

# Towards Seamless Serverless Computing Across an Edge-Cloud Continuum

Emilian Simion  
Institute of Informatics, University of  
Amsterdam  
Faculty of Science, Vrije Universiteit  
Amsterdam  
Amsterdam, The Netherlands  
emilian.simion@student.uva.nl

Yuandou Wang  
Multiscale Networked Systems,  
University of Amsterdam  
Amsterdam, The Netherlands  
y.wang8@uva.nl

Hsiang-ling Tai  
Informatics Institute, University of  
Amsterdam  
Faculty of Science, Vrije Universiteit  
Amsterdam  
Amsterdam, The Netherlands  
hsiang-ling.tai@student.uva.nl

Uraz Odyurt  
High-Energy Physics, Radboud  
University  
Nijmegen, The Netherlands  
Nikhef  
Amsterdam, The Netherlands  
uodyurt@nikhef.nl

Zhiming Zhao  
Multiscale Networked Systems,  
University of Amsterdam  
LifeWatch ERIC Virtual Lab and  
Innovation Center (VLIC),  
Amsterdam, The Netherlands  
Amsterdam, The Netherlands  
z.zhao@uva.nl

## ABSTRACT

Serverless computing has emerged as an attractive paradigm due to the efficiency of development and the ease of deployment without managing any underlying infrastructure. Nevertheless, serverless computing approaches face numerous challenges to unlock their full potential in hybrid environments. To gain a deeper understanding and firsthand knowledge of serverless computing in edge-cloud deployments, we review the current state of open-source serverless platforms and compare them based on predefined requirements. We then design and implement a serverless computing platform with a novel edge orchestration technique that seamlessly deploys serverless functions across the edge and cloud environments on top of the Knative serverless platform. Moreover, we propose an offloading strategy for edge environments and four different functions for experimentation and showcase the performance benefits of our solution. Our results demonstrate that such an approach can efficiently utilize both cloud and edge resources by dynamically offloading functions from the edge to the cloud during high activity, while reducing the overall application latency and increasing request throughput compared to an edge-only deployment.

## KEYWORDS

Serverless Computing, Edge-Cloud Continuum, Knative, Offloading

## 1 INTRODUCTION

Serverless computing, as a new and greatly popular paradigm with the cloud computing community, simplifies software development and deployment in a cloud environment. Serverless enables developers to run their code without having to manage any underlying infrastructure [4, 5]. One typical offering is Functions-as-a-Services (FaaS), exemplified by public Cloud services such as AWS Lambda [24], Azure Functions [13], and Google Cloud Functions [16]. FaaS offers event-driven execution of functions for cloud

customers to run their code for virtually any type of application or backend service without provisioning or managing any servers.

Apart from serverless computing, the emerging applications in various areas such as manufacturing [6, 21], healthcare [1, 20], smart cities [12, 15], agriculture and farming [10, 19] and transportation [14, 27], have been both nurturing and demanding edge computing in recent years. Edge computing brings processing, data storage, and applications closer to the edge of the network, where end devices such as Internet-of-Things (IoT) devices and smartphones generate and consume data, which benefits from low latency, improved reliability, better data privacy, cost savings, and energy efficiency [2, 7, 23]. At the same time, it is more common to have applications composed of different modules distributed over different tiers (e.g., edge, fog, cloud) and interoperate between themselves [17]. Such computational model has emerged under the term Edge-Cloud continuum, in which infrastructure’s geo-distributed and heterogeneous nature presents unique challenges and opportunities [3, 22]. However, interoperating control across an Edge-Cloud continuum is still a challenge. Several existing works have made much progress in tackling this challenge. Nevertheless, resource management, scheduling, fault tolerance, deployment complexity, and cold-start mitigation [8, 9, 18, 25, 26] still need to be addressed.

In this paper, we address the interoperability problem between edge and cloud environments through serverless computing, making the deployment process more efficient across the Edge-Cloud continuum and improving the performance of processing applications. Our vision for interoperable serverless computing, which enables an Edge-Cloud continuum, involves a platform that abstracts not just the infrastructure, but also the location where functions are executed, using available resources as efficiently as possible. We aim to gain practical insights into the intricacies and challenges associated with this architecture. Our primary contribution can be summarized as follows:

- We review the current state of serverless platforms and compare them based on predefined requirements.

- We implement a serverless platform and develop a novel edge orchestration technique that enables a seamless deployment of serverless functions across both edge and cloud environments.
- We propose an edge offloading strategy and conduct extensive experiments to showcase its performance benefits. The source code is available at the GitHub repository<sup>1</sup>.

The remainder of this paper is structured as follows. Section 2 presents related work and in Section 3, we pose the requirement analysis, platform choices and our framework. Section 4 details the experimental setup and discusses the results. Finally, we conclude our work in Section 5.

## 2 RELATED WORK

Recently, research has addressed serverless applications in the edge-cloud continuum, focusing on three dimensions: (1) scheduling functions in resource-limited edge environments [25, 26], (2) optimizing serverless resource usage in edge environments [9, 11], and (3) deployment complexity for seamless integration of serverless computing across this continuum [8, 18]. Wang *et al.* [26] introduced Lass, a platform for running latency-sensitive serverless apps on the edge using queuing theory to allocate resources and auto-scale as needed. Tang *et al.* [25] proposed a deep learning task scheduling algorithm for resource utilization improvement at the edge. By contrast, we extend Knative’s default round-robin scheduling to enable offloading requests from edge to cloud based on function response times. Gadepalli *et al.* [9] explored WebAssembly’s potential for efficient serverless computing at the edge due to its low resource overhead. Jeon *et al.* [11] optimized resource usage by caching function dependencies using deep reinforcement learning. Our approach focuses on offloading work to the cloud, not optimizing runtime overhead. Nastic *et al.* [18] presented Serverless Computing Fabric (SCF) for the Edge-Cloud continuum, addressing edge-native backend services, resource usage, and edge intelligence. Ferry *et al.* [8] introduced a solution for Cloud-Edge-IoT applications with a modeling language. Our approach does not target IoT scenarios specifically; however, it may have higher overhead in some IoT contexts, as Knative’s features have been designed with a focus on cloud environments.

## 3 METHODOLOGY

### 3.1 Requirement Analysis

The design of a serverless platform running on the Edge-Cloud continuum necessitates careful consideration of multiple factors, including scalability, security, cost-effectiveness, and performance. On the one hand, this method must be capable of operating at both the Edge, close to the source of data, as well as in the Cloud, where it can leverage the resources of a large data center. On the other hand, the design should address the unique challenges presented by this hybrid environment, such as managing data transfer between the Edge and Cloud and handling variable traffic levels in real-time. By harnessing the advantages of both the Edge and the Cloud, our serverless platform can provide a flexible and efficient solution for a

wide range of applications. To achieve this, we define the following functional and non-functional requirements.

The **functional requirements** we propose for our solution are the following:

- **Location Agnostic.** The same function definition can be utilized to run serverless functions in both Edge and Cloud environments.
- **Scalability.** The platform is capable of accommodating the addition of new edge clusters and nodes dynamically.
- **Dynamic Scheduling.** The edge cluster gateway must have the ability to dynamically determine the execution location of a function, either at the Edge or in the Cloud.

We also define the **non-functional requirements** as follows:

- **Heterogeneity.** The system should support nodes with varying degrees of hardware capabilities (e.g. x86, ARM).
- **Resource Allocation.** The system must possess the capability to dynamically allocate resources in accordance with workload demands.
- **Fault Tolerance.** If a worker node experiences a failure, the system can still operate normally.
- **Reliability.** The system should be robust to bad Edge-Cloud connection.
- **Security.** Communication between Edge and Cloud should be secure.

We establish a set of **selection criteria** for platform comparison that will guide the design process.

- (1) The serverless platform should be actively maintained in the community.
- (2) The serverless platform should be open-source and have a permissive license. We wish to be able to extend the platform while avoiding any restrictive licensing.
- (3) The serverless platform must possess the capability to scale-to-zero and scale workloads according to actual demand, which are essential features for its intended purpose.
- (4) The serverless platform is able to run on limited resources and a varying degree of heterogeneous hardware.

These criteria serve as the foundation for our solution, ensuring that the final solution aligns with the desired goals and objectives.

### 3.2 Technology Investigation

**Table 1:** For each investigated serverless platform, we have assessed whether they fit our platform criteria #(1-4).

| Serverless Platform | Platform criteria |      |      |      |
|---------------------|-------------------|------|------|------|
|                     | #(1)              | #(2) | #(3) | #(4) |
| Kyma                | ✓                 | ✓    | ✓    | ✓    |
| Knative             | ✓                 | ✓    | ✓    | ✓    |
| OpenFaaS            | ✓                 |      | ✓    | ✓    |
| Fission             | ✓                 | ✓    | ✓    | ✓    |
| OpenLambda          |                   | ✓    | ✓    | ✓    |
| OpenWhisk           | ✓                 | ✓    | ✓    | ✓    |
| Kubeless            |                   | ✓    | ✓    | ✓    |
| Fn                  |                   | ✓    | ✓    | ✓    |
| IronFunctions       |                   | ✓    | ✓    | ✓    |

<sup>1</sup>Knative Edge. <https://github.com/jevvk/knative-edge>

We collect information from literature and official sources, such as GitHub statistics incl., stars, forks, issues, number of commits, and software documentation, and conclude nine mainstream open-source serverless platforms. We have checked the capabilities of each platform as understood from the official documentation and initial deployments of the platforms on our experimental setup and evaluated the reasons for including or excluding them from our platform options. This evaluation process helps us to determine which serverless platforms are best suited to meet our platform’s functional and non-functional requirements and to make informed decisions about which platform to choose for our implementation. We summarize our findings in Table 1 and explain our reasoning below for each serverless platform on a case-by-case basis.

- **Knative** is a well-regarded serverless platform that has gained popularity in academic research circles and within the open-source community. Our requirements analysis has determined that it is a suitable choice to include in the platform options.
- **Kyma**. As a FaaS solution based on Knative, Kyma inherits this underlying framework’s technological capabilities and constraints. It meets all the previously defined requirements.
- **OpenFaaS**. Despite its technical capabilities, the licensing of core serverless features in OpenFaaS prevents us from using it; it requires a paid license for crucial features such as scaling to zero and event handling using message brokers.
- **Fission**. Our analysis has indicated that it has the capabilities and features to fulfill the intended purpose of the platform, as well as meet the non-functional requirements such as performance, scalability, security, and cost-effectiveness.
- **OpenLambda**. Despite being designed for academic research, OpenLambda lacks more support for cluster deployments, which is a critical requirement for deploying our solution.
- **OpenWhisk** is a well-established serverless platform. Our requirements analysis indicates that it satisfies all the previously established functional and non-functional requirements.
- **Kubeless**. Despite being widely recognized for its performance, we exclude Kubeless from the design of the serverless platform because it is no longer actively maintained.
- **Fn**. While Fn was one of the pioneers in the open-source serverless space, it is no longer maintained and will not be considered in our analysis.
- **IronFunctions**. Like Fn, IronFunctions is an early example of an open-source serverless solution. Unfortunately, like Fn and Kubeless, it has become inactive and, as a result, has been excluded from our design’s list of platform options.

During our practical assessment using our test bed, we encountered various deployment challenges for different serverless platforms. Notably, we faced memory limitations that prevented us from deploying OpenWhisk due to requirements associated with the necessary message broker. While Fission was successfully deployed (as shown in Table 1), it exhibited suboptimal performance on our test bed, with function instances frequently hanging and necessitating frequent restarts. In contrast, both Knative and Kyma were successfully deployed and thoroughly tested. After careful

consideration, we selected Knative as the foundation for our solution. This choice was supported by the fact that Kyma is built on top of Knative, ensuring compatibility between our solution and Kyma.

### 3.3 Platform Design and Implementation

We design and implement Knative Edge as an extension of the existing Knative serverless platform as shown in Figure 1(A).

**3.3.1 Cloud-to-Edge Replication.** The Knative Edge controller mirrors Knative Services from the Cloud cluster to the Edge cluster by watching for changes in the Kubernetes resources of both clusters and modifying the Edge resource to keep a consistent definition. As shown in the replicate of Figure 1(B), it uses several different components within the edge cluster to achieve its goals. However, replicating a resource may risk making unnecessary changes to the resources it replicates. It is most evident when a Knative Service is replicated, as any changes can trigger Knative Serving to react and make more changes to the resource, which can trigger Knative Edge to make more changes. This feedback loop can cause degradation of the serverless functions running either on the Edge or Cloud and can increase the traffic between the Cloud and Edge.

Our approach to this issue is to selectively compare fields in the Knative Service definition. Whenever Knative Edge receives an update for services it replicates, it copies the current definition from the Edge and overwrites a subset of its fields using the definition from the Cloud. When overwriting, we skip over the state of the service and any annotations which are not defined by Knative Edge; thus, the internal state of the Edge resources are persisted. The new service definition is compared to the current definition on the Edge cluster and deployed to the cluster if any change is detected.

**3.3.2 Edge-to-Cloud Offloading.** For efficiently monitoring and managing runtime platform metrics in our Edge cluster, we utilize an instance of Prometheus deployed on each Edge cluster, as shown in Figure 1 (C). This instance is configured to be aware of all Knative components in the cluster, including all running function instances. It scrapes these metrics regularly and temporarily stores them in a time series database, which is optimized for storing and querying metrics such as those generated by Knative. Since only recent data is being queried by Knative Edge, we configure short data liveness to reduce as much as possible the overhead of Prometheus. The scheduling is driven by a load-balancing algorithm that spreads the traffic to the different routes based on a defined percentage. We implement a simple default offloading strategy that uses the request latency metrics of all the functions running at the Edge. The API Gateway makes the decision randomly, and only a percentage of traffic (decided by the offloading strategy) is being sent to the cloud.

Let  $X_l(t)$  be the distribution of request latencies at time  $t$  and  $p_{95}$  and  $p_{50}$  are the 95<sup>th</sup> and 50<sup>th</sup> percentile, respectively. The weighted sum of the latest latency response ratio  $r_l(t)$  can be given by,

$$r_l(t) = \frac{p_{95}(X_l(t))}{p_{50}(X_l(t))} \quad (1)$$

Giving more importance to recent values than to older ones, we use an exponentially decreasing weighted sum with  $c_{\text{decay}}$  as exponent

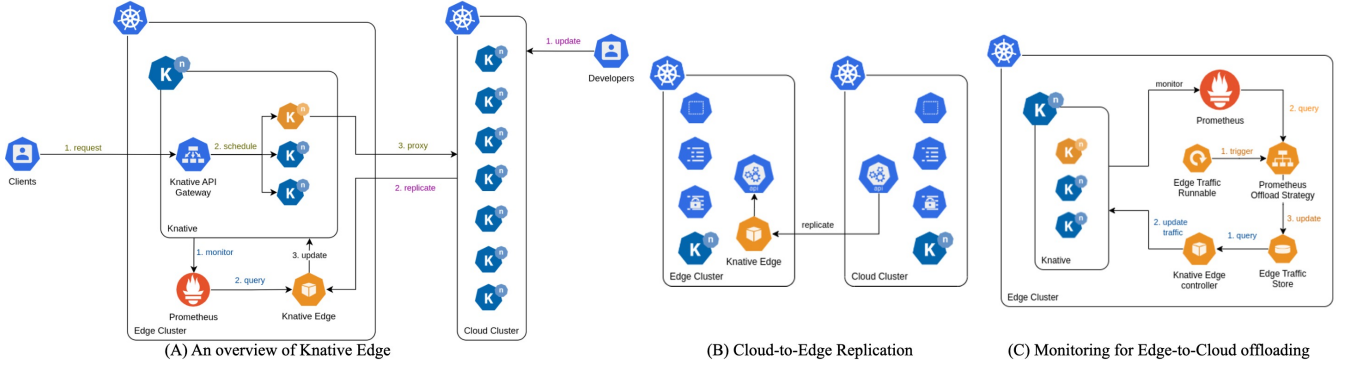


Figure 1: An overview of the System Architecture and core components.

to implement  $r'_l(t)$ , which is given by,

$$r'_l(t) = \frac{\sum_{k=0}^{c_t} c_{\text{decay}}^k \times r_l(t-k)}{\sum_{k=0}^{c_t} c_d^k} \quad (2)$$

where  $c_t$  defines the how many steps in time are used to calculate the weighted sum of the latest request latency ratios measured. It calculates  $r_t$  in Equation (3), the intended traffic percentage that should be forwarded to the Cloud. Through different iterations, we found that directly using  $r_t$  for setting the traffic percentage lead to unstable offloading. We define  $r_t(t)$  as

$$r_t(t) = \begin{cases} 0, & \text{if } r'_l(t) < c_{\text{soft}}, \\ 100, & \text{if } r'_l(t) > c_{\text{hard}}, \\ 100 \times \frac{r'_l(t) - c_{\text{soft}}}{c_{\text{hard}} - c_{\text{soft}}}, & \text{otherwise.} \end{cases} \quad (3)$$

in which  $c_{\text{soft}}$  and  $c_{\text{hard}}$  are the soft limit and hard limit of the percentile ratio. For any  $r'_l$  that falls bellow  $c_{\text{soft}}$ , the traffic percentage is set to 0. For values above  $c_{\text{hard}}$ , the traffic percentage is set to 100. And finally, for values between, we interpolate them between 0 and 100, depending on where the values lie in respect to  $c_{\text{soft}}$  and  $c_{\text{hard}}$ . Hence, we define  $R_t$  in Equation (4) as a more stable way of updating the traffic percentage.

$$R_t(t) = R_t(t-1) \times c_{\text{in}} + r_t(t) \times (1 - c_{\text{in}}), R_t(0) = 0. \quad (4)$$

where  $c_{\text{in}}$  is the inertia factor, which measures how much  $r_t$  influences  $R_t$ .

## 4 PLATFORM EVALUATION

Table 2: For each workload type and traffic split, we present the total number of successful responses.

| Traffic | MatMult | Image Proc. | I/O  | Mixed |
|---------|---------|-------------|------|-------|
| 0%      | 2406    | 3627        | 4852 | 4152  |
| 25%     | 2699    | 4044        | 5947 | 5237  |
| 50%     | 2664    | 4045        | 9371 | 7486  |
| 75%     | 2683    | 3970        | 9371 | 8619  |
| 100%    | 2668    | 3969        | 9408 | 8725  |
| auto    | 2700    | 4016        | 6548 | 7989  |

### 4.1 Experimental Setup

We utilized four Raspberry Pi 3B+ devices, a low-power x64 edge device, and a cloud virtual machine (VM) to establish our platform. This platform was meticulously developed and rigorously tested on Knative Serving 1.7 and Kubernetes 1.24. Our experiments involved four real-world workloads: matrix multiplication (MatMult), image processing (Image Proc.), random I/O, and a combination of these three loads (Mixed). These workloads exhibit varying demands on CPU, memory, disk, and network resources during execution, making them suitable targets for evaluating our platform’s capabilities. We generated requests for these workloads using a specialized experiment runner, allowing us to control the request rate: initially, we used a low request rate, then increased the rate linearly to a high request rate, which was maintained until the end of the run. The rates were chosen in such a way that the low request rate could be handled entirely by the edge devices and the high request rate would overload the edge devices under no offloading. Additionally, we employed an edge scheduling strategy to determine how traffic was distributed between the Edge and the Cloud. This strategy could be configured to various distribution levels (0%, 25%, 50%, 75%, 100%, and auto) in our experiments, as detailed in Table 2. These settings enabled us to assess the impact of different workload types and edge scheduling strategies on the platform’s performance. We measured several performance metrics, including average response time, CPU and memory utilization of the edge devices, as well as network utilization of the cloud VM.

### 4.2 Results and Evaluation

Figure 2 illustrates the average latency of the responses, the average CPU and memory utilization of the edge devices, and network utilization of the cloud VM, all measured over the course of each experiment run.

**Latency.** The offloading functionality improves considerably response times. For instance, our solution initially exhibits slower reaction times, leading to reduced response times. However, as more requests are offloaded to the Cloud, the response time converges towards the lower bound given by our edge network conditions as soon as the request rate is increased (see Figure 2 (Latency)).

**CPU.** Our results indicate that our solution can effectively offload excessive workload from the Edge cluster but may also underserve



Figure 2: Performance results, including latency (seconds), CPU, memory, and network usage, are presented for workload traffic (refer to Table 2).

many requests (see Figure 2 (CPU)). As our algorithm optimizes the response times of workloads, CPU utilization is reduced as a consequence. However, determining the optimal level of CPU utilization is more complex. If the CPUs of edge devices are not fully utilized but requests are being forwarded to the Cloud, Edge cannot be efficiently utilized. Furthermore, reserving spare CPU resources could also benefit the Edge cluster when they are needed for possible bursts of requests or system operations. Finally, our test bed uses Raspberry Pi 3B+ as edge devices which have a low CPU power, and more powerful edge devices, such as Nvidia Jetson, might need more analysis and investigation in order to optimally balance where requests should be served from.

**Memory.** We assess that, in most cases, memory utilization remains within reasonable bounds. Our initial assessment of the workloads suggested that we would observe a significant increase in memory utilization for both image processing and matrix multiplication. While the latter did see a noticeable increase in memory consumption, the former has seen a much-lessened effect, performing similarly to the random I/O workload, which we estimated to have no increase in memory (see Figure 2 (Memory)).

**Network.** Bandwidth saturates for image processing and matrix multiplication when all requests offload to the cloud, however the mixed workload never hits the maximum of 100MB/s. Similarly, our network ingress findings indicate that image processing and matrix multiplication are constrained by Edge-to-Cloud network bandwidth at full offloading. In case the network is the bottleneck (which is in the case of the matrix multiply and mixed workloads), then the offloading does not help. It would make the response times worse, depending on how they compare to the edge. Additionally, our offloading strategy does not take into account the network latency or bandwidth between the Edge and Cloud environments, and a more sophisticated strategy is required to optimally offload in different network conditions.

## 5 CONCLUSION

The topic of serverless computing on the Edge-Cloud continuum is still in its infancy. This paper aims to provide valuable insights into the design of serverless platforms for a unified Edge-Cloud computing environment. We presented our platform, an extension of the serverless platform Knative that enables deployments across edge and cloud environments and offloads requests from edge devices to the cloud. We demonstrated that our approach can deploy a serverless application across edge and cloud environments and demonstrated its offloading capability. In future, we will explore more about offloading strategies for resource optimization and performance improvement.

## ACKNOWLEDGMENTS

This work has been partially funded by the European Union's Horizon 2020 research and innovation program through the project CLARIFY (860627), the ENVRI-FAIR (824068) project, BlueCloud-2026 (101094227) project, by the NWO LITE-LIFE project and by the LifeWatch ERIC.

## REFERENCES

- [1] ABDELLATIF, A. A., MOHAMED, A., CHIASSERINI, C. F., TLILI, M., AND ERBAD, A. Edge computing for smart health: Context-aware approaches, opportunities, and

- challenges. *IEEE Network* (2019).
- [2] AI, Y., PENG, M., AND ZHANG, K. Edge computing technologies for internet of things: a primer. *Digital Communications and Networks* (2018).
- [3] ASLANPOUR, M. S., TOOSI, A. N., CICONETTI, C., JAVADI, B., SBARSKI, P., TAIBI, D., ASSUNCAO, M., GILL, S. S., GAIRE, R., AND DUSTDAR, S. Serverless edge computing: vision and challenges.
- [4] BALDINI, I., CASTRO, P., CHANG, K., CHENG, P., FINK, S., ISHAKIAN, V., MITCHELL, N., MUTHUSAMY, V., RABBAH, R., SLOMINSKI, A., ET AL. Serverless computing: Current trends and open problems.
- [5] CASTRO, P., ISHAKIAN, V., MUTHUSAMY, V., AND SLOMINSKI, A. The rise of serverless computing.
- [6] CHEN, B., WAN, J., CELESTI, A., LI, D., ABBAS, H., AND ZHANG, Q. Edge computing in iot-based manufacturing. *IEEE Communications Magazine* (2018).
- [7] DILLEY, J., MAGGS, B., PARIKH, J., PROKOP, H., SITARAMAN, R., AND WEIHL, B. Globally distributed content delivery. *IEEE Internet Computing* 6, 5 (2002), 50–58.
- [8] FERRY, N., DAUTOV, R., AND SONG, H. Towards a model-based serverless platform for the cloud-edge-iot continuum. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* (2022).
- [9] GADEPALLI, P. K., PEACH, G., CHERKASOVA, L., AITKEN, R., AND PARMER, G. Challenges and opportunities for efficient serverless computing at the edge. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)* (2019).
- [10] HU, S., HUANG, S., HUANG, J., AND SU, J. Blockchain and edge computing technology enabling organic agricultural supply chain: A framework solution to trust crisis.
- [11] JEON, H., SHIN, S., CHO, C., AND YOON, S. Deep reinforcement learning for qos-aware package caching in serverless edge computing. In *2021 IEEE Global Communications Conference (GLOBECOM)* (2021).
- [12] KHAN, L. U., YAQOOB, I., TRAN, N. H., KAZMI, S. A., DANG, T. N., AND HONG, C. S. Edge-computing-enabled smart cities: A comprehensive survey. *IEEE Internet of Things Journal* (2020).
- [13] KURNIAWAN, A., LAU, W., KURNIAWAN, A., AND LAU, W. Introduction to azure functions.
- [14] LIN, J., YU, W., YANG, X., ZHAO, P., ZHANG, H., AND ZHAO, W. An edge computing based public vehicle system for smart transportation. *IEEE Transactions on Vehicular Technology* (2020).
- [15] LV, Z., CHEN, D., LOU, R., AND WANG, Q. Intelligent edge computing based on machine learning for smart city. *Future Generation Computer Systems* (2021).
- [16] MALAWSKI, M., GAJEK, A., ZIMA, A., BALIS, B., AND FIGIELA, K. Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions.
- [17] MORABITO, G., SICARI, C., CARNEVALE, L., GALLETTA, A., DI MODICA, G., AND VILLARI, M. Securing serverless workflows on the cloud edge continuum.
- [18] NASTIC, S., RAITH, P., FURUTANPEY, A., PUSZTAI, T., AND DUSTDAR, S. A serverless computing fabric for edge & cloud. In *2022 IEEE 4th International Conference on Cognitive Machine Intelligence (CogMI)* (2022).
- [19] O'GRADY, M., LANGTON, D., AND O'HARE, G. Edge computing: A tractable model for smart agriculture?
- [20] OUEIDA, S., KOTB, Y., ALOQAILY, M., JARARWEH, Y., AND BAKER, T. An edge computing based smart healthcare framework for resource management. *Sensors* (2018).
- [21] QI, Q., AND TAO, F. A smart manufacturing service system based on edge computing, fog computing, and cloud computing. *IEEE access* (2019).
- [22] RAITH, P., RAUSCH, T., DUSTDAR, S., ROSSI, F., CARDELLINI, V., AND RANJAN, R. Mobility-aware serverless function adaptations across the edge-cloud continuum.
- [23] SATYANARAYANAN, M. The emergence of edge computing. *Computer* (2017).
- [24] SBARSKI, P., AND KROONENBURG, S. *Serverless architectures on AWS: with examples using Aws Lambda*. 2017.
- [25] TANG, Q., XIE, R., YU, F. R., CHEN, T., ZHANG, R., HUANG, T., AND LIU, Y. Distributed task scheduling in serverless edge computing networks for the internet of things: A learning approach. *IEEE Internet of Things Journal* (2022).
- [26] WANG, B., ALI-ELDIN, A., AND SHENOY, P. Lass: Running latency sensitive serverless computations at the edge. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing* (2021).
- [27] ZHOU, X., KE, R., YANG, H., AND LIU, C. When intelligent transportation systems sensing meets edge computing: Vision and challenges. *Applied Sciences* (2021).