Oakestra: An Orchestration Framework for Edge Computing

Giovanni Bartolomeo  Simon Bäurle  Nitinder Mohan  Jörg Ott
Technical University of Munich

ABSTRACT

Edge computing enables developers to deploy their services on compute resources deployed closer to the users. The abstraction requires powerful orchestration capabilities and the resolution of complex optimization problems. While edge computing is a consistently growing trend, the community (research and industry) still largely embraces adaptations and extensions of existing cloud technologies that have been proven ineffective on edge (e.g. Kubernetes). In this work, we present Oakestra, a novel hierarchical orchestration framework specifically designed for supporting service operation over heterogeneous edge infrastructures. In this demonstration, we showcase the various features and operations of Oakestra using our latency-critical augmented reality (AR) application.

CCS CONCEPTS

• Computer systems organization → Distributed architectures; Heterogeneous (hybrid) systems; • Software and its engineering → Development frameworks and environments.

KEYWORDS

Edge Computing, Orchestration Framework, Resource Management

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGCOMM '22 Demos and Posters, August 22–26, 2022, Amsterdam, Netherlands
© 2022 Copyright held by the owner/author(s).
https://doi.org/10.1145/3546037.3546056

1 INTRODUCTION

Within a decade since inception, edge computing has found a wide range of industrial and research use-cases [14, 23, 24]. In order to cope with the strict QoS requirements of latency-critical applications [11, 22], orchestration frameworks for edge computing must support efficient service management along with cloud offloading capabilities [1, 26]. Heterogeneity of the resources and fluctuations in the network makes orchestration particularly challenging [9, 10, 13, 25]. Popular approaches for orchestrating edge infrastructures adapt existing cloud-native technologies [27]. For example, solutions like Kubernetes (k8s) [6], even the lightweight distributions like k3s [8], KubeEdge [7], and mikroK8s [5] struggle at the edge due to their strong consistency assumptions [4, 20]. Furthermore, edge adaptations of k8s [17, 18, 20] carry-forward the burden of k8s footprint on constrained hardware resources. Other research orchestration approaches at the edge like [2, 15, 16, 19, 21, 28] also fail to (a) integrate cloud and edge workloads with minimal efforts, (b) manage geographically distributed heterogeneous hardware clusters, and (c) interconnect coordinated microservices considering diverse processing and network constraints. As a solution, with our work we bring the following contributions:

1) We propose Oakestra, a novel hierarchical orchestration framework that supports federated infrastructures for application deployment, management, and computation offloading. Different clusters of heterogeneous resources over multiple geographical locations can be combined to build a federated cloud-to-edge continuum. We implement Oakestra as lightweight and scalable framework supporting compute-constrained edge devices in combination with powerful cloud resources with minimal overhead.

2) Application providers can deploy services at the edge by specifying high-level SLA, such as hardware, latency, bandwidth, etc. as constraints within deployment descriptors. Oakestra's hierarchical management structure, in conjunction with the delegated service scheduling principle, allows it to decentralize task placement and find effective deployment much faster than the state-of-the-art.

3) Using semantic IP addresses we natively support flexible and transparent load balancing techniques. This way, we provide support for service mobility and hardware constraints while enabling easy portability of applications as no code adaptation is required. Moreover, the platform provides an overlay network for service-to-service communication to allow the traversal of NATs and firewalls.

2 OAKESTRA: A PRIMER

Oakestra is a lightweight orchestration framework natively designed to support the many constraints of edge environments. The key innovation lies in the two-tier logical hierarchical orchestration of compute resources which, unlike flat architectures of k8s-inspired frameworks, decomposes the control plane management...
into many clusters. As shown in fig. 1, each cluster is managed by its local \textit{cluster orchestrator} which is only responsible for resources within its cluster. The cluster orchestrator coordinates with the parent \textit{root orchestrator} and sends it aggregated resource utilization of the cluster (such as CPU/GPU cores, memory, and storage) and current service operation statistics (e.g., resources consumed, location, latency, etc.). Application developers only interact with the root orchestrator for deploying their services at the edge – submitting the \textit{deployment descriptor} containing the SLA requirements along with the service code via API/CLI/Web UI. The SLA may include latency thresholds or geographical area constraints, resource requirements (bandwidth in/out, vCPU/vGPU/vTPU cores, memory, and disk size), and convergence time for service scheduling operations. Thanks to its hierarchical structure and \textit{delegated scheduling mechanism}, \textit{Oakestra} can quickly resolve service scheduling requests at the edge, which is a well-known NP-hard problem [12]. The root orchestrator first calculates the best-fit clusters for every service request by broadly mapping the requirements to aggregated statistics it received from the cluster orchestrators. Further, it offloads the service request to the cluster scheduler, which finds the optimal deployment of the resources within the cluster. Since the problem space is greatly reduced, frequent service (re)scheduling is no longer costly using \textit{Oakestra}.

We also design and implement \textit{Oakestra} such that multiple edge (or cloud) operators can contribute their resources (as different clusters) to shared infrastructure and retain administrative control. For instance, we allow each cluster operator to incorporate different scheduling policies within their clusters, such as latency, fairness, etc. To support service interactions over resources across different clusters (possibly behind NATs/firewalls), \textit{Net Manager} in the worker utilizes a novel \textit{semantic addressing} scheme which can dynamically (and transparently) adjust communication endpoints in response to infrastructure changes, e.g., service migrations, resource failures, etc., ensuring uninterrupted service interactions. Moreover, \textit{Oakestra} is extremely lightweight which allows it to operate effectively over highly compute-constrained resources with minimal overhead. Our experiments over real edge infrastructures reveal that \textit{Oakestra} outperforms state-of-the-art solutions like k8s and k3s. In comparison, we achieved a $\approx 10x$ reduction on computing resources footprint and 10% application performance improvement measured in terms of max FPS achieved by the same AR pipeline (not shown due to space restrictions).

3 DEMONSTRATION

This demonstration will showcase the capabilities of \textit{Oakestra} in an edge-cloud infrastructure using a latency-critical AR application.

\textbf{AR Application.} The AR pipeline (see fig. 2) is composed of three networked microservices [3]. \textit{Pre} is the \textit{pre processing} service that takes live camera stream and scales down the image as per the model specification of rest of the pipeline. \textit{Obj} performs \textit{object detection} on each frame and outputs the bounding boxes coordinates. \textit{Rec} performs \textit{object recognition} on the bounding boxes found by \textit{Obj} and labels them. It sends the output to the display which scales up the output to original resolution. Each component of the pipeline can be replicated in multiple instances and is GPU accelerated.

\textbf{Oakestra Setup.} We will create a two cluster infrastructure deployment managed by \textit{Oakestra} as shown in fig. 3. The edge cluster will be composed of heterogeneous resources (Raspberry Pi’s and Jetsons) on the demo table while the cloud cluster will have VMs in a cloud datacenter. The \textit{root orchestrator} will also be in the cloud, albeit from a different cloud operator than cloud cluster to highlight the multi-cloud operation. We will demonstrate (i) how to deploy applications at the edge using \textit{Oakestra}’s APIs, (ii) how \textit{Oakestra} handles sudden spikes in application load via reschedulings/replications, and (iii) how \textit{Oakestra} transparently handles resource and service failures at the edge. Simultaneously, audience can observe the live performance of the application and the operational load of the infrastructure.

\textbf{Demonstration Design.} In step ①, we will deploy the AR application using \textit{Oakestra}’s web interface which will schedule its microservices on edge resources. Additionally, we will deploy another instance of \textit{Pre} in the cloud such that both instances are independently operational in the infrastructure. In step ② we will scale the number of clients, thereby increasing the load on the pipeline. As a consequence, \textit{Oakestra} will scale-up the bottleneck \textit{Obj} instances in real-time – which will be evident from latency reduction in the pipeline output. In step ③ we will demonstrate resilience of \textit{Oakestra} by injecting node failures in edge cluster – abruptly killing the \textit{Pre} service, which will seamlessly re-route client traffic to the instance in cloud cluster. In meantime, \textit{Oakestra} will re-deploy the failed \textit{Pre} instance on an unused resource in the edge cluster and will resume service operation at the edge. Finally, in step ④, we will kill the node hosting the only \textit{Rec} service instance which requires GPU for operation – resulting in loss of labels in pipeline output. As a result, \textit{Oakestra} will reschedule the failed instance of \textit{Rec} to the only available GPU node in edge cluster, restoring optimal pipeline operation with minimal downtime.

\textbf{Acknowledgements.} This work is supported by EU Celtic project Piccolo (C2019/2-2), Federal Ministry of Education and Research of Germany (BMBF) project 6G-Life (16KISK002) and Bavarian Ministry of Economic Affair project 6G Future Lab Bavaria.