Event-Driven FaaS Workflows for Enabling IoT Data Processing at the Cloud Edge Continuum

Christian Sicari, Daniel Balouek, Massimo Villari, Manish Parashar

Event-Driven FaaS Workflows for Enabling IoT Data Processing at the Cloud Edge Continuum

Christian Sicari
csicari@unime.it
University of Messina
Messina, Italy

Daniel Balouek
daniel.balouek@inria.fr
IMT Atlantique, INRIA, LS2N, UMR CNRS 6004, F-44307
Nantes, France

Massimo Villiari
mvillari@unime.it
University of Messina
Messina, Italy

Manish Parashar
manish.parashar@utah.edu
SCI Institute, University of Utah
Salt Lake City, UT, United States

ABSTRACT

Continuum Computing encompasses the integration of diverse infrastructures, including cloud, edge, and fog, to facilitate seamless migration of applications based on their specific needs, ensuring optimal satisfaction of their requirements. The primary obstacles in this particular context mostly pertain to the incapacity to promptly respond to changes in the environment or the quality of service (QoS) constraints of the application, as well as the incapability to maintain an application in a stateless manner, hence impeding its relocation without the risk of data loss. The objective of this research is to tackle the aforementioned issues through the introduction of a framework based on Function-as-a-Service (FaaS) and event-driven architecture. This framework enables the decomposition, localization, and relocation of applications inside a Continuum infrastructure, facilitated by a rule engine that is both system and data-aware.

ACM Reference Format:
Christian Sicari, Daniel Balouek, Massimo Villiari, and Manish Parashar. 2023. Event-Driven FaaS Workflows for Enabling IoT Data Processing at the Cloud Edge Continuum. In *(UCC '23).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3492323.3495590

1 INTRODUCTION

As one of the largest online data sources, the Internet of Things (IoT) is currently being utilized for a wide range of applications, including smart city and environmental surveillance [42], sports [40], healthcare [6], and Industry 4.0 (IIoT) [35]. Real-time processing of Internet of Things (IoT) data is necessary when the data is time-sensitive and needs to be used immediately; this means that data is processed as it is generated instead of being stored for analysis later. However, there are many variables that can affect whether or not data from one or more sensors must be used immediately, such as Quality of Service (QoS) constraints, user or human needs, or the data itself.

From those limitations, the idea of continuum computing emerged, aiming to leverage the popular cloud, fog, edge, and IoT infrastructures to execute data analysis algorithms at the optimal moment [9], taking into account variables such as pipeline optimization [18], data location [36], energy [22], and network latency [22], which all relate to data location. Unfortunately, there are no Continuum native apps, as the scientific literature [5, 11] points out, therefore anything that is meant to work on the Continuum has to be redesigned.

The term "continuum computing" describes the idea of smoothly integrating computer services and resources across many platforms and devices, enabling users to access their data and apps from anywhere at any time, on any device. Thus, the ability to seamlessly leverage any continuum layer and move the computation over its infrastructure is what it means to be continuum native[34]. However, moving an application—which is typically made up of numerous microservices—to meet dynamic requirements is still a challenging task[4, 10]. These reasons include:

(1) inability to respond dynamically to changes in the environment, application, or data;
(2) inability to maintain an application's statelessness, which allows for relocation without causing data loss;
(3) inability to profile the behavior of a particular task and subsequently predict its future requirements.

The R-Pulsar software stack [29] has been introduced recently to tackle the first issue mentioned. This continuum-native framework can federate cloud, edge, and IoT infrastructures by connecting data consumers and providers through an advanced profile match system, and it can relocate data and analysis through a dynamic rule engine system that can recognize changes in the data and respond appropriately.

Function as a Service (FaaS) must be introduced and used to address the remaining two problems. FaaS was first developed for the cloud but has just lately been deployed in edge [16, 26] and sporadically in continuum [32, 33]. Essentially, Function is a generic code that FaaS platforms wrap inside a multi-architecture container, expose via an HTTP or Pub/Sub interface, deploy across

The paper is partially supported both by the Italian Ministry of Health Piano Operativo Salute (POS) trajectory 2 "eHealth, diagnostica avanzata, medical device e mini invasività" through the project "Rete eHealth: AI e strumenti ICT Innovativi orientati alla Diagnostica Digitale (RAIDD)"(CUP J43C22000380001). This research was supported in part by the NSF under grants numbers OAC 2238064.

UCC '23, December 6–9, 2023, Italy

ACM ISBN 978-1-4503-9163-4/21/12…$15.00
https://doi.org/10.1145/3492323.3495590

an orchestrator like Kubernetes or Faasd, and then call upon it upon the occurrence of a predefined event. Since FaaS functions are stateless by design, there won't be any data loss if the host executing them changes. Second, having a well-known named function allows us to profile, examine, and predict the behavior of the object in the future.

Our goal in this effort is to jointly solve the aforementionned challenges in the field of continuum computing. We aim at accomplishing this by the means of the R-Pulsar framework by federating the infrastructure, orchestrating the computation, and responding to changes in the environment or in the data. In addition, we improve it by offering a distributed Faas layer that enables the definition, deployment, relocation, and analysis of functions. This paper offers for the following contributions:

- compute at the Continuum using FaaS functions;
- monitor and analyze functions when they are run on various environments with varying configurations and data;
- dynamically respond to changes in the environment or data by adapting and relocating functions on the continuum.

The remainder of the paper is organized as follows: Section 2 discusses the motivational scenario while 3 analyzes related works and . Section 4 describes the building blocks on which this work starts. In Section 5, the components of this new work are presented and in Section 6, we analyze its performance. Finally, in 7, we summarize this paper and discuss future works.

2 USE CASE

An urgent computing situation across the Edge-Cloud Continuum-using IoT to detect fires in large, remote areas—is the motivation for this study [2]. The process involves employing video cameras and air analyzer sensors to identify smoke in the impacted areas. After that, the gathered data is preprocessed at the network edge using the computational power on board to examine its content and decide whether more data collection or postprocessing is required. Should additional processing be necessary, the information is sent to the cloud for change detection analysis. This analysis can involve comparing the data to historical records or simply for storage purposes.

This scenario can be extended to a wider class of problems involving highly-accurate computational models coupled with high-fidelity data streaming from multiple data sources, a recent development in Urgent Computing to efficiently monitor and manage the effects of extreme events and natural disasters [3, 37].

In order to implement time-sensitive computing applications, developers and service providers must coordinate data-driven workflows effectively. This coordination must consider not only the significant heterogeneity of the underlying cyberinfrastructure but also the inherent uncertainty associated with the availability and reliability of the data sources [20]. Consequently, the overall performance of applications deployed across the continuum relies on programming abstractions that enable the expression of application behaviors capable of responding to unexpected events during runtime. Additionally, autonomic runtime mechanisms are necessary to facilitate the adaptation of resources and execution paths across the continuum. In addition, the complex interplay between resources and system dynamics presents additional challenges in implementing time-sensitive computing applications [13].

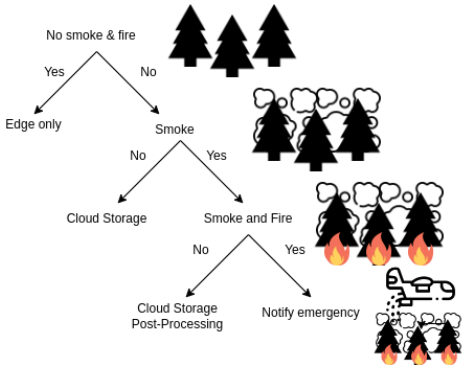


Figure 1: Urgent computing decision stages and reactions based on collected data

3 RELATED WORK

Architecture-oriented approaches for addressing the Continuum. The continuum problem has been the subject of numerous recent architectural works. For example, in [8], authors proposed a data-driven model to publish and compute data globally, essentially addressing the continuum problem statically by deploying a consumer in a specific node, thus restricting the abilty to move around. A custom pub/sub broker with relocatable customers is suggested in [21] as a comparable concept, although even in that scenario, the placement of the data storage does not provide any optimization. In [30], the authors propose to leverage Kubernetes for managing a cloud-edge federation. They then find pods and improve the network latency between relevant pods using a custom scheduler. Though many people are using this method, Kubernetes still requires the continuum to have a centralized master-slave design, and the optimum parameters only take network bandwidth into account.

A federated learning strategy for scheduling tasks from a task queue has been presented in other publications [23]. Unfortunately, as the writers themselves point out, figuring out the task's nature is not always easy, and determining the ideal spot is frequently challenging. An alternate technique is to analyze historical data to determine a task's optimal placement [17]; however, this approach is dependent on meaningful patterns throughout time. A scheduling technique based on the declaration of required data and the degree of connection with other tasks is proposed by the authors in [17] in an effort to maximize request performances and improve feedback to the subsequent schedules.

This section of the State of the Art emphasizes how understanding the task at hand is essential to handling it effectively. Unfortunately, unless they fall outside of a particular applied domain, generic process tasks are difficult to categorize.

Leveraging Function-as-a-Service for the Continuum. The benefits of using the FaaS paradigm include improved task knowledge, which is encapsulated in a function and partially assists with task

accounting. Using FaaS at the continuum is still considered a novelty, however some recent works have been proposed with this goal.

The work previously presented in [8] was expanded upon by the authors in [7], who improved the Fiware-based infrastructure with Faas capabilities at the Fog layers. The authors here can profile functions with ease, but they don't use that to find data or do calculation more effectively. The creators of [12] continue to use the Fiware stack to subscribe to Faas platform ads at the network's edge. However, the bottleneck of having a single gateway is removed by the replication of many Faas, and the data are not scaled because they are duplicated in all the context brokers where they are required. The work from [30] was expanded upon in [31], wherein OpenFaas pods were geo-scheduled using Kubernetes pods. Although this method ensures a dependable scheduler, restricted edge devices may find using a Kubernetes cluster to be excessively demanding.

The authors of [38] suggest a data-declaration-based algorithm for scheduling OpenWhisk-based functions; although taking data location into account helps determine where to compute, it does not ensure system flexibility. Additionally, OpenWhisk's poor support for limited arm devices could pose a challenge for edge computing [39]. Lastly, the network-based scheduler has been utilized once more, including for Faas scheduling in [27]; however, this method just employs a static dataset to more accurately statically position a function at the edge. It does not learn from the experience.

Lastly, the authors of [24] suggest using an efficient Serverless Workflow engine based on Kubernetes to distribute operations at any size. Although functionality in this project can be scaled and deployed globally in accordance with user specifications, Kubernetes may prove challenging to maintain at any scale.

4 BACKGROUND

4.1 The R-Pulsar software stack

Given the fact that Internet of Things (IoT) applications necessitate the processing of substantial amounts of streaming data within complex workflows, depending exclusively on cloud resources becomes unfeasible. Hence, the strategic utilization of resources in close proximity to the edge assumes paramount importance, despite their inherent limitations in terms of capabilities. Therefore, it is imperative to achieve an appropriate balance between the aspects of data processing, namely quality, immediacy, and cost, in a manner that takes into account the contextual factors. The R-Pulsar system has been developed with the objective of constructing data-driven pipelines for applications that span both the cloud and the network edge. The primary objective of this project is to enhance the capabilities of edge devices by extending cloud functionalities to the edge. This integration allows for the collection and analysis of data in close proximity to its source, enabling autonomous responses to local events [29].

Initial investigations have centered on the enhancement of the Associative Rendezvous (AR) interaction paradigm, with the objective of extending its capabilities for supporting data-driven Internet of Things (IoT) applications. The AR paradigm enables decoupled interactions based on content, where interactions are defined using semantic profiles instead of names in an asynchronous manner. These interactions provide the ability to implement reactive behaviors, which serve as the fundamental services for executing data-driven workflows and making decisions. The preliminary findings of a study on a specific disaster recovery scenario [29] have shown promising results in terms of validating and assessing the proposed extensions. The augmented reality (AR) model was utilized to facilitate the implementation of workflow topologies, which were triggered and scheduled based on the content of data streams. Recent developments focused on the challenges associated with developing time-sensitive applications and put forth a a high-level methodology for managing data-driven decisions [1], and a prototype linking edge, cloud and HPC resources for coupling fire science and air quality scenarios [2].

The R-Pulsar system utilizes a distributed architecture that consists of five distinct layers. These layers include the self-organizing overlay, the content-based routing layer, the serverless messaging layer, the memory-mapped data processing layer, and the programming abstraction layer. **The self-organizing overlay**, which integrates location awareness, functions as a conceptual representation for the layers positioned above it. By exposing only a single function to the other layers, it becomes possible to direct AR Messages to the nearest R-Pulsar node in proximity to the data source. **The Content-based Routing Layer** is an extension of the self-organizing overlay in order to enhance the functionality of message routing between clients of R-Pulsar. The system utilizes the underlying infrastructure to effectively route messages. **The Memory-mapped Streaming Analytics Pipeline** is responsible for consolidating data from various sources, processing it, and making it accessible for further utilization. **The Serverless Messaging Layer** enables the deployment and execution of code fragments as reaction to some specific bound events without requiring the explicit specification of IP addresses. The **programming abstraction layer** provides a set of customizable If-Then rules based on individual values or statistical trends of streaming windows to facilitate the programming of reactive behaviors.

4.2 FaaS computing at the Continuum

The Faas paradigm aims at streamlining the process of developing and deploying cloud-native applications. The emergence of the product in question can be traced back to about 2017, coinciding with the start of the AWS Lambda project. Subsequently, other prominent e-commerce businesses have entered the market with similar offerings [14, 15]. Function-as-a-Service (FaaS) computing is a paradigm that enables the encapsulation of a basic stateless application, referred to as a "function," within a pre-existing container. Subsequently, the FaaS platform has the responsibility of enveloping this function and furnishing an interface, commonly HTTP-based, via which interactions with the function can occur. The underlying container orchestrator assumes responsibility for monitoring and load balancing the container.

In recent times, there has been a significant surge in the popularity of some open-source projects within the realm of Function-as-a-Service (FaaS). Notably, Knative [19] and OpenFaas [25] have garnered considerable attention. Of particular note, OpenFaas has shown widespread adoption, even in resource-limited and edge computing environments [28].

The utilization of edge and cloud computing enables the execution of functions, therefore concealing the architectural distinctions from end users. This has led to the emergence of Function-as-a-Service (FaaS) as a potential means of proliferating computing across the continuum, as evidenced by recent research studies [12, 32]. The primary obstacle faced by serverless applications pertains to the ability to establish connections between functions in a graph-like manner, enabling the composition of intricate continuous systems without relying on a central coordinating point. In a prior study, authors have introduced OpenWolf, a serverless-oriented broker that facilitates the distribution and orchestration of functions across a Kubernetes federation comprising diverse nodes[1]. This is achieved through the utilization of a manifest, which consists in a JSON file that draws inspiration from the serverless workflow description language (source). It serves the purpose of initializing functions in various contexts and can also be used to describe function interactions using a Boolean equation language. The ultimate outcome is a distributed function-based graph, as depicted in Figure 2.

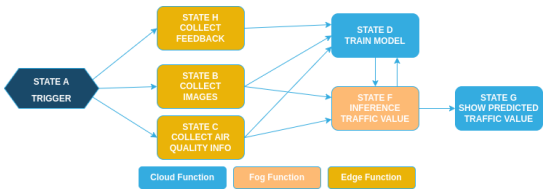


Figure 2: FaaS continuum workflow

4.3 Dynamic Faas scheduling

The limitations of a system like OpenWolf include the fixed allocation of a workflow function to a predetermined tier, as determined at the beginning of the manifest or as decided by OpenWolf during the workflow bootstrap process. Additionally, there is difficulty in dynamically linking newly incoming functions to existing ones. Regrettably, as emphasized in section 3, the prioritization of a particular computation is subject to fluctuations based on factors such as incoming data, system overload, or external changes. Consequently, adhering strictly to a predetermined function schedule does not adequately address the necessity of repositioning a function based on its urgency. Furthermore, in response to newly acquired data, the system may necessitate the use of novel functions inside the existing workflow. This would entail the implementation of a fresh workflow manifest, resulting in potential time wastage or the modification of function invocation parameters.

In contrast, RPulsar inherently facilitates the dynamic binding of new producers and consumers, as well as the ability to respond to changes in the data or system status. However, RPulsar lacks support for Function as a Service (FaaS), hence missing out on the various advantages it offers.

The works of Rpulsar and OpenWolf exhibit an obvious complementarity, as the objective of this work is to address the inherent weaknesses of one work by leveraging the strengths of the other.

[1]https://github.com/christiansicari/OpenWolf

In order to achieve this, we retained the RPulsar core functionality that enables the dynamic association of producers and consumers. However, we made a modification to the structure of the consumer. It is now implemented as a function wrapper, encompassing both the reference to the function it is intended to execute and the configuration settings that override the default values. At this point, the consumer is capable of assuming the role of a producer, thereby generating system data as an output. Subsequently, more consumers can be associated with this output. The necessity of a manifest is obviated as the definition of a Workflow is implicitly established via the binding of producers and consumers. The rule engine from RPulsar has been preserved, albeit with modifications to accommodate the Function monitoring system. These modifications enable the generation of reactions, such as the execution of a function at the edge instead of the cloud, in response to changes in data, required Quality of Service (QoS), or system overloading.

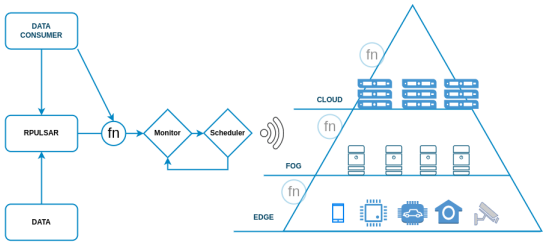


Figure 3: Function spreading on Continuum

5 ARCHITECTURE

As depicted in Figure 4a, the architectural structure of this work consists of three distinct layers, namely the Infrastructure Layer, Workflow Layer, and Application Layer.

5.1 Infrastructure Layer

The infrastructure layer includes all the hardware and primary services used in RPulsar, such as IoT devices and sensors used to capture and send data on RPulsar, and the Continuum infrastructure composed of edge, fog, and cloud nodes. Independent of their type, Continuum nodes are composed of a hardware part (embedded processors, workstations, and data centers) and a service part composed of a serverless platform and/or a storage area.
The **Serverless Platform** is the core of the project, and we implemented it using OpenFaaS. OpenFaaS is an open-source serverless platform that lets one build, deploy, and manage functions using Kubernetes or faasd. OpenFaas is composed of:

(1) The gateway that lets us invoke functions or use API to interact with OpenFaas;
(2) NATS that collect asynchronous function requests;
(3) Prometheus, which provides metrics about the running functions;

We have chosen OpenFaas among all other open-source FaaS providers because of its performance, low resource utilization, ability to work on multiple architectures, and support of asynchronous functions. The **Storage Area** is realized using Minio, an S3-compatible object storage, and it plays a critical role in providing a

(a) Architecture overview **(b) ARMessage exchange sequence diagram**

Figure 4: Architecture overview and message exchange

space where functions can store and retrieve input and output data that, in turn, are used by all the functions in the triggered workflow. The Rule Engine at the Service Layer determines which storage area has to be used as well as where to run a function.

5.2 Workflow Layer

The Workflow Layer can be considered an abstract layer since it is not composed of components or hardware like the Infrastructure and the Application layers, but it is shaped by the interaction of those layers. This layer is composed of the functions and the objects that are available during the life cycle of a workflow. Each function is represented using the *fn* symbol, and they are double linked with one (or more) object represented with the *obj* symbol, and both are connected to the Infrastructure Layer. These links exist because each object belongs to one or more functions; on the contrary, a function consumes and produces objects. This map between functions and objects is not hidden to the final users that submit data consumers to RPulsar. The Rule Engine determines the Workflow Layer and Infrastructure Layer links, which locates and relocates functions and objects to satisfy some given constraints. In this case, this mapping is hidden to the final users, who are not interested in where functions and data are located but in the satisfaction of the QoS parameters they defined. Of course, functions interact with each other accordingly to their profile matches, and this implicitly creates a FaaS workflow.

5.3 Application Layer

The Application Layer contains all the services needed to design, trigger and adapt a workflow over the infrastructure layer, and it is composed of the Data Log, the Function Repository, the Enhanced Proxy, the Rule Based Engine, and the Data Consumers.

The **Data Log** is a distributed MongoDB instance, and each peer contains the sets of information related to the continuum host where they are installed. Each peer is filled with the information that the Enhanced Proxy can provide about the function's execution that happened in that node, and it can be queried to build metrics-based scheduling rules.

The **Function Repository** is a centralized service used to publish, version, and retrieve functions that can be used immediately on any node at the Infrastructure Layer. Functions are stored using pre-configured docker container images that allow building functions in Python, Java, Golang, Javascript, and Ruby. Each function can be cross-compiled and published for different architectures, like armv7, armv8, amd64, or ppc64le. The deployment of a function on a node will success if the right architecture image is available. Using a single Function Registry improves the code reusability, letting share functions among all the RPulsar users instead of using custom code at any time.

The **Enhanced Proxy** redirects the requests to a function to the Infrastructure node where that function is run for that specific workflow. In this way, we avoid directly interacting with a function, then hiding the composition of the Infrastructure Layer. This proxy, while forwarding the requests and the responses to and from a function, asynchronously queries the Prometheus instance installed in the Serverless Platform where the function has been run and stores the fetched metrics in the Data Log to be used by the Rule Engine.

The **Rule-Based Engine** is the highest level service component that takes advantage of all the previous components to let the final users to define rules that drive the position and relocation of a specific function inside a workflow. Those rules can be based on one or more of these factors:

- Incoming data;
- current loading of the system;
- History about previous similar jobs.

The policies can be evaluated at any new incoming data, then letting RPulsar dynamically spread and optimize the computation (and storage) on the Continuum when needed.

[caption=Payload based Rule,label=lst:payloadrule,language=Java]
if(payload < 5) invokeFunction(faas=FZoneX, f=function, data=payload, cfg=config, zone$_out$ = SX)$elseinvokeFunction(faas = FZoneY, f = function, dat$

In the listing **??**, we define a rule that selects a node where to run a function based just on the content of the payload. Depending on the value and the meaning of this value, we can define where to compute and where to store the function's output.

[caption=Metrics based Rule,label=lst:metricsrule,language=Java] node=Datastore.getMetrics("TTR < 10s", "1d").sort("TTR: ASCENDING")[0] invokeFunction(faas=node, f=function, data=payload, cfg=config, $zone_out = node.closest()$)

In the listing **??**, instead, we are accessing the datastore to find the list of nodes where the Time to Respond (TTR) has been less than 10 seconds in the metrics collected in the last day, then we retrieve the first node from this list. The invocation of the function will then use the retrieved node to run the function, and it will store the function's output in the closest storage zone.

5.4 Message Exchange

RPulsar is able to build and activate Workflows using an advanced Pub/Sub model based on the matching of producers' and consumers' profiles. This process is deeply explained in [29], but we needed to modify them in order to include the use of the FaaS.

To start a workflow, we need that producers and consumers exchange ARMessages in the order shown in Figure 4b. Producers and Consumers do not know each other, then they just interact with an RPulsar node (RP), which will create a communication channel between them later. In the figure, the Consumer C1 sends a NotifyData message, this message is used to let RP know that it is interested in data that has a profile *p1*, and when it receives it, it will apply a *foo()* function on it, providing a result with a *p2* profile (C1 will become a producer too then). Afterward, a producer (P1) sends a NotifyInterest message to RP, saying that it can send data with *p1* profile.

RP notices that C1 is interested in consuming P1's data; namely, there is a match. RP then activates the Rule Engine and schedules the C1's *foo()* function on one of the Serverless Platforms (OF) and the Object Storage (OS) to use in the Infrastructure Layer. RP will use the OpenFaas API to deploy the function on the scheduled node, it will receive back the endpoint to use to invoke the function. When the OF is ready, RP will send a NotifyMatch message to P1, then P1 will start to produce and send data. Finally, RP will invoke the function on OF sending the P1's data in the payload. Once executed, the function will store the result on the OS and inform RP that the computation is completed. If a new consumer C2 interested on data with profile *p2* will appear, this workflow restarts, but this time the producer will be acted from C1, that instead of sending directly the data will tell RP the OS address where the data are.

6 PERFORMANCE ANALSYIS

6.1 Evaluation methodology

In order to evaluate the enhancements made in this enhanced version of RPulsar, a Continuum scenario has been set up. In this scenario, RPulsar assumes the responsibility of both publishing and receiving Internet of Things (IoT) data through the utilization of Function-as-a-Service (FaaS) capabilities. In this part, a series of tests were conducted to compare the suggested processing technique with a conventional approach in which stream processing is situated at a fixed place within the infrastructure's core.

6.1.1 Dataset. In the scenario we considered, we aim at consuming sensor and camera data from the SAGE platform [2] to detect the presence of smoke. To publish sensor data, we will locate an RPulsar producer at the Edge of the network, while a second RPulsar node will consume it using a function that can be located in any FaaS Zone.

6.1.2 Testbed. In this testbed, we considered three zones located at the edge of the network where data are even produced, in the cloud, specifically in a High-Performance Data Center, and in one in the fog, namely in the middle of the route between the edge and cloud zones. More details about those zones are given in Table 1.

6.1.3 Baselines. We are interested in two metrics, the Computation Time (CT) and the Request Time (RT). The CT metric measures just the time to execute the function on the payload and get a result; it does not include any other task. The RT measures the time elapsed from when a request is started to when the result is sent back to the client. Indeed, this metric involves the CT, as well as the network card's ability to compress and decompress the request, and the network bandwidth used to transfer the data from the client to the serverless platform. These two metrics are obtained from the Data Log component and can even be used to build performance-oriented deployment rules, as seen before.

We tested the Edge-Fog-Cloud Serverless environment, considering two main parameters: the payload size and the number of concurrent requests. Regarding the payload size, we have considered two different kinds of datasets, the lightweight, composed of five payloads of sizes 100KB, 200KB, 300KB, 400KB, and 500KB; and the medium weight, composed of five payloads of size 1MB, 1.5MB, 2MB, 2.5MB, and 3MB.

We selected these two datasets among all the possible choices because those are the smallest that allow us to demonstrate a change of behavior in terms of performances between the cloud, the fog, and the edge nodes. With respect to concurrent requests, we considered 11 different use cases, from 1 request at a time to 100. Even in this case, these values allow us to study a change of behavior in the computation layer.

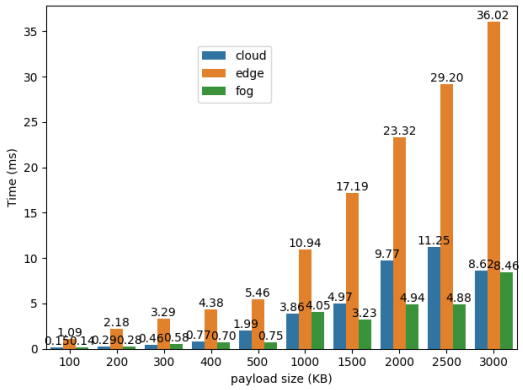


Figure 5: Best Computation Time (CT) on each zone increasing payload size

The first simple test is shown in Figure 5. The test lets us see the difference in terms of hard computation on all three zones when they compute heavier payloads. As totally expected considering

[2] https://sagecontinuum.org/

Table 1: Cluster's nodes characteristics

[gray]0.8**Instances**	[gray]0.8**Continuum Tier**	[gray]0.8**CPU**	[gray]0.8**Memory**	[gray]0.8**Operating System**	[gray]0.8**Location**	[gray]0.8**Network Upload Speed to Server**	[gray]0.8**Network Download Speed from Server**
1	Edge	Cortex-A72 4-cores 1.8 GHz	8 GB	Ubuntu 18.04 Aarch64	CloudLab Utah Datacenter	17.6 Gbits/sec	17.6 Gbits/sec
1	Fog	Intel i7-5930K 12-cores 3.50GHz,	32 GB	Ubuntu 22.04 Amd64	SCI Institute, Utah	543 Mbits/sec	542 Mbits/sec
1	Cloud	Intel Xeon 54-cores 2.00GHz	254 GB	Ubuntu 18.04 Amd64	Gigabit P2P CloudLab Clemson	407 Mbits/sec	406 Mbits/sec

the resources described in table 1, the edge zone performs worse than cloud and fog zones for any payload size, and the gap between the zones increases, increasing the payload size by a super linear factor.

Fog and cloud zones, instead, keeps comparable performance results for any payload size.

6.2 Evaluation results

Based on the collected data presented in Figure 5, together with the measured network latency between the tiers utilizing the iperf3 tool [3], as documented in Table 1, it is possible to calculate the optimal RT value for various payload sizes. The comparison between the ideal value and the actual value has been conducted, and the findings are presented in Figure 6. The chart illustrates that actual performances generally are inferior compared to ideal performances, particularly for cloud and fog. Moreover, there is a constant delta between real and ideal values. This assertion is untrue in the edge where the difference between real and ideal performances becomes negligible with small computed data. However, when considering medium-weight datasets, the real value significantly deteriorates performance, hence amplifying the delta between it and the ideal value. This trend suggests that the edge layer may exhibit better performance under light workloads, mostly due to minimal network use. However, it is anticipated that this performance may significantly fluctuate when subjected to larger workloads, as we want to evaluate in the near future. In general, we can affirm that when the payload is small and therefore the pure computation time is minimal, the effective time spent to consume a request is taken by environment and infrastructure factors: such as network bandwidth, system load and internal process scheduling.

In figure 7, we measured the RT, when just one request is sent per time on all three tiers. Figure 7a shows the performance guaranteed using the small dataset. Here, despite the tests shown in figure 5, the edge dominates both the performance of cloud and fog; in fact, the involvement of the network that is needed to send the data away from the edge outweighs the time needed to compute the data, producing the shown result.

Subfigure 7b instead represents the same information in figure 7a, but using the medium-weight dataset. Results shown here totally revert to what was discussed previously; in fact, in the edge, we are no more able to efficiently compute incoming requesting, performing overall worse than both the cloud and the edge. Respect the previous figure, it is absolutely interesting even to analyze the stability at the edge; in fact, while for small payloads, all the tests run on the edge have almost the same performance, we see many more outliers and bigger interquartile ranges for bigger payloads.

Figure 8 illustrates the RT when requests arrive with an increasing parallel factor and the payload size is fixed, in an effort to evaluate the scaling capabilities on the three tiers.

The payload in subfigure 8a is fixed at 400KB. The edge and cloud indicate a more stable behavior than the fog in this instance. Specifically, the edge node needs 16 times longer to process 100 requests than the cloud node, which does not significantly alter its RT. In spite of this, there are very few outliers and all executions are fairly near to the median value. On the other hand, fog node performs worse than both cloud and edge, and its anticipated value is unreliable due to a large number of executions that deviate from the median value.

In conclusion, the scaling property is examined in subfigure 8b when the payload size is set at 3 MB. We have already shown that the edge performs the worst in these circumstances; in fact, this behavior is even more pronounced when there are concurrent requests. As we can see from the figure, the edge performs worse overall on all scales, but there is also a high degree of instability in the results, which is indicated by the enormous interquartile ranges in the box plots. One important thing to keep in mind when doing urgent computing is the stability of the outcome. When 100 simultaneous requests are taken into account, as shown in this image, the best median value we can get is in the edge, which should respond in 832 ms as opposed to 2845 ms in the cloud and 3465 ms in the fog. It is clear that Edge and Fog may do far worse than Cloud in the worst scenario if we include even the outliers and the highest values in the boxplots. Because of this, cloud reliability is significantly higher when ensuring a result within a given time frame is crucial. However, the edge can be a preferable option if our goal is to answer as quickly as possible and the risk of missing this deadline is acceptable.

6.3 Summary

The objective of these experiments was to assess the capabilities of a continuum environment in terms of performance, specifically when numerous independent serverless layers are distributed across both cloud and edge infrastructures. The scaling factor has been considered, which is quantified as the capacity to maintain consistent

[3] https://github.com/esnet/iperf

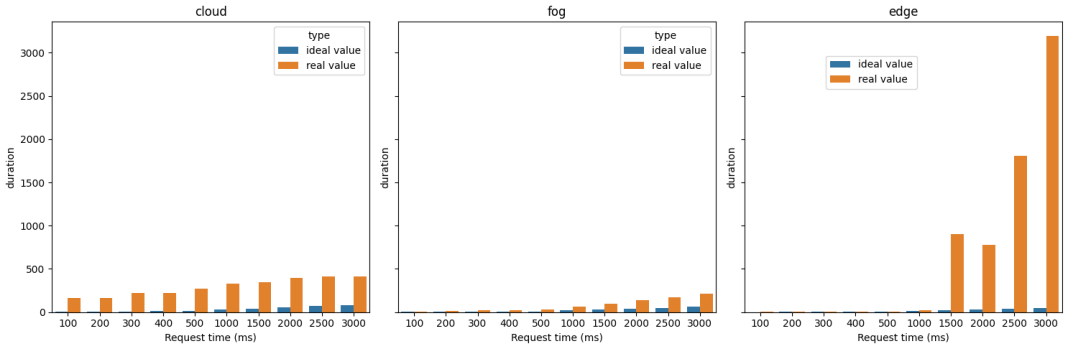
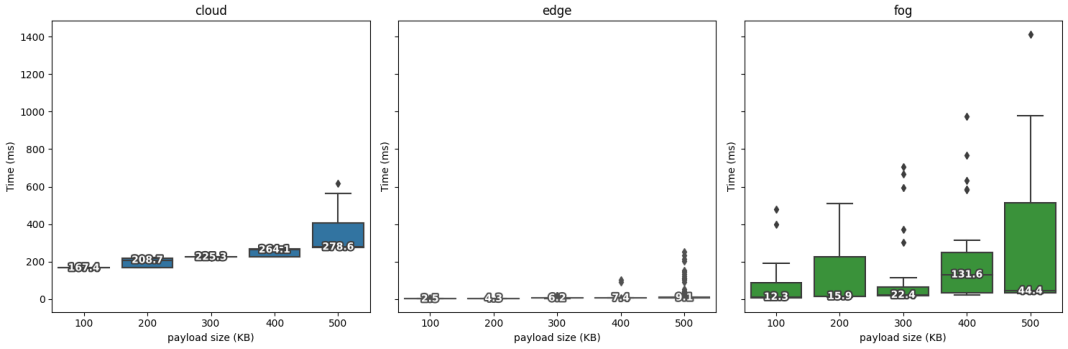
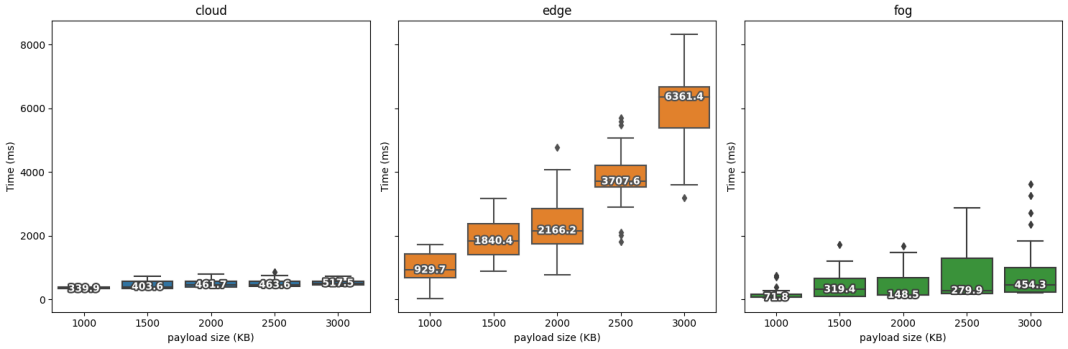


Figure 6: Ideal request duration vs real one on increasing payload



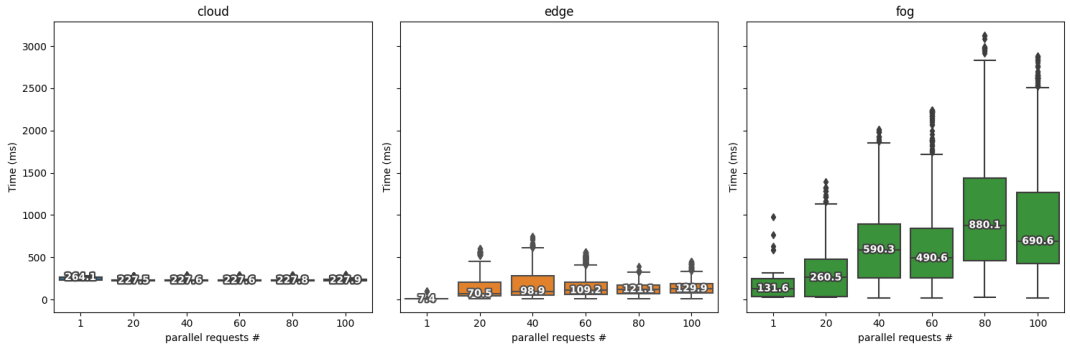
(a) Request Time (RT) single request with small dataset



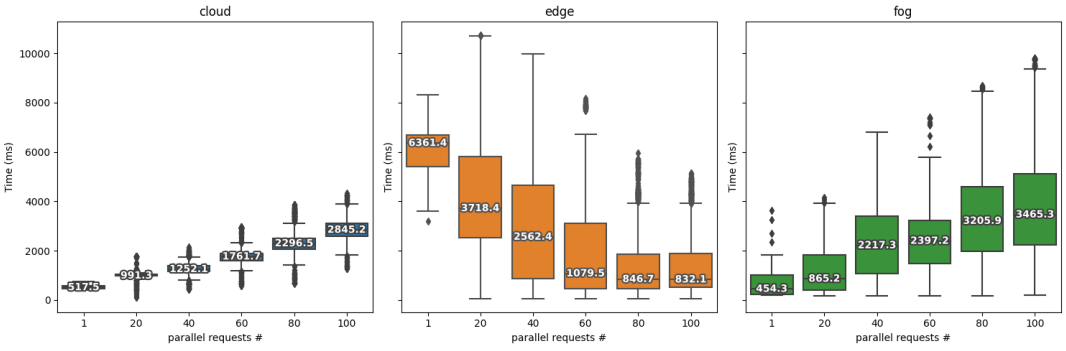
(b) Request Time (RT) single request with medium dataset

Figure 7: RT with one single request at time

performance levels despite a growing need for computational resources or data processing. The findings of our study align with the existing knowledge in the field of continuum computing. We have emphasized the benefits of utilizing edge infrastructures for performing low-demand and small-scale computations. This approach leverages the near-zero network cost while ensuring an optimal balance between computational power and resource use. Cloud computing, as anticipated, has demonstrated superior performance in handling high demand and intensive workloads. All of the aforementioned data can be accessed within the Data Log, rendering them suitable for the formulation of scheduling auto-balanced rules.



(a) Request Time (RT) on increasing requests with 500KB payload



(b) Request Time (RT) on increasing requests with 3MB payload

Figure 8: Request Time (RT) on increasing parallel requests

7 CONCLUSION AND FUTURE WORKS

Previous research has introduced RPulsar as a tool that is native to edge computing. It is utilized for the purpose of designing and implementing event-driven workflows that are centered around the Internet of Things (IoT). This is achieved through the utilization of a robust profile match engine, which facilitates the automatic connection between generic data producers and consumers. Subsequently, in light of the significant increase in popularity of FaaS open-source platforms, our objective was to leverage these platforms for the purpose of designing and deploying continuum native workflows. These workflows would connect FaaS functions through the utilization of a distributed broker known as OpenWolf.

The objective of this study was to modify the adaptability of RPulsar to implicit design workflows, namely by integrating the OpenWolf potentiality to distribute and reuse functions across several Continuum tiers. The construction of a connected distributed Continuum environment was achieved by utilizing the RPulsar core architecture. Subsequently, modifications were made to the structure of the producers and consumers' profiles, enabling the inclusion of a function reference for the purpose of publishing and consuming data. In contrast to generic tasks, FaaS functions possess a distinct task domain, and the ability to anticipate their behavior is facilitated by the availability of a function execution history for querying purposes. As a result, the RPulsar Rule Engine has been enhanced to utilize the execution history as a basis for determining the optimal location for executing a function, hence guaranteeing the desired Quality of Service (QoS).

After the implementation of the aforementioned modifications, a comprehensive evaluation and analysis of the augmented RPulsar software stack were conducted to assess its performance in a dynamic environment subjected to varying levels of system stress. The evaluation was carried out by simulating a smoke detection use case as a representative scenario for Urgent Computing.

Our future endeavors are focused on developing a default automated scheduling engine that determines the optimal location for computation, considering factors such as recent historical data, needed quality of service, and the current system's workload. It is likely that this system would incorporate Federated Learning, which has the potential to leverage a large distributed edge cloud environment [41] similar to the one on which RPulsar is intended to operate.

REFERENCES

[1] Daniel Balouek-Thomert, Eddy Caron, Laurent Lefevre, and Manish Parashar. 2022. Towards a methodology for building dynamic urgent applications on continuum computing platforms. In *2022 First Combined International Workshop on Interactive Urgent Supercomputing (CIW-IUS)*. IEEE, 1–6.

[2] Daniel Balouek-Thomert, Ismael Perez, Sam D Faulstich, Heather A Holmes, Ilkay Altintas, and Manish Parashar. 2023. Keynote Talk: Leveraging the Edge-Cloud Continuum to Manage the Impact of Wildfires on Air Quality. In *2023 IEEE International Conference on Pervasive Computing and Communications Workshops*

and other Affiliated Events (PerCom Workshops). IEEE, 27–31.
[3] Daniel Balouek-Thomert, Ivan Rodero, and Manish Parashar. 2020. Harnessing the Computing Continuum for Urgent Science. *SIGMETRICS Perform. Eval. Rev.* 48, 2 (nov 2020), 41–46. https://doi.org/10.1145/3439602.3439618
[4] Luiz Bittencourt, Roger Immich, Rizos Sakellariou, Nelson Fonseca, Edmundo Madeira, Marilia Curado, Leandro Villas, Luiz DaSilva, Craig Lee, and Omer Rana. 2018. The Internet of Things, Fog and Cloud continuum: Integration and challenges. *Internet of Things (Netherlands)* 3-4 (2018), 134–155. https://doi.org/10.1016/j.iot.2018.09.005 arXiv:1809.09972
[5] Victor Casamayor Pujol, Andrea Morichetta, Ilir Murturi, Praveen Kumar Donta, and Schahram Dustdar. 2023. Fundamental Research Challenges for Distributed Computing Continuum Systems. *Information* 14, 3 (2023). https://www.mdpi.com/2078-2489/14/3/198
[6] Parag Chatterjee, Leandro J. Cymberknop, and Ricardo L. Armentano. 2017. IoT-based decision support system for intelligent healthcare — applied to cardiovascular diseases. In *2017 7th International Conference on Communication Systems and Network Technologies (CSNT)*, Vol. 29. 362–366. https://doi.org/10.1109/CSNT.2017.8418567
[7] Bin Cheng, Jonathan Fuerst, Gurkan Solmaz, and Takuya Sanada. 2019. Fog Function: Serverless Fog Computing for Data Intensive IoT Services. In *2019 IEEE International Conference on Services Computing (SCC)*. 28–35. https://doi.org/10.1109/SCC.2019.00018
[8] Bin Cheng, Gürkan Solmaz, Flavio Cirillo, Ernö Kovacs, Kazuyuki Terasawa, and Atsushi Kitazawa. 2018. FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities. *IEEE Internet of Things Journal* 5, 2 (2018), 696–707. https://doi.org/10.1109/JIOT.2017.2747214
[9] Ivan Cilic, Ivana Podnar Zarko, and Mario Kusek. 2021. Towards service orchestration for the cloud-to-thing continuum. *2021 6th International Conference on Smart and Sustainable Technologies, SpliTech 2021* (2021). https://doi.org/10.23919/SpliTech52315.2021.9566410
[10] Schahram Dustdar. 2022. Distributed Computing Continuum Systems. In *2022 IEEE International Conference on Services Computing (SCC)*. 356–356. https://doi.org/10.1109/SCC55611.2022.00060
[11] Schahram Dustdar, Victor Casamajor Pujol, and Praveen Kumar Donta. 2022. On distributed computing continuum systems. *IEEE Transactions on Knowledge and Data Engineering* XX (2022), 1–14. Issue X. https://doi.org/10.1109/TKDE.2022.3142856
[12] Nicolas Ferry, Rustem Dautov, and Hui Song. 2022. 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)*. 851–858. https://doi.org/10.1109/CCGrid54584.2022.00101
[13] Gordon Gibb, Rupert Nash, Nick Brown, and Bianca Prodan. 2019. The Technologies Required for Fusing HPC and Real-Time Data to Support Urgent Computing. In *2019 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*. 24–34. https://doi.org/10.1109/UrgentHPC49580.2019.00009
[14] Google. 1999. Google Cloud Functions. https://cloud.google.com/functions
[15] IBM. 2023. IBM Serverless Functions. https://www.ibm.com/cloud/functions
[16] Runyu Jin and Qirui Yang. 2022. EdgeFaaS: A Function-based Framework for Edge Computing. arXiv:2210.01410 [cs.DC]
[17] Albert Jonathan, Abhishek Chandra, and Jon Weissman. 2016. Awan: Locality-Aware Resource Manager for Geo-Distributed Data-Intensive Applications. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*. 32–41. https://doi.org/10.1109/IC2E.2016.15
[18] Dragi Kimovski, Christian Bauer, Narges Mehran, and Radu Prodan. 2022. Big Data Pipeline Scheduling and Adaptation on the Computing Continuum. In *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. 1153–1158. https://doi.org/10.1109/COMPSAC54236.2022.00181
[19] knative. 2023. Knative. https://knative.dev/
[20] Sergey V. Kovalchuk, Pavel A. Smirnov, Sergey V. Maryin, Timofey N. Tchurov, and Vladislav A. Karbovskiy. 2013. Deadline-driven Resource Management within Urgent Computing Cyberinfrastructure. *Procedia Computer Science* 18 (2013), 2203–2212. https://doi.org/10.1016/j.procs.2013.05.391 2013 International Conference on Computational Science.
[21] A. Luckow, K. Rattan, and S. Jha. 2021. Pilot-Edge: Distributed Resource Management Along the Edge-to-Cloud Continuum. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE Computer Society, Los Alamitos, CA, USA, 874–878. https://doi.org/10.1109/IPDPSW52791.2021.00130
[22] Jacopo Massa, Stefano Forti, and Antonio Brogi. 2022. Data-Aware Service Placement in the Cloud-IoT Continuum. In *Service-Oriented Computing*, Johanna Barzen, Frank Leymann, and Schahram Dustdar (Eds.). Springer International Publishing, Cham, 139–158.
[23] Gabriele Proietti Mattia and Roberto Beraldi. 2021. Leveraging Reinforcement Learning for online scheduling of real-time tasks in the Edge/Fog-to-Cloud computing continuum. In *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*. 1–9. https://doi.org/10.1109/NCA53618.2021.9685413
[24] Gabriele Morabito, Christian Sicari, Armando Ruggeri, Antonio Celesti, and Lorenzo Carnevale. 2023. Secure-by-design serverless workflows on the Edge–Cloud Continuum through the Osmotic Computing paradigm. *Internet of Things* 22 (2023), 100737. https://doi.org/10.1016/j.iot.2023.100737
[25] OpenFaas. 2023. OpenFaas. https://www.openfaas.com/
[26] Tobias Pfandzelter and David Bermbach. 2020. tinyFaaS: A Lightweight FaaS Platform for Edge Environments. In *2020 IEEE International Conference on Fog Computing (ICFC)*. 17–24. https://doi.org/10.1109/ICFC49376.2020.00011
[27] B. Przybylski, P. Zuk, and K. Rzadca. 2021. Data-driven scheduling in serverless computing to reduce response time. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE Computer Society, Los Alamitos, CA, USA, 206–216. https://doi.org/10.1109/CCGrid51090.2021.00030
[28] Kaustubh Rajendra Rajput, Chinmay Dilip Kulkarni, Byungjin Cho, Wei Wang, and In Kee Kim. 2022. EdgeFaaSBench: Benchmarking Edge Devices Using Serverless Computing. In *2022 IEEE International Conference on Edge Computing and Communications (EDGE)*. 93–103. https://doi.org/10.1109/EDGE55608.2022.00024
[29] Eduard Gibert Renart, Daniel Balouek-Thomert, and Manish Parashar. 2019. An Edge-Based Framework for Enabling Data-Driven Pipelines for IoT Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 885–894. https://doi.org/10.1109/IPDPSW.2019.00146
[30] Fabiana Rossi, Valeria Cardellini, Francesco Lo Presti, and Matteo Nardelli. 2020. Geo-distributed efficient deployment of containers with Kubernetes. *Computer Communications* 159 (2020), 161–174. https://doi.org/10.1016/j.comcom.2020.04.061
[31] Fabiana Rossi, Simone Falvo, and Valeria Cardellini. 2021. GOFS: Geo-distributed Scheduling in OpenFaaS. In *2021 IEEE Symposium on Computers and Communications (ISCC)*. 1–6. https://doi.org/10.1109/ISCC53001.2021.9631492
[32] K R Sheshadri and J Lakshmi. 2022. QoS aware FaaS for Heterogeneous Edge-Cloud continuum. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. 70–80. https://doi.org/10.1109/CLOUD55607.2022.00023
[33] Christian Sicari, Lorenzo Carnevale, Antonino Galletta, and Massimo Villari. 2022. OpenWolf: A Serverless Workflow Engine for Native Cloud-Edge Continuum. In *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. 1–8. https://doi.org/10.1109/DASC/PiCom/CBDCom/Cy55231.2022.9927926
[34] Christian Sicari, Antonino Galletta, Antonio Celesti, Maria Fazio, and Massimo Villari. 2021. An Osmotic Computing Enabled Domain Naming System (OCE-DNS) for distributed service relocation between cloud and edge. *Computers & Electrical Engineering* 96 (2021), 107578. https://doi.org/10.1016/j.compeleceng.2021.107578
[35] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. 2018. Industrial Internet of Things: Challenges, Opportunities, and Directions. *IEEE Transactions on Industrial Informatics* 14, 11 (2018), 4724–4734. https://doi.org/10.1109/TII.2018.2852491
[36] Antero Taivalsaari, Tommi Mikkonen, and Cesare Pautasso. 2022. Towards Seamless IoT Device-Edge-Cloud Continuum:. In *ICWE 2021 Workshops*, Maxim Bakaev, In-Young Ko, Michael Mrissa, Cesare Pautasso, and Abhishek Srivastava (Eds.). Springer International Publishing, Cham, 82–98.
[37] Domenico Talia and Paolo Trunfio. 2023. Urgent Computing for Protecting People From Natural Disasters. *Computer* 56, 4 (2023), 131–134. https://doi.org/10.1109/MC.2023.3241733
[38] Yang Tang and Junfeng Yang. 2020. Lambdata: Optimizing Serverless Computing by Making Data Intents Explicit. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. 294–303. https://doi.org/10.1109/CLOUD49709.2020.00049
[39] Achilleas Tzenetopoulos, Evangelos Apostolakis, Aphrodite Tzomaka, Christos Papakostopoulos, Konstantinos Stavrakakis, Manolis Katsaragakis, Ioannis Oroutzoglou, Dimosthenis Masouros, Sotirios Xydis, and Dimitrios Soudris. 2021. FaaS and Curious: Performance Implications of Serverless Functions on Edge Computing Platforms. In *High Performance Computing*, Heike Jagode, Hartwig Anzt, Hatem Ltaief, and Piotr Luszczek (Eds.). Springer International Publishing, Cham, 428–438.
[40] Jiachen Wang, Ji Ma, Kangping Hu, Zheng Zhou, Hui Zhang, Xiao Xie, and Yingcai Wu. 2023. Tac-Trainer: A Visual Analytics System for IoT-based Racket Sports Training. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2023), 951–961. https://doi.org/10.1109/TVCG.2022.3209352
[41] Qi Xia, Winson Ye, Zeyi Tao, Jindi Wu, and Qun Li. 2021. A survey of federated learning for edge computing: Research problems and solutions. *High-Confidence Computing* 1, 1 (2021), 100008. https://doi.org/10.1016/j.hcc.2021.100008
[42] Preeti Yadav and Sandeep Vishwakarma. 2018. Application of Internet of Things and Big Data towards a Smart City. In *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, Vol. 29. 1–5. https://doi.org/10.1109/IoT-SIU.2018.8519920