effectively managing microservices requires diverse, highly specialized, and often adaptive techniques and algorithms to monitor, adapt, and optimally configure them. However, modern IT management frameworks are relatively static and rigid.

Motivated by the rigidity of existing management frameworks, cloud operators (e.g., IBM [3], AppDynamics [53], VMWare [43]) and web-scale providers are shifting towards data-driven control loops - AIOps system (artificial intelligence for IT operations) [4, 14, 36]. These AI-based algorithms provide a distinct advantage over traditional hand-tuned heuristics because they can learn optimal behavior while adapting to workload, application, and infrastructure changes. Moreover, they adapt by ingesting a large number of metrics from the loosely coupled microservices. While training these learning algorithms requires a significant amount of data, experimenting on infrastructure deployments is an ill-suited method for capturing learning data for the following reasons: first, such deployments are often very costly (e.g., the cost increases linearly with variation in request per second [30]) and second, they are extremely cumbersome [22, 33] (i.e., effort and time to correctly set up the virtual machine, the service, add load balancers, etc.).

This paper addresses these problems by designing a general simulation-based digital twin for microservices-based cloud-native infrastructures. The automotive, manufacturing industries, and indeed networking [38, 40, 57] inspire our approach. These industries have created digital twins of their infrastructures when faced with similar challenges to enable machine learning at scale. KubeKlone's design addresses the following key challenges. First, effectively and accurately simulating a broad set of microservices requires simulating the microservice, but also the underlying infrastructure to capture resource contention; and, to support the infrastructure services unique to microservice deployments (e.g., the Service Mesh and various sidecars). Second, simplifying the design of a broad set of AIOps learning algorithms requires introducing abstractions to hide simulator complexity and reduce the barrier to developing custom algorithms.

KubeKlone is a queuing-based network simulator based on *uqSim* [59] which overcomes several of *uqSim*'s key limitations: (i) by introducing network-specific queues and global stateful events to support global orchestration and the associated network resource contention, (ii) by simulating various sidecars and third party services to faithfully support a broad range of microservices, and (iii) by abstracting

simulator details behind carefully designed interfaces to simplify the design of general AIOps model.

Our main contributions are as follows:

- We present the design of a novel simulation-based framework, KubeKlone, for simplifying the design, development, and testing of microservices-based AIOps learning algorithms.
- We identify key attributes and properties of a microservices deployment that must be included in any simulator to model modern microservices accurately.
- We demonstrate the accuracy and precision of KubeKlone by implementing several AIOps resource management techniques within KubeKlone and evaluating their performance relative to a cluster deployment on Google cloud engine.

2 BACKGROUND

The microservices-based approach to designing applications decouples an application into a set of loosely coupled components: each component running within its own container. This section briefly summarizes the properties of a microservices-based deployment (Section 2.1), the challenges associated with effectively managing and modeling microservices-based deployment (Section 2.2), and the impact of shifting to microservices (Section 2.3).

2.1 Architecture

In a microservices-based application, the different components are called *services*. A request flows through the services that comprise an application in a prespecified pattern that is captured by a *service graph*. More formally, the service graph is a directed acyclic graph that captures the communication patterns between the services within an application.

As requests flow through each service within an application, the requests are subject to contention and interference due to competition for host and network resources. The performance and processing time for a request is thus a function of both service processing times and resource contention. Contention occurs at both the infrastructure (i.e., server level) and the platform level (i.e., network level).

> **Implications:** To accurately model the performance of a microservice, a simulator or emulator must capture both service processing times as well as the various forms of contention while replicating the flow embedded within the service graph.

2.2 Service Types

A microservice is often described as a collection of loosely coupled services developed by the application developer. However, in reality, many practical deployments consist of a non-trivial number of services that are designed or developed by external or their party entities. In particular, there are two types of such services.

SideCar Services: These services encapsulate and provide common features, e.g., security, authentication, load balancing, required by many services. Traditionally, such features would be encapsulated in a library and built into the application. However, in many microservices deployments, such features are deployed in a separate container colocated alongside each microservice container — this deployment pattern is called a *sidecar*. The sidecar acts as a proxy for the microservice intercepting all external network communication in order to provide serialization, security, encryption, logging, configuration management, and service discovery. This pattern of deploying support and management functionality in its own container introduces performance overheads while upholding the core microservices principles of decoupling and independence. In Table 1, we briefly highlight interesting functionality that is currently deployed wihin SideCars.

SideCar	Property
Istio [21], Linkerd [39], Consul [9]	Service Mesh
Nginx [8], Kong Gateway [24]	API Gateway
Fluentd [27], Elasticsearch [11]	Log Aggregator
Sensu [44], Telegraf [5]	Monitoring

Table 1: Representative list of SideCars and functionality provided.

Two key sidecars standout:

- **Service Mesh:** provides communication, service discovery, security, traffic management, observability, and replication between different services within an organization without requiring separate teams to reimplement redundant functionality.
- **API-Gateway:** provides uniform access and control between external clients and an organization's services without requiring separate teams to reimplement access and control functionality themselves. Clients issue requests to replicated Gateways, which relays the requests to the appropriate services. Gateways provide authentication (Open Policy Agent), encryption (TLS, mTLS), traffic management, and observability for microservices.

A key distinction between the Service Mesh and APIGateway is that Service Mesh provide management over internal traffic between services whereas APIGateways provide management over external traffic between clients and services.

External Services: The salient feature of this class of services is that they are deployed and hosted on external platforms, e.g., Amazon S3 or Facebook API calls. Thus, developers have little insight or control over their configurations or internal logic (source code). In essence, these services should be viewed as black-box services which process RPC or REST calls.

> **Implications:** To generalize to a broad range of microservices-based applications, a simulator or emulator must model both services created by in-house developers, as well as, third-party and external services. More importantly, failure to model the sidecar services fundamentally compromises the accuracy and precision of any microservices simulator or emulator.

2.3 Management Implications of Microservices

The shift to a microservices-based deployment has several implications for managing, operating, and running applications. First, decoupling an application into multiple components increases the

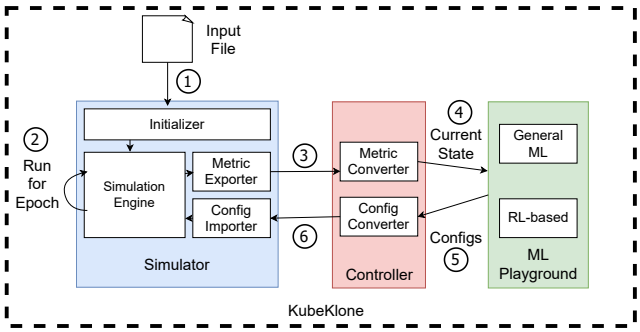

Figure 1: Components of KubeKlone (a) Simulator, (b) Controller, (c) ML Playground.

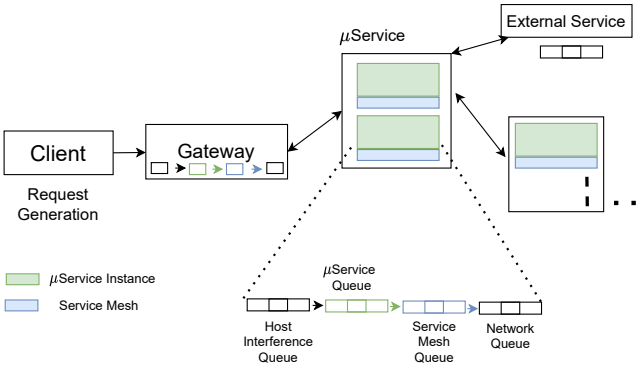

Figure 2: Different Queues in KubeKlone.

number of entities that operators need to manage. For example, while many web-scale companies have three distinct applications when migrated to a microservices-based architecture, these companies have 100-1000s of components (e.g., Uber claims 1000s). Second, the independence afforded by this decoupling introduces heterogeneity, e.g., programming languages or coding pratices (Netflix [35], Google [45], eBay [45]), which complicates management as operators need to design unique tools for each microservice. Third, the independence also enables constant code changes: whereas traditional web companies deploy code every couple of months, microservice-based companies deploy code every couple of hours or days (Amazon [46], Netflix [18]). A key implication of this rapid velocity is that operators have less time to develop effective management scripts.

> **Implications:** The microservice paradigm results in operators configuring a large and more complex infrastructure than they are used to configuring. Crucially, operators need to do this in significantly less time than they would normally need. To this end, infrastructure operators are turning to AIOps (Artificial Intelligence for IT Operations) to address these growing concerns.

3 DESIGN

Our design of KubeKlone is motivated by the following design goals:

- **Fast:** Our simulator must be fast to ensure that machine learning algorithms, e.g., Deep Reinforcement learning, can quickly explore the large number of scenarios required to learn about the microservice meaningfully.
- **General:** Our framework must be amenable to a broad range of machine learning algorithms and microservice deployment scenarios.
- **High Fidelity:** Most importantly our simulator must accurately and precisely capture the various attributes of a canonical microservices deployment ranging from the infrastructure and platofrm to the service level.

Our initial two goals, *fast and general*, stand at odds with each other: fast implementations are often specialized and tailored to work in concertion with specific ML frameworks (e.g., uqSim is optimized to work for energy optimizations). We tackle this by decoupling KubeKlone into two components, the simulator, and the MLPlayground, which are designed and developed around different key principles but coordinate through IPCs.

The simulator addresses high fidelity and speed by simulating the infrastructure using a combination of queues and events. In particular, we choose to abstract all infrastructure components at the request level and model various points of resource contention using queues. These choices significantly reduce the number of events that our simulator needs to process.

To provide high fidelity, we model both host-level and network-level dynamics and resource contention points. Specifically (as Shown in Figure 2), we model resource contention due to CPU scheduling using a set of Host-contention queues, we model network interference using network-contention queues, we model host microservice processing using microservice queues, and we model service orchestration using service mesh queues. These queues are independently configurable and support different types of stateful and non-stateful events. Of particular interest are the service mesh queues which include global stateful events that control traffic engineering, fault-tolerance, and service routing.

To capture external services, e.g., Amazon S3 block storage or third-party API calls, we include a special queue type for these external services. Unlike internal services, where we can capture and thus model both host and network dynamics, operators have no visibility into external services, and thus, we model each external service as one queue which captures the service's serving times. Although this abstraction conflates host and network-level dynamics, this does, in fact, accurately capture the developer's perspective of an external service – i.e., a black box that responds to requests.

In Figure 1, we present a high-level architecture for KubeKlone. From this figure, we observe both the decoupling between the simulator and the ML playground as well as the controller which mediates communication between them.

System Workflow: KubeKlone divides time into a series of epochs which are configurable periods. At the start of an epoch, the simulator reads configurations, which specifies parameters for configuring the queues and events, then uses these parameters to simulate the services (Figure 1 Step 2). At the end of an epoch, the simulator captures and exports various performance metrics (Figure 1 Step 3). Next, KubeKlone's controller converts the performance metrics into python objects that present the current state of the simulator and then calls the appropriate methods in the interfaces to run the appropriate ML control loop within the ML Playground (Figure 1 Step

4). Subsequently, the controller captures the actions (configuration parameters) generated by the ML algorithms (Figure 1 Step 5) and converts them into a format appropriate for the simulator (Figure 1 Step 6) and then runs the simulator for the next epoch. To initialize the simulator; we provide a set of configuration files that are read during the initialization phase (Figure 1 Step 1).

The **simulator component** (illustrated in Figure 2) models each service within a service graph, except for the *external service*, using a combination of four queues. In particular, a request flows through the host-interference queues for modeling CPU-specific interference, the microservice queues for modeling service-specific processing, the service mesh queues for traffic engineering, routing, and finally, the networking queues for modeling network contention. The behavior of the different queues and events are defined in a set of configurations that can be dynamically modified at runtime. The simulator captures performance metrics and counters from the queues and aggregates them to provide varying levels of detail (Section 3.1).

The **MLPlayground component** provides two different interfaces that abstract and hide the details of the underlying simulator from the ML-Algorithm designers. First, we provide a general ML interface that can support Bayesian optimizations, decision trees, and other non-DeepRL algorithms. This interface consists of two key functions to be utilized by the developers, i.e., (a) `init` - which initializes internal data structures to capture and encapsulate the service configuration used by the simulator and service metrics created by the simulator, (b) `run_agent` - which runs the agent with the desired optimization algorithm (e.g., BO, random) utilizing the data-structures (e.g., state, action) initialized in the `init` function. The second interface is a niche interface that supports the growing interest in DeepRL based AIOPs system [15, 16]. This interface require the developers to use three key functions (a) `create_environment` - which creates desired environment with custom properties (e.g. maximum number of training episodes) for the RL agent where it performs the actions, (b) `create_agent` - which creates a custom agent with the required properties (e.g., learning rate, objective function), (c) `run_agent` - which runs the agent for specified number of episodes, states, and actions. The key difference between the two interfaces is that the specialized interface encapsulates a python deep-learning library (i.e., TensorFlow). The MLPlayground can be utilized both for training and inference.

3.1 Metrics and Configuration

Next, we briefly summarize the metrics and configurations/control parameters currently supported by KubeKlone.

Performance Metrics and Counters (State-Space). KubeKlone exports performance metrics and counters at varying granularity to support a broad range of AIOps management tasks that may require observations from different perspectives. In particular, from the infrastructure perspective KubeKlone exposes resource usage and localized performance metrics. Whereas from the client perspective KubeKlone exposes end-to-end performance metrics:

- **Infrastructure metrics:** KubeKlone exports metrics for individual pods (an instance of service), which can be aggregated in various manners to provide server-level metrics (by aggregating across pods on a server), service level (by aggregating across pods in a service), and cluster level (by aggregating across servers in a cluster). The metrics range from resource utilization to queuing time and local latencies.
- **ClientSide-level metrics:** KubeKlone captures client metrics which provide end-to-end performance and success information.

Control Parameters (Action-Space). KubeKlone exposes both network-level knobs (e.g., rate-limit, retries, load balancing, timeouts, etc.) for achieving global optimizations (e.g., traffic engineering, service graph resiliency, etc.) and host-level knobs (e.g., CPU frequency, threads, core, etc.) for enabling local optimizations (e.g., optimizing the resource consumption of each microservice).

4 PROTOTYPE

In this section, we highlight KubeKlone's key implementation details.

Simulator. The simulator is written in 6371 Lines of C++ code. The simulator accepts user provided initial configurations for (a) the service graph of the application and (b) client workload.

ML Playground. The ML playground is written in Python in 600+ LoC. Below we elaborate on the two control loops supported by KubeKlone. To implement the General-ML control loop, we developed an agent class, which initializes the data structures and processes them to communicate with the core simulator. In this control pipeline, we provide an interface that has to be provided with a list of updated knobs to advance both the module and hence the simulator to the next round. To ease the deployment of RL-based control loops, we integrated the Tensorforce [28] library to extend our general-ML pipeline. To use any RL-based algorithm, the developer must set up a custom environment consisting of states, actions, and an execute function, which executes the actions and returns the next state, the reward, and a flag terminal indicating that the round has ended or not.

Developer Workflow: To use KubeKlone, the developer needs to capture a microservice model by running the microservice in a cluster and capturing the request processing times (distribution of processing times) for the different services. The processing times and the service graph (service communication patterns) are provided to KubeKlone as configuration inputs. Additionally, the developer will need to specify the client request distributions. The developer instantiates one of the interfaces and writes a new model. The developer can run KubeKlone to create a model. A developer can evaluate this model by testing the trained model on a different service graph, different distribution of processing times, or different distribution of client requests.

5 EXPERIMENTS

Our evaluation of KubeKlone is motivated by the following questions: (i) How precise is KubeKlone at reproducing the tail latency distribution of complex production grade microservices? (ii) How accurate is KubeKlone at simulating the behavior of popular ML-based systems?

Setup. To evaluate KubeKlone, we deployed Google's 10-tier Online-Boutique [37] with Istio on Google Compute Engine (GCE). The cluster includes three compute instances, each with eight vCPUs, 32 GB memory, 2.3 GHz, and Ubuntu 18.04.5 LTS. To simulate client workloads, we use wrk2 [55] to generate load with constant throughput and accurate latency details to 99.9999th percentile.

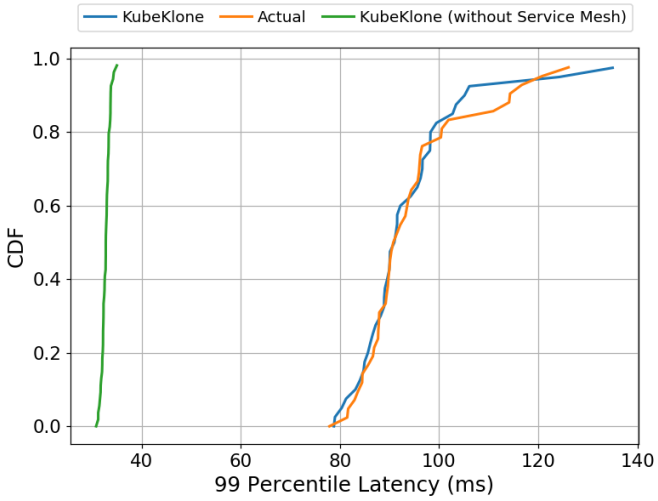

Figure 3: CDF of 99th Percentile Latency for Multiple Runs of the experiment in the (i) Actual Cluster, (ii) KubeKlone, (iii) KubeKlone (without Service Mesh).

Unless otherwise stated, we simulate a workload consisting of 50 product requests per second across ten connections in our experiments. The duration of each run is for 60 seconds. We performed 40 runs for the cluster experiments and KubeKlone simulations.

5.1 Validating Microservice Fidelity

We begin by demonstrating KubeKlone's ability to faithfully replicate a microservice and highlight the value of simulating the service mesh. To do this, we focus on analyzing the difference in performance between a service running in the cloud and the same service simulated with KubeKlone.

In Figure 3, we present the CDF of the 99th percentile tail latency across different runs. We observe that the values from KubeKlone closely align with the "actual" deployment. To quantify the similarities between these distributions, we use the Kolmogorov–Smirnov test – a popular statistical nonparametric test for quantifying the distance between two distributions. The Kolmogorov D statistic (i.e., absolute max distance between the CDFs of the two samples) is 0.092 with p-value (.98) greater than .05. The closer the D-value is to zero, the higher the samples are likely drawn from the same distribution and a higher p-value indicates that samples are identical.

Next, we focus on understanding the impact of ignoring the service mesh: Essentially, on the "KubeKlone (without service mesh)" curve, which captures the distribution of simulating the microservices without the service mesh. We observe significant degradation in the Kolmogorov D statistic from 0.092 to 1.0 – indicating that the samples are not drawn from the same distribution. We also observe a mean difference in latency of 57ms. This observation highlights the fact that simulating service mesh behavior is critical for microservices as the overheads introduced by the service mesh can be considerable.

Service Name	Mean Absolute Error (MAE)
FrontendService	0.03323525
AdService	0.04601590
CartService	0.07540984
CurrencyService	0.07478546
ProductService	0.01868797
RecommendationService	0.06093937

Table 2: Mean Absolute Error in CPU Utilization for different services.

5.2 Validating KubeKlone Fidelity

Now, we turn our attention to the end-to-end behavior of KubeKlone. In particular, we validate the behavior of running an ML algorithm in KubeKlone against running the algorithm on Google's cluster. For this analysis, we ported the PARTIES [7] resource management ML algorithm to KubeKlone using KubeKlone's general ML interface and train the model. All services are configured to request 900 millicores and 1 GB of memory at the start of the algorithm.

Table 2 summarizes the Mean Absolute Error between the CPU utilization simulated by KubeKlone against those recorded in the actual cluster deployment for each service. We observe that KubeKlone can faithfully reproduce a similar CPU Utilization rate with a mean error of .018 - .075 across different microservices.

6 USE CASE

KubeKlone is a digital twin for cloud-native infrastructures which enables developers to design ML algorithms for managing the performance of microservice deployments. This section highlights two famous microservices-based use cases and illustrates how KubeKlone can help design artificial intelligence for IT operations (AIOps) learning algorithms.

6.1 5G

5G is the fifth generation of mobile networks, and it offers multi-Gbps speed, ultra-low latency for emerging media technologies – AR, VR, etc. The success of many production 5G networks, e.g., Huawei [1], Ericsson [17], Nokia [20], heavily relies on redesigning key aspects of the 5G control plane [51]. In particular, these mobile operators have adopted microservices-based architecture because of their low total cost of ownership, scalability, high resiliency, etc.

One of the critical challenges faced in the microservices-based 5G architecture paradigm lies in providing ultra-low latency, which requires finding the optimal configurations that minimize the end-to-end latency.

KubeKlone plays a crucial role in enabling telecoms to effectively operate their 5G networks by providing them with both a simulator and the MLPlayground to explore and design effective learning algorithms for minimizing end-to-end latency. We plan to port emerging 5G microservices [25] to KubeKlone, and introduce representative 5G workloads. With these workloads and microservices, developers/researchers can use KubeKlone to explore different ML algorithms and understand their tradeoffs.

6.2 Edge Cloud

To support compute-intensive applications, organizations have started deploying clusters of servers closer to the clients, i.e., Edge Cloud. One of the popular methods for automating the decoupling and deployment of applications on edge clouds is to adopt a microservice-centric approach to application design [19]. For example, Walmart [42] and Chick-Fil-A [6] have introduced edge clouds at their stores to improve the performance and reliability of their POS (point of service) checkout systems.

Unlike the previous use case, reliability and resilience are key concerns [42]; this requires exploring the configuration of reliability-related parameters [32], e.g., retry, timeouts, and aborts.

Similarly, KubeKlone can reduce the barrier for designing algorithms to effectively auto-tune and improve reliability. Specifically, we plan to port emerging edge cloud microservices [26, 41] to KubeKlone, and introduce representative failures. With these failures and microservices, developers/researchers can use KubeKlone to explore ML algorithms for improving reliability and resilience.

7 RELATED WORK

A variety of simulators (e.g., SDN [29], data centre [31], wireless networks [58], underwater WSN [56], etc.) have been used over time to simulate complex systems and support systems research. The closest related work, uqSim [59] aims to simulate microservices; however, lacks the action-space to model a broad range of deployments and lacks support for training a broad range of learning models. Several other simulators [10, 23, 34, 47] have been also developed to simulate microservices chains but they lack a features for enabling experimenting on AIOPs systems for the emerging use cases described above. Both academic institutions [13, 49, 52] and industry [37, 48, 50] have open-sourced several microservice applications for benchmarking and testing; as part of future work, we plan to incorporate these benchmarks into KubeKlone.

8 CONCLUSION

As the community continues to develop and adopt Artificial Intelligence for IT Operations (AIOps) management techniques to tackle the complexity of microservices, we are in dire need of effective and practical approaches for building and training efficient AIOps models. KubeKlone represents a first step towards addressing this need. KubeKlone's design departs from existing simulations in several ways: first, in our decoupling of responsibilities; second, in our modeling of the service mesh; and finally, in our inclusion of both network and host contention and modeling times.

9 ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their helpful comments. This work was supported by NEC Labs.

REFERENCES

[1] 5G Network Architecture : A High-Level Perspective 2016. details retrieved from. https://carrier.huawei.com/~/media/CNBG/Downloads/Program/5g_nework_architecture_whitepaper_en.pdf. (2016). Accessed: 6-2-2021.

[2] Adopting Microservices at Netflix: Lessons for Architectural Design 2021. Details retrieved from. https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices. (2021). Accessed: 6-2-2021.

[3] AIOps 2021. Details retrieved from. https://www.ibm.com/cloud/learn/aiops. (2021). Accessed: 6-2-2021.

[4] AIOps (Artificial Intelligence for IT Operations) 2021. Details retrieved from. https://www.gartner.com/en/information-technology/glossary/aiops-artificial-intelligence-operations. (2021). Accessed: 6-2-2021.

[5] Application Stats Collection in Kubernetes via Telegraf Sidecars and Wavefront 2021. Details retrieved from. https://blogs.vmware.com/cloud/2018/08/08/application-stats-collection-kubernetes-via-telegraf-sidecars-wavefront/. (2021). Accessed: 6-2-2021.

[6] Bare Metal K8s Clustering at Chick-fil-A Scale 2020. details retrieved from. https://medium.com/@cfatechblog/bare-metal-k8s-clustering-at-chick-fil-a-scale-7b0607bd3541. (2020). Accessed: 6-2-2021.

[7] Shuang Chen, Christina Delimitrou, and José F. Martínez. 2019. PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 107–120. https://doi.org/10.1145/3297858.3304005

[8] Choosing the Right API Gateway Pattern for Effective API Delivery 2021. Details retrieved from. https://www.nginx.com/blog/choosing-the-right-api-gateway-pattern/. (2021). Accessed: 6-2-2021.

[9] Consul 2021. Details retrieved from. https://www.consul.io/. (2021). Accessed: 6-2-2021.

[10] Clément Courageux-Sudan, Anne-Cécile Orgerie, and Martin Quinson. 2021. Automated performance prediction of microservice applications using simulation. In *2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 1–8. https://doi.org/10.1109/MASCOTS53633.2021.9614260

[11] Elastic Search 2021. Details retrieved from. https://netflixtechblog.com/introducing-raigad-an-elasticsearch-sidecar-350c7e01339f. (2021). Accessed: 6-2-2021.

[12] D. G. Feitelson, E. Frachtenberg, and K. L. Beck. 2013. Development and Deployment at Facebook. *IEEE Internet Computing* 17, 4 (2013), 8–17. https://doi.org/10.1109/MIC.2013.25

[13] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud ; Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 3–18. https://doi.org/10.1145/3297858.3304013

[14] How Ancestry used AI to optimize its microservices apps 2021. Details retrieved from. hhttps://techbeacon.com/enterprise-it/how-ancestry-used-ai-optimize-its-microservices-apps. (2021). Accessed: 6-2-2021.

[15] How Ancestry used AI to optimize its microservices apps 2021. Details retrieved from. https://techbeacon.com/enterprise-it/how-ancestry-used-ai-optimize-its-microservices-apps. (2021). Accessed: 6-2-2021.

[16] How Deep Reinforcement Learning Makes Real Performance Tuning Possible 2021. Details retrieved from. https://opsani.com/blog/how-deep-reinforcement-learning-makes-real-performance-tuning-possible/. (2021). Accessed: 6-2-2021.

[17] How Ericsson is leveraging cloud native to enable 5G transformation ? 2021. details retrieved from. https://www.cncf.io/case-studies/ericsson/. (2021). Accessed: 6-2-2021.

[18] How Netflix Deploys Code 2021. Details retrieved from. https://www.infoq.com/news/2013/06/netflix/. (2021). Accessed: 6-2-2021.

[19] https://thenewstack.io/the-use-case-for-kubernetes-at-the-edge/ 2020. details retrieved from. TheUseCaseforKubernetesattheEdge. (2020). Accessed: 6-2-2021.

[20] Implementing microservices and containers in a cloud-native core 2021. details retrieved from. https://www.nokia.com/blog/implementing-microservices-and-containers-cloud-native-core/. (2021). Accessed: 6-2-2021.

[21] Istio 2021. Details retrieved from. https://istio.io/. (2021). Accessed: 6-2-2021.

[22] Java EE microservices: why start-up time and size matter 2021. Details retrieved from. https://www.linkedin.com/pulse/java-ee-microservices-why-start-up-time-size-matters-matjaz-b-juric/. (2021). Accessed: 6-2-2021.

[23] Michel Gokan Khan, Javid Taheri, Auday Al-Dulaimy, and Andreas Kassler. 2021. PerfSim: A Performance Simulator for Cloud Native Computing. *CoRR* abs/2103.08983 (2021). arXiv:2103.08983 https://arxiv.org/abs/2103.08983

[24] Kong Gateway 2021. Details retrieved from. https://konghq.com/kong/. (2021). Accessed: 6-2-2021.

[25] Kube5G 2021. Details retrieved from. https://mosaic5g.io/kube5g/. (2021). Accessed: 6-2-2021.

[26] Kubedge Sample Applications 2019. details retrieved from. https://github.com/kubeedge/examples. (2019). Accessed: 6-2-2021.

[27] Kubernetes Log Management using Fluentd as a Sidecar Container and preStop Lifecycle Hook- Part IV 2021. Details retrieved from. https://tinyurl.com/2n8m57w4. (2021). Accessed: 6-2-2021.

[28] Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. 2017. Tensorforce: a TensorFlow library for applied reinforcement learning. Web page. (2017). https://github.com/tensorforce/tensorforce

[29] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*. Association for Computing Machinery, New York, NY, USA, Article 19, 6 pages. https://doi.org/10.1145/1868447.1868466

[30] Philipp Leitner, Jürgen Cito, and Emanuel Stöckli. 2016. Modelling and Managing Deployment Costs of Microservice-Based Cloud Applications. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*. 165–174.

[31] David Meisner, Junjie Wu, and Thomas F. Wenisch. 2012. BigHouse: A simulation infrastructure for data center systems. In *2012 IEEE International Symposium on Performance Analysis of Systems Software*. 35–45. https://doi.org/10.1109/ISPASS.2012.6189204

[32] N. C. Mendonca, C. M. Aderaldo, J. Camara, and D. Garlan. 2020. Model-Based Analysis of Microservice Resiliency Patterns. In *2020 IEEE International Conference on Software Architecture (ICSA)*. 114–124. https://doi.org/10.1109/ICSA47634.2020.00019

[33] Microservice in Action 2021. Details retrieved from. https://livebook.manning.com/book/microservices-in-action/chapter-8/129. (2021). Accessed: 6-2-2021.

[34] microsim (Microservice Simulator) 2021. Details retrieved from. https://kellyjonbrazil.github.io/microsim/. (2021). Accessed: 6-2-2021.

[35] Netflix Builds a Pipeline for Polyglot Programming 2021. Details retrieved from. https://thenewstack.io/netflix-builds-pipeline-polyglot-programming/. (2021). Accessed: 6-2-2021.

[36] Netreo Releases AIOps: Autopilot to Automatically Identify and Repair Deviations in IT Infrastructure 2021. Details retrieved from. https://tinyurl.com/yckp3t6e. (2021). Accessed: 6-2-2021.

[37] Online Boutique 2021. details retrieved from. https://github.com/GoogleCloudPlatform/microservices-demo. (2021). Accessed: 6-2-2021.

[38] Menuka Perera Jayasuriya Kuranage, Kandaraj Piamrat, and Salima Hamma. 2020. Network Traffic Classification Using Machine Learning for Software Defined Networks. In *Machine Learning for Networking*, Selma Boumerdassi, Éric Renault, and Paul Mühlethaler (Eds.). Springer International Publishing, Cham, 28–39.

[39] Retries and Timeouts 2021. Details retrieved from. https://linkerd.io/2/features/retries-and-timeouts/. (2021). Accessed: 6-2-2021.

[40] Saim Salman, Christopher Streiffer, Huan Chen, Theophilus Benson, and Asim Kadav. 2018. DeepConf: Automating Data Center Network Topologies Management with Machine Learning. In *Proceedings of the 2018 Workshop on Network Meets AI (NetAI'18)*. Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/3229543.3229554

[41] SAP IOT edge Samples 2019. details retrieved from. https://github.com/SAP-samples/iot-edge-samples. (2019). Accessed: 6-2-2021.

[42] Seamless Customer Experience powered by Kubernetes Edge at Walmart Stores 2018. details retrieved from. https://tinyurl.com/3khmhw5w. (2018). Accessed: 6-2-2021.

[43] Self-Driving Operations 2021. Details retrieved from. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/products/vrealize/vmware-self-driving-vrealize-operations-final.pdf. (2021). Accessed: 6-2-2021.

[44] Sensu 2021. Details retrieved from. https://sensu.io/resources/whitepaper/whitepaper-monitoring-kubernetes-the-sidecar-pattern. (2021). Accessed: 6-2-2021.

[45] Service Architectures at Scale: Lessons from Google and eBay 2021. Details retrieved from. https://www.infoq.com/presentations/service-arch-scale-google-ebay/. (2021). Accessed: 6-2-2021.

[46] Service Architectures at Scale: Lessons from Google and eBay 2021. Details retrieved from. https://www.leanix.net/en/blog/why-netflix-amazon-and-apple-care-about-microservices. (2021). Accessed: 6-2-2021.

[47] Simulating APIs for Effective Testing: (Micro)Service Virtualisation 2021. Details retrieved from. https://tinyurl.com/33kmmav3. (2021). Accessed: 6-2-2021.

[48] SockShop 2021. Details retrieved from. https://microservices-demo.github.io. (2021). Accessed: 6-2-2021.

[49] Akshitha Sriraman and Thomas F. Wenisch. 2018. µSuite: A Benchmark Suite for Microservices. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. 1–12. https://doi.org/10.1109/IISWC.2018.8573515

[50] Stan's Robot Shop – a Sample Microservice Application 2021. Details retrieved from. https://www.instana.com/blog/stans-robot-shop-sample-microservice-application/. (2021). Accessed: 6-2-2021.

[51] System Architecture for the 5G System (3GPP TS 23.501 version 15.2.0 Release 15) 2021. details retrieved from. https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/15.02.00_60/ts_123501v150200p.pdf. (2021). Accessed: 6-2-2021.

[52] Jóakim von Kistowski, Simon Eismann, Norbert Schmitt, André Bauer, Johannes Grohmann, and Samuel Kounev. 2018. TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 223–236. https://doi.org/10.1109/MASCOTS.2018.00030

[53] What is AIOps? | AppDynamics 2021. Details retrieved from. https://www.appdynamics.com/topics/what-is-ai-ops. (2021). Accessed: 6-2-2021.

[54] What Led Amazon to its Own Microservices Architecture 2021. Details retrieved from. https://thenewstack.io/led-amazon-microservices-architecture/. (2021). Accessed: 6-2-2021.

[55] wrk2 2021. details retrieved from. https://github.com/giltene/wrk2. (2021). Accessed: 6-2-2021.

[56] Peng Xie, Zhong Zhou, Zheng Peng, Hai Yan, Tiansi Hu, Jun-Hong Cui, Zhijie Shi, Yunsi Fei, and Shengli Zhou. 2009. Aqua-Sim: An NS-2 based simulator for underwater sensor networks. In *OCEANS 2009*. 1–7. https://doi.org/10.23919/OCEANS.2009.5422081

[57] Noe Marcelo Yungaicela-Naula, Cesar Vargas-Rosales, and Jesus Arturo Perez-Diaz. 2021. SDN-Based Architecture for Transport and Application Layer DDoS Attack Detection by Using Machine and Deep Learning. *IEEE Access* 9 (2021), 108495–108512. https://doi.org/10.1109/ACCESS.2021.3101650

[58] X. Zeng, R. Bagrodia, and M. Gerla. 1998. GloMoSim: a library for parallel simulation of large-scale wireless networks. In *Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS '98 (Cat. No.98TB100233)*. 154–161. https://doi.org/10.1109/PADS.1998.685281

[59] Yanqi Zhang, Yu Gan, and Christina Delimitrou. 2019. uqSim: Scalable and Validated Simulation of Cloud Microservices. (2019). arXiv:cs.DC/1911.02122