Transactions on Networks and Communications - Vol. 9, No. 5
Publication Date: October, 25, 2021
DOI: 10.14738/tnc.95.11231.
Kjorveziroski, V., Canto, C. B., Roig, P J., Gilly, K., Mishev, A., Trajkovik, V., Filiposka, S. (2021). IoT Serverless Computing at the Edge: Open Issues and Research Direction. Transactions on Networks and Communicaitons, 9(5). 1-33.

IoT Serverless Computing at the Edge: Open Issues and Research Direction

Vojdan Kjorveziroski
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University, Skopje, North Macedonia

Cristina Bernad Canto
Department of Computer Engineering
Miguel Hernandez University of Elche, Elche, Spain

Pedro Juan Roig
Department of Computer Engineering
Miguel Hernandez University of Elche, Elche, Spain

Katja Gilly
Department of Computer Engineering
Miguel Hernandez University of Elche, Elche, Spain

Anastas Mishev
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University, Skopje, North Macedonia

Vladimir Trajkovik
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University, Skopje, North Macedonia

Sonja Filiposka
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University, Skopje, North Macedonia

ABSTRACT

Novel computing paradigms aim to enable better hardware utilization, allowing a greater number of applications to be executed on the same physical resources. Serverless computing is one example of such an emerging paradigm, enabling faster development, more efficient resource usage, as well as no requirements for infrastructure management by end users. Recently, efforts have been made to utilize serverless computing at the network edge, primarily focusing on supporting Internet of Things (IoT) workloads. This study explores open issues, outlines current progress, and summarizes existing research findings about serverless edge computing for IoT by analyzing 67 relevant papers published between 01.01.2015 and 01.09.2021. We discuss the state-of-the-art research in 8 subject areas relevant to the use of serverless at the network edge, derived through the analysis of the selected articles. Results show that even though there is a noticeable interest for

this topic, further work is needed to adapt serverless to the resource constrained environment of the edge.

Keywords: Edge Computing; Cloud Computing; Function as a Service; Serverless Computing; Internet of Things; Review.

INTRODUCTION

The cloud revolution has introduced the concept of * (anything) as a service [1], an abstraction allowing users to think of computing infrastructure and software in general as just another utility for which they pay monthly expenditures. While from the user perspective the Software as a Service (SaaS) offerings unburden them from thinking about new features, upgrades, and even security patches, developers still have to interact with lower-level abstractions. One of the most popular developer-oriented products is Platform as a Service (PaaS), allowing them to more easily publish and host applications. However, not all infrastructure related aspects are abstracted to the desirable extent and the underlying programming models have not changed; developers still need to provide a complete software package and decide what additional components they would like to use, such as the database product, messages queues, or caching systems.

The recent introduction of the serverless computing concept, comprised of Function as a Service (FaaS) and Backend as a Service (BaaS) aims to alleviate these developer problems allowing them to focus just on the core functionality of their product. Despite its name, there are still servers involved in this paradigm, but they are not the concern of the developers, since the provider deals with the scaling, deployment, and runtime configuration of not only the developer's code but the associated backend services such as databases as well. The developer simply provides the core functionality in terms of the necessary function code (Function as a Service) and achieves statefulness, caching, user registration, and all other backend related services by utilizing other managed products by the provider (Backend as a Service) [2].

Even though the first commercial serverless product has been published in 2015 [3] as a cloud service aimed primarily at web developers, quickly alternative use-cases have been identified as well. With the meteoric rise in the number of internet of things (IoT) devices, there is an ever-growing need for increased compute and network capacity, as well as new product features which would enable new usage scenarios. To lower the communication latency with infrastructure located in the cloud, the trend of edge computing has emerged, moving computing capacity closer to the data source, reducing delays. However, from the developers' perspective, the programming practices have not changed, with a major difference being that they also need to account for the more limited computing capacity when their applications are deployed at the edge compared to the cloud.

In recent years the idea of utilizing serverless computing at the network edge has emerged, thus allowing the deployment of lightweight functions across the infrastructure. Existing cloud providers have quickly adapted their product portfolios [4], [5], allowing users to self-host the serverless runtimes on their own hardware at the network edge. A number of existing open-source serverless projects have also published more lightweight versions of their products [6], [7] and completely new platforms have been proposed as well [8]. Attempts to transparently

bridge the divide between the edge and cloud by allowing cross-compatibility of the developed functions have also been made, thus offering an edge-cloud continuum.

Although serverless edge computing is a new and dynamic research field with great potential for event driven IoT workloads [9], there are a number of open challenges that hinder a wider adoption. The aim of this paper is to describe these issues and outline recent efforts aimed at solving them. To this effect, we have conducted a systematic analysis of 67 state-of-the-art research papers published between 1st of January 2015 and 1st of September 2021, with the intention of identifying the current research trends and open issues.

The rest of this paper is organized as follows: in section 0 we analyze related research to IoT serverless computing at the network edge, and then continue to section 0 where we outline the methodology for our survey. In section 0 we describe the identified open issues and discuss in detail the relevant state-of-the-art research aimed at solving them. We conclude the paper with section 0, focusing on next steps and future research directions.

Related Research Papers for Serverless IoT

Both serverless and IoT are active research topics, gathering sizeable interest from the community because of the wide-ranging effects that they might have on people's everyday lives. The full extent of IoT applications ranges from smart devices at home, to increased manufacturing efficiency, and improvements to quality of life through better environmental predictions and early warning systems. Combining this with the efficiency and easy scalability of the serverless paradigm, augmented by the greatly reduced development effort and shorter time to market, makes IoT and serverless an appropriate match for accommodating the expected influx of new IoT devices.

These aspects are contributing factors to the existence of numerous review papers describing open-research problems and disseminating existing findings either for serverless or IoT, but, to the best of our knowledge, no paper focusing explicitly on both of those aspects has been published yet. The aim of this review is to fill that gap and look at the serverless research challenges associated when it is used primarily for event based IoT workloads, especially when such workloads require low latency, and need to be performed at the network edge.

A common theme across existing literature is the consensus that the usage of serverless is expected to skyrocket in the near future [10], [11], as solutions to the open problems are emerging. Perhaps one of the more pressing issues for serverless computing is the difference in the various implementations regarding the runtime environments and the variable performance that they are offering as a result of the chosen architecture for function execution, such as virtual machines, containers, or native execution [12], [13]. The authors of [12] propose unikernels as a possible workaround for this problem, allowing a function to be packaged with all of its library dependencies and hardware interaction frameworks, omitting requirements for a base operating system or a hypervisor.

The introduced abstraction of BaaS, and its externalization in terms of the running functions is identified as another area for improvement, resulting in poor performance for I/O bound workloads that need to communicate with fast storage, which in this case is accessed through network APIs, adding overhead and latency, leading to longer execution times [12], [14], [15].

This increase in execution time can potentially cause function timeouts, since depending on the platform, function execution time can be severely limited [12]. Another concern is also the increased cost that might be incurred, as a result of this slowdown, even though serverless is deemed a more cost-efficient solution that other alternatives that do not offer a scale-to-zero capability [15].

One way in which cost can be reduced is by migrating from a commercial serverless platform to a self-hosted one, potentially avoiding the vendor lock-in commonly associated with such public platforms [16]. Further research is required in this area, to determine common patterns for the initial creation of serverless applications or migration of existing cloud-based applications to a serverless architecture [15], [16]. However, the very nature of granular functions with a well-defined functionality might be of great benefit to such efforts, since existing code can be reused in various other systems, promoting composition and the mash-up development paradigm. Such ideas will likely result in the establishment of many function marketplaces where developers will be able to either share or sell their functions [13], [15]. Of course, further work on vendor abstraction and cross-platform serverless compatibility is needed before such community sharing efforts can be realized.

Reusing functions from unknown developers naturally opens the question of security. Serverless computing differs from regular cloud computing based on VMs or even physical servers, since the security aspects are in most cases handled by the provider, which is especially true in public environments, limiting the customizability of security policies by end users [12], [14], [15]. When devising new runtime optimizations, great care should be taken not only on function performance, but on function isolation as well, keeping in mind that serverless platforms are multi-tenant environments.

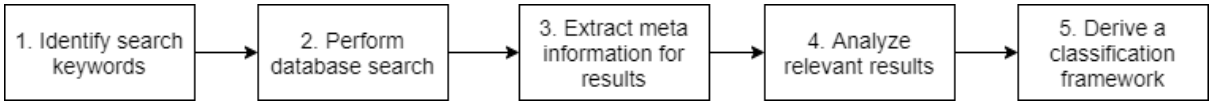

Figure 1. High-level Overview of the Applied Research Method

Despite these challenges, serverless is considered one of the enabling aspects of future IoT infrastructures [13], [14], [17]. By moving serverless computing to the edge of the network, and establishing an edge-cloud continuum, application developers can get the best of both the edge and cloud paradigms. Through the utilization of advanced scheduling strategies for selecting the location where the workload will be executed, either the potentially limitless computing capacity of the cloud can be leveraged, or the low latency of the edge, depending on the context. Distributed scheduling algorithms [18] play a very prominent role in such edge-cloud scenarios, where the lower execution performance of the edge can easily offset the time savings achieved by eliminating the transfer delay to the more resource rich cloud, leading to suboptimal implementations. This requires careful consideration of the scale-down-to-zero behavior which although results in lowered cost, increases the initial start-up time of the first function instance.

RESEARCH METHOD

To obtain broad initial results we have identified the following terms for our database search: (“serverless” ∨ “faas” ∨ “function as a service” ∨ “function-as-a-service” ∨ “baas” ∨ “backend as

a service" ∨ "backend-as-a-service") ∧ ("iot" ∨ "internet of things" ∨ "internet-of-thigs"). Even though our main focus are serverless challenges at the network edge, we have purposefully omitted mention of either *edge* or *edge computing* in the search query, so that even papers who do not explicitly mention them but do mention serverless in an IoT context are returned. Authors often discuss serverless issues and present new approaches in a wider network context without explicitly targeting or even mentioning the network edge, but in many cases these contributions are indeed applicable to it and thus provide valuable insight.

Using the previously identified search keywords we have used six popular databases: IEEE Xplore, ACM, Arxiv, Google Scholar, Springer, and Science Direct to identify relevant papers. The criteria for consideration of any paper were: i) published between 01.01.2015 and 01.09.2021; ii) full-text available to the authors of this paper in English; iii) contains clear reference to IoT serverless computing and discusses issues directly or indirectly applicable in an edge context; iv) is peer reviewed, grey literature was omitted.

We have initially obtained 217 results, which after applying the above inclusion criteria were reduced to 64 which underwent a full-text reading. During this phase we have actively searched for additional cited relevant papers. As a result of the full text reading, we have identified 11 papers that were not relevant to the topic and added 14 extra entries as a result of the snowballing technique. At the end, we have accepted 67 papers in total, based on which we have derived our classification framework presented in the next section. Figure 1 shows a graphical representation of our workflow.

Open Issues and Existing Research Findings

In the subsections that follow we discuss the relevant open issues identified through the full-text analysis of the selected papers and describe potential solutions and mitigations. We follow up on the topic categorization that was performed in [19] which represented an exploratory systematic mapping of serverless trends at the network edge. In this paper we provide an in-depth discussion for each identified topic of interest and relevant subtopics, outlining the current state-of-the-art research in this emerging field.

Efficiency

The fact that at the lowest level serverless architectures are a time-sharing concept, efficiency optimizations are at the heart of this paradigm, since any breakthroughs directly impact both the service providers, as well as the customers. Reducing the functions' footprint and execution times allows more workload to be executed on the same infrastructure, as well as lowering costs, thus promoting serverless as a feasible alternative for latency constrained applications.
The set of requirements that a serverless platform, especially one deployable to the network edge, should support can be summarized with the following characteristics [20]: i) provide an event-driven and short-lived execution of serverless functions; ii) offer support for high density and multi-tenancy; iii) provide low latency as to support the requirements of IoT workloads; iv) deal with high churn of serverless functions, executed repeatedly, with a short-lived execution time.

The main problem in this area that has attracted a significant interest both from academia and the industry is how to reduce the initial start-up time of a serverless function. This is also popularly known as the cold start problem. We elaborate further on this issue in the subsection

that follows. We then proceed to introduce the various proposed solutions to this problem, starting with WebAssembly, and moving onto other runtime environments such as micro virtual machines and unikernels. We conclude the discussion on efficiency and optimization with a description of recent efforts to reduce serverless functions' sizes. Table 1 provides an overview of the discussed topics in each paper relevant to this section.

Table 1. Overview of Research Topics and Related Papers from an Efficiency Perspective

Topic	Papers	Total
Footprint Reduction	[21]–[27]	7
Containers	[12], [26], [28]	3
WebAssembly	[20]–[22]	3
Native Execution	[12], [27]	2
Micro VMs	[12]	1
SDN	[29]	1
Unikernels	[12]	1
VMs	[12]	1

The Cold Start Problem and Container Pre-Warming

The cold start problem is a direct consequence of the scale-down-to-zero approach, which keeps a function instance running for a limited amount of time, until it finishes its execution, or it reaches the timeout limit, after which it is terminated. Upon the next invocation, another instance needs to be created, incurring larger start up costs in comparison to the scenario where the same runtime environment is reused [9], [21]. To mitigate this drawback, numerous approaches have been proposed. One of them is to keep a pool of pre-warmed containers, ready to serve a given function without incurring the cold start delay during the first call [22], [30]. Even though this is a possible solution to the problem, it leads to greater resource usage, and eliminates one of the positive aspects of serverless - the scale-down-to-zero feature. A slight variation to this solution is not discarding the runtime environment after a function is done executing, and instead reusing it for any future invocations, a strategy practiced by many serverless platforms, both commercial and open-source. However, in this way runtime isolation is not guaranteed, leading to security risks in which temporary data from a previous invocation by a different user can be accessed by a future invocation of the function, in cases where the same environment is used.

The drawbacks of the aforementioned solutions have sparked the interest of migrating away from containers for serverless function execution and onto more efficient runtimes. Examples include WebAssembly, micro virtual machines, and unikernels, among others. The purpose of these efforts is not only to reduce start up times, but also achieve performance comparable to the native execution on bare-metal hardware.

WebAssembly

One example of a more efficient runtime for serverless functions that looks promising and attracts a noticeable research interest is WebAssembly. Even though first envisioned as a web technology, bearing this mark even in its name, it has since been adapted to a general-purpose

execution environment, not limited to the browser. A number of open-source WebAssembly engines can be directly embedded into applications. Hall et al. [21] take this approach where they implement a serverless platform inspired by the OpenWhisk architecture, but unlike OpenWhisk, they use WebAssembly as an execution environment. Their contribution also characterizes function access patterns into three distinct categories: i) single client, multiple access; multiple clients, single access; multiple clients, multiple access; and then provide benchmarks comparing the performance of the WebAssembly prototype to OpenWhisk. Results show that while the startup time is significantly reduced, execution time is increased, and it is slower than both native and container-based execution.

A possible solution to the reduced performance is provided by Gadepalli et al. first in [22], where a prototype implementation is discussed and later in [20] where a functional platform [31] is presented. Their approach is different since they are not using a JavaScript virtual machine as the execution environment for the WebAssembly serverless functions, because even though it does offer better start up times compared to containerization, it reduces execution performance, as seen in [21]. Instead, they develop a new WebAssembly compiler, and a runtime framework called aWsm [32], allowing multiple functions to share a single runtime instance, and bypass the kernel scheduling and isolation features, instead relying on its own implementations. This approach is reported to offer start up times measured in microseconds and function memory footprint in kilobytes. Results in [20] where the Sledge platform is introduced, utilizing the described approach, show 4 times better latency and throughput comparable to Nuclio [33], an alternative open-source serverless platform, commonly praised for its speed.

To conclude, even though WebAssembly can be seen as a potential solution to the cold start problem, several issues remain before it can satisfy all the requirements of an efficient serverless runtime at the edge. Platforms utilizing it require careful consideration regarding the execution performance, as not to offset any benefits acquired through the reduced start up times. Furthermore, with the increased popularity of WebAssembly for other use-cases besides serverless, it is expected that there will be even wider and better support by high-level programming languages, giving developers a wider set of options.

Micro Virtual Machines

Micro virtual machines can also be seen as a potential solution to the slow start-up speed of regular virtual machines and the lower isolation provided by containers compared to the traditional VM approach. The offered benefits in terms of hardware-backed separation between workloads and milliseconds start-up times can benefit serverless use-cases as well, providing greater isolation between tenants and wider choice in terms of supported underlying platforms. The first commercial services using this technology both in a serverless and serverful context are already available [34]. Amazon Firecracker is one such infrastructure product whose micro VM implementation has also been open-sourced [35].

The application of micro virtual machines is still an area under active research, with interest from both industry and academia, and the potential benefits both to execution speed and workload isolation can be significant [36].

Unikernels

Unikernels also present an alternative to the containerized execution of serverless workloads, offering reduced overhead in comparison to the other approaches. The idea behind unikernels is to package the application together with all its dependencies, as well as system functions for directly interfacing with the underlying hardware, so that it can be instantiated independently, without the need for a base operating system. In this way, the resulting artifact size is an order of magnitude smaller than traditional virtual machine images, while allowing native execution performance, leading to more efficient resource usage and planning.

The authors of [12] identify unikernels as a potential approach that might be greatly utilized in the future for FaaS execution. While there are some existing unikernel implementations today [37] this still remains an under-researched topic.

Code Size Reduction

The function size plays a significant role in the cold start up times of serverless functions, especially on resource constrained edge devices. Additionally, most commercial serverless providers charge for the storage of the function's source code as well, which can reach hundreds of megabytes when all library dependencies are taken into account. Even in the case where serverless platforms run on private infrastructures, the excessive storage use of large numbers of functions cannot be overlooked and pose a challenge for computing devices, even more so to those with capacity and performance constrained storage. To overcome this problem, several optimization techniques have been proposed [23], [24], the main idea being to prevent function size from ballooning in cases where a large library is added as a dependency. This is especially important during the migration of existing workloads to a serverless architecture, where prior technical debt might lead to the reuse of well-tested, albeit not serverless optimized libraries. More details regarding the possible optimizations performed on existing codebases to reduce their size and avoid a full rewrite are given in subsection 0, Serverless Migration Guidelines and Benefits, below.

Scheduling

Taking into account the distributed nature of the edge, efficient scheduling algorithms are required to determine the best execution point for a given serverless function. However, as a result of the large number of functions that might potentially be available, additional attention should be paid to the limited resources of the edge devices, which might limit how many functions can be served at a given point in time, due to storage and performance constraints. The main efforts in this area are focused on developing efficient scheduling algorithms that can optimize latency, cost, or bandwidth usage, and help establish an edge-fog-cloud continuum, automatically finding the best execution location that matches the previously defined criteria or even transparently migrating in-progress workloads. There are examples that apply new scheduling approaches to commercial services [24], [25], as well as to open-source and self-hosted ones [38]–[40]. Table 2 categorizes the relevant papers to this topic according to the discussion provided in the subsections below.

Table 2. Overview of Research Topics and Related Papers from a Scheduling Perspective

Topic	Papers	Total
Latency Optimization	[18], [20], [22], [24], [26], [29], [38]–[53]	22
Bandwidth Optimization	[18], [29], [40], [48], [51], [54]	6
AI & ML	[26], [39], [41]–[43]	5
Container Prewarming	[26], [30], [39], [41], [55]	5
Price Optimization	[18], [25], [29], [43], [50]	5
On-the-fly Migration	[18], [44], [54]	3

Optimization of Commercial Serverless Platforms

Elgamal et al. [25] present an algorithm aimed at the AWS serverless portfolio, capable of scheduling functions either on the edge, on devices utilizing AWS Greengrass, or in the cloud, using the AWS Lambda service. The fact that functions can be seamlessly reused across these two services, without any modification, allows the scheduling system to see the edge as an extension to the cloud. The presented approach is modeled as a constrained shortest path problem, with the aim of finding the minimum execution cost, without going over the predefined latency threshold. Further cost savings can be achieved during cloud execution by combining the logic of multiple distinct functions into a single one, eliminating the need for function chaining, which is charged extra. Pelle et al. [24] also implement an abstraction layer between Greengrass and AWS Lambda, creating a decision system about which infrastructure to use and where to execute the functions, incorporating metrics from commercial and on-premise SDN networking devices.

A similar approach to the previously described system by Elgamal et al. in [25] is taken by Pelle et al. in [24], with the focus on optimizing the deployed function layout through function grouping and resource requirement optimization, achieving better performance and lower cost. The proposed solution uses a two-layer architecture, where the first layer can generate a software layout which describes how the different functions will be grouped, what will be their resource requirements, and whether they will be executed at the edge or in the cloud, all based on previous metrics and defined constraints. The second layer acts as an adapter layer and is responsible for the deployment of the generated layout on the used infrastructure. Even though in the initial prototype only AWS services are supported, the abstraction provided by this second layer allows other serverless products to be supported in the future as well, which would allow function scheduling across different providers, selecting the one that offers the most favorable terms.

Optimization of Open-Source Serverless Platforms

Unlike contributions that focus on commercial platforms, those using open-source software meant to be run and maintained independently focus on latency as the primary optimization goal. Cicconetti et al. [38] present a scheduling algorithm for serverless functions that can select the most optimal node from a list of distributed nodes, all offering the same function instance. Utilizing a software defined network (SDN) architecture, and assigning different roles to edge devices, a hierarchical decision system is implemented, allowing to choose the device offering the most optimal execution latency. The prototype implementation is developed on top of the OpenWhisk platform, and the source code is publicly available [56].

Another approach focusing on an existing open-source serverless platform is presented in [39], using Knative. By using linear regression models to predict when requests for a particular function will be received, as well as their volume, a prediction system is built. The idea is that by using prefetching and traffic prediction, the cold start delay penalty incurred during the initial start-up of the container hosting the given serverless function can be mitigated, as a result of pre-warming the required number of container instances. Similar prediction systems whose aim is reducing the cold start penalty have been built for different open-source serverless platforms commonly deployed at the network edge as well. Agarwal et al. [41] design a reinforcement learning scheduling algorithm for Kubeless, optimizing the number of container replicas backing a given serverless function depending on user demand, introducing it as an alternative to the native Kubernetes auto-scaler which is also being utilized by Kubeless itself. Continuing the trend of offering better scheduling algorithms to existing serverless platforms, the authors of [26] have recently developed a queuing theory algorithm for function scheduling on top of OpenWhisk. Their implementation also offers runtime optimizations by limiting the allocated CPU shares to already executing functions, on-the-fly.

Smart Scheduling Algorithms for the Network Edge

This idea of using prediction models is shared by Patman et al. [42], where a heuristic based machine learning algorithm is presented to dynamically decide what combinations of functions needs to be deployed where, in addition to when, so that serverless tasks can be efficiently scheduled. However, their vision is to reuse dormant computing capacity of nearby devices for the task execution instead of offloading it to a dedicated edge infrastructure.

Cho et al. [43] also apply machine learning to scheduling algorithms but this time in a multi-access edge computing (MEC) context, where the initial tier of the serverless infrastructure is collocated with mobile base stations. Using a deep reinforcement learning approach, workloads are distributed among edge and remote cloud nodes based on various execution characteristics, including QoS, cost, and performance requirements. This trend of collocating compute infrastructure with mobile base stations, following the MEC specification and exploring efficient algorithms of choosing the best node is also explored in [40]. The parameter being optimized by Cicconetti et al., in this case, is the total delay for completing the function, taking into account transmission as well as execution delays. However, another constraint is added to the decision process as well since the algorithm is made aware of potential specialized hardware such as GPUs that might be available at specific collocated nodes.

On-the-fly Migration

Even though there are multiple scheduling techniques that consider various factors during the decision-making process such as latency, cost, execution speed, or available hardware, they mainly focus on the initial function placement. Karhula et al. [44] present a way in which a running serverless function can be migrated to a different node, while it is being executed. In this manner, it is possible to better balance the workload and alleviate memory pressure by migrating function instances on-the-fly, instead of prematurely terminating their execution in resource constrained scenarios. Also, looking at the problem from a different perspective, this also provides the means for temporarily pausing any function which is currently blocked and waits for another function or an I/O operation to complete. This approach has benefits both for customers and service providers, reducing execution cost, as well as offering better time sharing of the underlying hardware. The potential cost reduction could be significant, and it can

also help to overcome the double-spending serverless problem, where a user is charged for the execution time of a function which is idling and is blocked by a synchronous call to another function, waiting for it to return a result.

IoT Serverless Edge Platforms

Table 3. Serverless Platforms Specifications

Name	Location	Runtime	Function Languages	Base	Open-Source
STOIC [45]	Any	Native & Containers	Any	Kubeless	✓ [57]
Serverless IoT [58]	Any	Containers	Any	OpenFaaS	✓ [59]
Pigeon [30]	Any	Containers	Dockerfile	Kubernetes	✕
Fog Function [54]	Any	Containers	Dockerfile	FogFlow	✓ [60]
[55]	Any	Containers	Any	OpenWhisk	✕
CSPOT [61]	Any	Containers	C, Python	Docker	✓ [62]
A3-E [46]	Any	Containers	Depends on base	OpenWhisk, AWS	✓ [63]
[64]	Any	Containers	Any	Kubernetes, AWS Lambda	✓ [65]–[67]
Clemmys [28]	Any	Containers	Any	OpenWhisk	✕
Hcloud [50]	Any	Proprietary Platforms, Containers	Python	IaaS & Commercial Serverless	✕
[47]	Edge, Cloud	Containers	Any	Edge Devices	✓ [68]
[69]	Edge, Cloud	Native & Containers	Python, Node.js, Java, C, C++	AWS Greengrass & Lambda	✕
[70]	Edge, Cloud	Containers	N/A	LXC	✕
EBI-PAI [48]	Edge	Containers	Any	OpenWhisk	✕
tinyFaaS [8]	Edge	Containers	Node.js	Docker	✓ [71]
Stack4Things [72]	Edge	Containers	Python, Node.js	Qinling, Iotronic	✓ [73]
Kappa [74]	Edge	Calvin Runtime	Calvin Script, Python	Calvin	✕
Sched-Sim [75]	Edge	Containers	Any	Kubernetes	✓ [75]
Serverless MEC [76]	Edge	Native	Any	Any	✓ [77]

Real-world implementations of serverless edge platforms have also attracted a great research interest. A number of different approaches have been taken, such as: reusing existing commercial offerings, adapting popular open-source alternatives to the network edge, or developing completely new solutions from scratch. A very encouraging aspect is the fact that the majority of the discussed platforms below have chosen to open-source their implementations, allowing others to reuse, or even improve them in the future. Another

interesting development are efforts to fuse new serverless platforms with existing popular commercial alternatives, through the introduction of various compatibility layers, allowing function reuse.

Table 3 lists the main characteristics of the discussed platforms below, and offers insight at their primary execution location, utilized runtime technology, supported languages for writing serverless functions, as well as their underlying base on top of which they are developed. It is evident that many researchers opt to base their implementations on existing serverless platforms, some of them open-source, a topic that is further discussed in section 0, Open-Source Solutions.

Serverless Platforms Supporting Multiple Execution Locations

The current serverless platform research is focused on providing versatile solutions capable of running both at the edge and in the cloud, thus ensuring an edge-cloud continuum, utilizing decision making algorithms and finding the most optimal execution location. 13 of the selected platforms mention that they are designed to take advantage of more than one execution location. The basic idea is to use the edge to process smaller data samples, offering a local view, while exploiting the larger processing power of the cloud to generate global views [70].

To this effect, Pinto et al. [58] develop a platform [59] utilizing a proxy component that can measure the execution time of serverless functions and based on past experience decide at what layer to execute a given function. The decision-making process in terms of finding the most optimal execution location is compared to the multi-armed bandit problem, and three different algorithmic implementations are presented, balancing the exploration and exploitation aspects of the issue. Additionally, both [50] and [46] offer platforms with incorporated scheduling logic for edge-cloud continuum. Whereas [46] factors in the functions' requirements in terms of service latency, battery consumption, and availability during the decision process, [50] aims to provide a compatibility layer between various serverless providers, taking into account the execution cost at the various layers as well. The presented solution continuously monitors the pricing information of multiple public serverless providers and based on this information deploys it to the most affordable one, as long as the function's resource requirements are satisfied. To ensure immutability of the recorded pricing information and prevent price tampering by the administrators of the unified platform, a blockchain backed storage medium is used for storing historical price data. Similar to these approaches, Zhang et al. [45] present a platform suited for AI workloads, which can either be natively executed at the edge, without any runtime abstraction, or in the cloud using the Kubeless open-source serverless platform. To better support complex workloads, the availability of specialized hardware is also considered during the decision process.

Avoiding the problem of deploying elaborate edge infrastructure or relaying on commercial providers, Avasalcai et al. [47] develop a community platform where participating users can opt to share their resources with each other. Each application is represented by a series of tasks that can be scheduled on available devices at the edge. If not enough resources are available, cloud capacity can be leveraged as well. The incentive mechanism which would entice users to share their resources, as well as means for task migration between devices have not been defined yet and will be a subject of future research.

Extending Existing Serverless Platforms

Research efforts are not concentrated only on developing novel solutions, but also improving the existing ones, in some cases even those that are commercially available. One such example is the Clemmys [28] platform, extending OpenWhisk and adding support for the Intel SGX secure enclave and describing optimizations that reduce the performance penalty, achieving comparable performance to native OpenWhisk execution. The use of Intel SGX in a serverless context is discussed in greater detail in section 0. Another example which improves an existing commercial solution is described in [69], where the AWS serverless products are used to develop a platform that can either execute serverless functions at the edge, using devices enrolled into AWS Greengrass, or in the cloud, utilizing the Lambda serverless service. Using automatic function downloading, Greengrass devices can autonomously reconfigure themselves, and fetch any missing functions from the cloud, thus optimizing the storage usage. No advanced decision making is involved, and the caller of the function decides whether to invoke the function at the edge or in the cloud. In cases where the cloud is chosen as the execution location, the local edge devices act as proxies in the edge-cloud communication.

Open-source alternatives which can also transform edge devices to function executors are available as well. Tricomi et al. [72] present a platform [73] which modifies the Qinling FaaS project by OpenStack [78] to extend function execution to end-devices. Additionally, a graphical user interface is offered, based on the popular Node-Red project, allowing users to easily define function chains and processing steps. As with other container-based solutions, per request isolation is not guaranteed since multiple functions requests can be executed by the same container. Similarly to this, Baresi et al. [55] also extend an existing open-source product, in this case the OpenWhisk serverless solution, and develop a shared persistence layer, acting as a cache between different function invocations, mitigating the unpredictability of whether any subsequent function invocations will be executed in the same or in a different container environment. As previously, no request level isolation is guaranteed, instead leaving the decision-making process to the OpenWhisk scheduler.

Serverless Platforms Compatibility Layers

To ease the adoption of a newly proposed solution, some platforms provide compatibility layers with existing commercial services, allowing function reuse. Such compatibility layers have been implemented previously as well, one prime example being the wide adoption of the S3 protocol initially developed by AWS for its object storage service. Today, various products exist, both open-source and commercial, which reuse the S3 API, providing programmatic compatibility with the original service. The authors of [61] present a system that offers a serverless compatibility layer with AWS Lambda and can run across a wide variety of devices with different hardware specifications, either at the edge, fog or in the cloud. This approach allows existing Lambda functions to be migrated and executed on the new platform, without requiring a rewrite. Containers are used as a runtime environment, and to mitigate the cold start issue, a single container can be shared by multiple functions, using memory isolation between them.

Intelligent Scheduling of Serverless Functions

Another technique for avoiding the cold start issue apart from container re-use is container pre-warming, where container instances are started ahead of time, and wait for a request to be received. Of course, while this does solve the initial delay incurred during the first start-up, it leads to higher resource usage, since resources are utilized for larger periods of time, even

when not explicitly required. The Pigeon [30] framework is one such example that implements an oversubscribed static pool of pre-warmed containers, keeping ready container instances of all resource sizes.

Cheng et al. [54] also introduce a system which can decide the execution location of a function based on multiple optimization parameters, but in contrast to the previously introduced solutions, the capability for migrating workloads on-the-fly between the different layers during execution is provided as well. This is made possible by joining all nodes into a pool representing a hierarchical overlay, from where execution targets can be dynamically chosen. The serverless functions are then executed by a container runtime on the selected node.

Finally, the authors of [64] promote the idea of using serverless functions in a scientific context, to increase the speed with which large amounts of data can be analyzed. Depending on the desired execution location, the platform can either take advantage of Kubernetes clusters operated on private clouds or at the network edge, as well as the commercial AWS Lambda service offering. This is especially useful for privacy sensitive information that needs to be preprocessed as soon as possible for anonymization purposes, before moving it to the cloud and leveraging its greater processing capacity for further analysis and aggregation.

Standalone Serverless Platforms for the Network Edge

When running serverless at the edge of the network, two distinct approaches are identified. The first approach involves reusing existing edge devices for function execution, utilizing their spare computing power. This is also called deviceless edge computing [70]. The second approach involves the building of a dedicated computing infrastructure with a single purpose, that of processing edge data. Even though often these two approaches are intertwined with each other, they are sometimes used to define even more hierarchical processing levels, placing them before the cloud.

An example of a serverless platform that is aimed solely at the edge is tinyFaaS [8], [71], where the driving idea is to make the implementation as lightweight as possible. Edge nodes are seen as independent devices, and no advanced scheduling algorithms are present. Functions are executed into containers with no per-request isolation, and to achieve lower overhead, the lightweight CoAP protocol is used for the function invocation endpoints, instead of the more traditional HTTP.

The repurposing of well-known DevOps tools at the edge has also been researched, Rausch et al. [49] describe efforts to implement a serverless edge platform on top of the Kubernetes container orchestrator, which, by default, comes bundled with a container scheduling algorithms on its own. Their results show that the included Kubernetes scheduler does not cope well in very dynamic environments, where large numbers of functions are invoked, as a result of the spawning and destruction of many containers in short amounts of time. Of course, this presents a challenge particularly to serverless projects that build their solution on top of Kubernetes, since they have to implement workarounds to this issue. As described previously, popular approaches include either container pre-warming or the elimination of runtime isolation, allowing container reuse. Both approaches have the same effect, reducing the number of container instantiations, or at least optimizing the time at which they are done. With many lightweight Kubernetes distributions targeted at the network edge[79], [80], even devices with

modest hardware, such as single board computers can be made part of a computing cluster for serverless functions.

Such development of solutions that allow any existing edge devices to be converted to serverless function executors has the potential to rapidly increase the adoption of serverless edge computing. Current examples include various commercial offerings such as AWS Greengrass and Azure IoT Hub, but research interest from academia is present as well. Persson et al. [74] discuss one such solution that can reuse existing devices for serverless execution, published in 2017, the same year that the first such commercial service, AWS Greengrass has officially been announced. The solution is built on top of the existing Calvin [81] framework, and uses CalvinScript for function declaration. Similar to its commercial counterparts, node tagging is also supported, allowing load scheduling based on the labels associated with a given device.

Multi-access Edge Computing – MEC Serverless Platforms

Multi-access edge computing (MEC) is an emerging trend, aiming to unify telecommunication technologies and infrastructure services, by collocating computing infrastructure with mobile providers' base stations. This approach, combined with advancements in telecommunication technologies such as the wider adoption of 5G, act as enablers for new use-cases, reducing latency and defining vendor and application independent interfaces. The development of the MEC initiative is done within the European Telecommunications Standards Institute (ETSI) [82], and has attracted a noticeable research interest as well.

Recently, attempts have been made to integrate serverless computing with the existing MEC initiatives, which might perhaps lead to greater vendor independence in the future through the adoption of the implementation agnostic MEC APIs. Cicconetti et al. in [76] elaborate their vision for such integration between serveless and MEC, tackling the function assignment problem between various infrastructures available at the providers' base stations, and testing their performance using simulations. The development of a MEC compliant interface for existing serverless architectures is a work in progress [77].

Contributing to the idea of moving serverless platforms close to the service providers' base stations, Yang et al. [48] reuse an existing open-source serverless project, OpenWhisk, and implement a prototype solution built on top of a software defined network (SDN) architecture. One of the benefits of making the service provider aware of the infrastructure is that function invocation can be done by dynamically reconfiguring DNS mappings, so that users are redirected to the most appropriate execution location, similarly to what is presented in [83].

Continuum

Table 4. Overview of Reseaerch Topics and Related Papers from a Continuum Perspective

Topic	Papers	Total
Edge, Cloud	[24], [25], [29], [42], [43], [45]–[47], [49], [61], [69], [70]	12
Edge, Fog, Cloud	[17], [50], [51], [54], [58], [64], [84], [85]	8

Edge computing with its idea of providing limited compute capacity closer to the data sources, near the edge, thus improving latency and using available bandwidth more efficiently, does not make the other infrastructures, placed higher in the hierarchy, obsolete. On the contrary, to

provide the best user experience, as well as the most economical solution, a way in which all existing infrastructure can be used is needed, exploiting the specific advantages of both the cloud, fog, and edge layers. **Table** 4 presents the included articles that discuss solutions capable of being executed at more than one network location at a given time.

This vision of achieving a hierarchical computing infrastructure, offering a seamless continuum between the different layers, is of great interest to both the creation of new scheduling algorithms as well as the development of full-fledged platforms, utilizing these algorithms. A careful balance must be established, between the edge and the cloud, because even though there is a reduction in latency, the computing power is often limited at the edge, and any time savings secured as a result of the reduced latency can be lost to longer execution times. For this reason, it is recommended to perform only initial preprocessing of the data at the edge, assuring its quality and providing a fast response to real-time applications. In cases where a more in-depth analysis is required, the filtered data should be sent to the cloud, leveraging its greater computing power. Zhang et al. utilize this concept of partitioning the workload in [51] to develop a system for video stream analysis. Multiple serverless functions are combined into a pipeline and partition points are defined. All functions before the given partition point are executed at the network edge, closer to the data source, while those after the partition point are sent to the cloud to leverage its better computing capacity.

There are already implementations of scheduling algorithms that dynamically take into account both edge and cloud resources, and try to optimize the function scheduling between them based on a number of different metrics [24], [25], [43]. Platforms that can be deployed across all computing layers, and transparently execute functions have been introduced as well [45], [46], [50], [54], [58], [69], [70].

The authors of [84] present one such platform that can be deployed across the whole network, and supports various execution environments, such as virtual machines, containers, and even HPC clusters. These environments are called pilots and developers can submit serverless functions for execution at a location of their choice. Communication between workloads executed on different pilots is possible using message brokers.

IoT Serverless Edge Applications

Despite the very active research in terms of serverless efficiency improvements, scheduling optimizations, and new platforms integrating these features, there is a sizable interest in the creation and description of serverless applications as well. These applications include, but are not limited to, the development of cyber-physical systems, smart city improvements, as well as better user experience for augmented reality and virtual reality workloads. While the level of technical sophistication varies, they all use either private or public serverless platforms, and in some cases extend this usage to the very network edge. Such practical examples verify the feasibility of the proposed solutions in relation to the existing open issues, as well as help in identifying new ones, by targeting specific use-cases or user requirements.

Table 5 showcases the relationship of the included papers to the subcategories discussed below, including their preferred deployment location.

Table 5. Overview of Research Topics and Related Papers from an IoT Serverless Applications Perspective

Topic	Papers	Total
Guidelines & Benefits	[9]–[11], [13], [14], [17], [18], [20], [21], [23], [49], [86]–[91]	17
Public Infrastructure	[10], [13], [23], [46], [52], [53], [83], [90], [92]–[96]	13
Private Infrastructure	[10], [13], [27], [46], [55], [83], [90], [92]–[95], [97]	12
Edge[a]	[14], [27], [52], [53], [55], [83], [87], [93]–[95], [97]	11
AR & VR	[13], [27], [40], [42], [46], [51], [55], [83]	8
AI & ML	[13], [14], [45], [48], [49], [64], [96]	7
Smart City	[23], [52]–[55], [64]	6
CPS	[92]–[95], [97]	5
MEC	[40], [48], [55], [76], [83]	5
Cloud[b]	[52], [53], [83], [93]	4
Fog[c]	[95]	1
Blockchain	[14]	1

[a] Denotes serverless applications designed to leverage edge computing
[b] Denotes serverless applications designed to leverage cloud computing alongside edge and fog
[c] Denotes serverless applications designed to leverage fog computing

CYBER-PHYSICAL SYSTEMS

Cyber-physical systems represent environments where computers are expected to directly interact with them, using real-world physical actions. Notable examples include smart grid management, medical systems, autonomous driving systems, and unmanned aerial vehicles (UAVs). The requirements for low latency computation, make serverless an option in the development of such applications, which is also confirmed by Gan et al. [92], who describe an application for coordinating swarms of UAVs during their execution of a specific action. Following the omnipresent dilemma of where to draw the line between a decentralized and a centralized system, two implementations are discussed. In the first one, the centralized one, the drones stream the data to the centralized platform which then analyzes it and extracts meaningful information, while in the second one, the computation is performed by the drones themselves, using their on-board processing power, resulting in a deviceless edge scenario, where end-devices themselves execute the workload. The main problem with the centralized solution is network congestion when there is a large number of drones that need to be coordinated, as well as the high round-trip latency. On the other hand, by using the decentralized approach, the devices are depleting their battery faster, reducing their flying time, and in some cases even becoming non-responsive, unable to do essential processing required for the execution of the flight, because of the high utilization incurred by the data processing tasks. A potential solution to these problems is to come up with a middle ground approach, where simple tasks that do not generate high compute loads are executed on the devices themselves, thus drastically reducing the round-trip latency. In contrast, high-load functions are executed in the cloud, since any latency benefits are eliminated as a result of the large processing times and faster battery depletion when executing them on the edge devices. This middle ground between these two approaches, decentralized and centralized, is an important aspect, not relevant only to cyber-physical systems, but to all IoT in general.

The reduced latency that processing at the network edge offers can be mission critical in certain scenarios and can decide whether a disaster will be averted or not. The authors of [97] describe

a smart oil field implementation which leverages serverless edge computing to analyze in real-time data generated from oil extraction sensors. In such unconventional environments, satellite communication is often the only way in which internet connectivity can be established, and using this approach allows all of the drawbacks associated with it to be alleviated, such as the cost, unreliability and high round-trip time.

Finally, Zhang et al. also present another implementation of a serverless application [93] for atypical scenarios, with components running both at the edge and in the cloud, whose purpose is to be used in emergency situations, where access to utilities might be limited. What is notable about this example is the fact that the whole solution, across all levels, ranging from the data acquisition, through its analysis and storage, to its presentation on a static web site, is implemented using the serverless approach, combining the FaaS and BaaS offerings of a commercial provider - AWS.

Smart City

While the line between cyber-physical systems and smart city applications can sometimes be bleak, serverless solutions have recently gained popularity in this area as well. The main goal of such applications is to simplify everyday lives and save costs by introducing smart IoT devices in these urban environments, which are in turn capable of monitoring and in some instances even automatically adjusting various aspects, such as: garbage usage [52], energy usage [94], [95], or access to transportation services [53].

Smart recycling is just one example where the cost efficiency of the serverless paradigm can provide great scalability at low cost, as a result of sporadic usage, and low data volume sent per device, albeit with a high number of devices in total. The authors of \cite{al-masri_recycleio_2018} present such a solution, capable of monitoring garbage bins, detecting classification violations and reporting remaining capacity. The data is first processed on local edge devices using a commercial serverless edge platform - Azure IoT Hub, before being sent to the cloud for more in-depth analysis and long-term storage.

Another solution which can directly benefit the everyday lives of citizens by reducing traffic congestion and by optimizing public transportation routes is presented in [53]. Through a combination of commercial FaaS and BaaS products an intelligent transportation system is introduced, which can identify rush hours and popular routes through the monitoring of ubiquitous Bluetooth equipped devices such as smartphones. The gathered data by the edge devices is sent to a BaaS database every 5 minutes, thus exploiting the scale-to-zero cost efficiency of the serverless approach. Various other smart city applications based on serverless functions executed at the network edge are also present in literature, such as the one for checking mask wearing compliance discussed in [64].

Scenarios requiring many more distributed edge devices have been shown to be feasible as well, with particular interest for the energy sector. Both [94] and [95] describe intelligent applications for monitoring energy management systems, utilizing edge and cloud devices, exploiting an edge-cloud continuum. Albayati et al. [95] envision an implementation of such a system at the country level, modernizing the currently used metering infrastructure. Authors of [94] extend this approach and implement Toci, an anomaly detection system for power grids, by using a commercial solution for deploying and management of edge devices (AWS

Greengrass) on top of which serverleses function can be executed, as well as a cloud FaaS counterpart (AWS Lambda), for additional analysis. The mutual compatibility of the Greengrass and Lambda solutions allows functions to be seamlessly reused between the edge and the cloud, without any modifications. Toci follows the same trend as [93] where the complete solution, together with the presentation of the data, is implemented using a serverless architecture.

Augmented Reality and Virtual Reality

AR and VR systems are distinct from the previously described categories since end-users have a direct interaction with them and the perceived latency plays a significant role in customer satisfaction. Furthermore, the nature of the workload requires specialized computing hardware such as graphics processing units (GPUs) that can accelerate the completion of the submitted tasks significantly, compared to a software implemented alternative, executed directly by the central processing units (CPUs).

To this effect, Salehe et al. [27] aim to exploit the fact that each household has a number of devices that are left idling for large periods of time during the day. By designing a JavaScript runtime capable of executing serverless functions on these performance constrained devices, a platform is developed, where each device can provide computing capacity for various purposes, for example AR/VR image analysis and processing. These home devices can be seen as edge device instances, and while they might not have great computing capacity, their strength lies in their numbers, and the extent to which the given workload can be granularly partitioned to run across them. A similar solution is provided for better equipped devices as well, where multiple concurrent executions can be supported by containerizing the developed runtime, instead of executing them natively.

The idea of utilizing serverless computing for AR/VR tasks is also presented by Baresi et al. [83], but in this case a multi-access edge computing architecture is described, co-locating computing infrastructure with mobile base stations. This is perhaps one of the first instances where MEC and serverless have been joined together. The mobile base station is responsible of routing the request for the serverless function to the nearest edge server that satisfies the requirements for its execution. The reference implementation, done using the open-source OpenWhisk platform, is compared to a cloud one, where AWS Lambda is utilized. The edge implementation shows significant benefits when compared to the cloud one in terms of latency, and comparable performance in scenarios with high concurrency of task execution.

In conclusion, the fact that there are already existing application implementations utilizing serverless in an IoT context is very encouraging and serve to further refine this computing paradigm. One of the outstanding issues, especially for more complex AR/VR scenarios is providing hardware acceleration features such as access to dedicated GPUs for running serverless functions. None of the commercial platforms currently support such a use case, but it is an area under active research, as showcased in [45].

Serverless Migration Guidelines and Benefits

Even though serverless offerings have been available for a number of years, the idea of using this computing paradigm at the edge for serving IoT workloads is more recent. Taking into account the vast number of IoT applications, and their different requirements in terms of cost, execution performance, and latency, specific guidelines are needed, elaborating the various

computing approaches that can be utilized, especially in resource constrained environments such as the edge. Pfandzelter et al. [86] present a decision framework, outlining what computing paradigm to use for applications dealing with either event processing or data analytics workloads. When it comes to event processing, their recommendation is to use infrastructure as close to the data source as possible, arguing that serverless is a preferred platform for such scenarios, as long as there is no complex shared state involved. By performing the event processing as close to the edge as possible, the latency is kept to a minimum, without incurring higher execution times due to the slower hardware, since the events are simpler. However, when it comes to more complex data streams, it is recommended to move to the cloud, since the faster execution times outweigh the latency advantages as a result of the closer proximity.

Further benefits and challenges for IoT serverless computing, with a focus on the edge are provided by Aslanpour et al. [9], agreeing that event driven applications, especially those with stateless lifecycles, are perhaps the best fit for serverless edge computing. The scale-to-zero feature further increases its attractiveness in scenarios where events are sporadic, allowing greater cost efficiency compared to the alternative approaches which keep the application instances warm at all times, incurring costs.

Finally, while the benefits of adopting serverless are enticing, the question of how to migrate existing workloads to a serverless architecture, remains. Authors of [23] propose such migration guidelines of existing applications, hoping to increase the serverless adoption, while Grossman et al. in [87] describe the migration process that they have undertaken to convert an existing application to a serverless architecture. The main goal is to avoid a full rewrite of the existing solution, and instead reuse as much of the existing codebase as possible, focusing instead on the required optimization aspects. One such aspect is the resulting function size, since traditional applications tend to use many libraries, all contributing to the total data volume. In order to reduce the critical start-up time of serverless function, as well as reduce storage costs, Christidis et al. in [23] propose a function minimization technique, where existing programming libraries are slimmed down by including only the code that is absolutely necessary for the application's execution, removing any unused functionality. This is possible by analyzing the function's access patterns to the library files and removing those which are unused. An alternative, which has the same goal of reduced function size is given in [24], where the authors present a way in which multiple functions and their libraries can be aggregated into efficient FaaS platform artifacts by grouping them together. Elgamal et al. [25] have implemented a similar approach, combining multiple serverless functions into a single one running on AWS Lambda, with the aim of reducing costs by avoiding extra charges resulting from transitions from one function to another, in cases where function chaining is performed.
Even though many papers are dedicated to outlining the benefits of serverless computing and various platforms exist with comprehensive documentation, developers still experience issues when converting existing and creating new serverless applications, as can be seen by the types and volumes of questions asked on popular developer forums like StackOverflow, a subject analyzed by Wen et al. in [88]. Results show that the majority of asked questions are related to application implementation and low-code development, with only 7.9% accounting for general questions regarding the serverless concepts. While this shows that there is enough information available for users to educate themselves about the basic concepts, more formal guidelines for adopting serverless computing and relevant programming patterns are required.

Benchmarking

The large amount of different serverless options available, targeted directly at the edge for IoT scenarios or designed for more resourceful environments such as the cloud, warrants some performance comparison to ease the decision-making process when choosing a new solution. However, such benchmarks must take into account platform specifics as well, such as the manner in which the computing resources are allocated, the runtime environment of the executing functions, the underlying hardware, as well as the type of workload being executed.
Table 6 categorizes the relevant benchmarking research depending on the type of platforms that it is aimed at. Many papers that discuss platform implementations or novel scheduling algorithms also offer benchmarking results, comparing their performance to alternatives. We have purposefully excluded these from the table below where we focus exclusively on full-fledged benchmarking suites. However, they are listed in the figure in section 0, visualizing the relationships between the different primary categories.

Table 6. Overview of Research Topics and Related Papers from a Benchmarking Perspective

Topic	Papers	Total
Public Infrastructure	[92], [96], [98], [99]	4
Private Infrastructure	[92], [98], [100]	3

Benchmarking Suites for Commercial and Open-Source Serverless Platforms

Palade et al. [100] evaluate open-source serverless platforms that can be deployed on modest edge infrastructure, where latency plays a key role. By simulating a resource constrained environment using only two nodes on top of which the popular container orchestrator Kubernetes is installed, the response time and the success rate of functions instantiated by the Kubeless, OpenWhisk, OpenFaaS, and Knative serverless platforms is evaluated. Using a JMeter benchmarking scenario a simulation is done where IoT devices continuously push measured data to a serverless function running on one Kubernetes node, placed within the same local network. Results show that Kubeless provides the shortest response time, comparable number of transactions per second with the other tested platforms, and high function success rates even in cases with high concurrency.

Das et al. [98] tackle the same issue of serverless platforms performance analysis, but from a different perspective, focusing instead on commercial offerings. Performance of function execution at the edge is evaluated by testing the AWS Greengrass and Azure IoT Hub products, which allow the respective runtimes to be installed on a customer owned equipment, thus enabling the same functions that can be instantiated on the respective cloud services, to run natively on the devices. These results are then cross-referenced with measurements obtained by applying the same testing scenarios to the cloud-based services, confirming the latency advantages of the edge solutions. The initial set of developed benchmarks is open-sourced [101] and is divided into three distinctive categories: i) speech-to-text generation; ii) image recognition; iii) sensor emulation using a scalar value generator. Unfortunately, the persistent vendor lock-in problem and the lack of standardized APIs, means that support for any other platforms, besides AWS and Azure is lacking, leaving it up to the open-source community to pursue this effort.

Kim et al. [96] take a similar approach to Das et al. and develop 4 benchmark types that can be executed exclusively on cloud-based commercial serverless platforms. These categories apart

from simulating real-world workloads such as machine learning model manipulations, and application execution performance, also include microbenchmarks, executed using traditional command line tools like iperf and dd. The idea behind these tools is to test raw hardware performance, such as network throughput, input-output operations per second supported by the storage, and number of executed instructions. Even though no edge scenarios are currently supported, neither open-sourced nor commercial, the source code is publicly available [102], providing means for further extension to different providers, or execution locations. Authors of [92] also try to offer an alternative to microbenchmarks, one that better represents real-world workloads. For this reason, they develop an open-source benchmarking suite [103] consisting of five applications which can be used for performance measurements of serverless platforms. The applications encompass scenarios such as payment systems, e-commerce, social networking, and UAV swarm coordination, providing workloads representative of the real world.

This idea of creating a set of reproducible, cross-platform benchmarks for evaluating different serverless implementations is also discussed by Gorlatova et al. [99], at an even larger scale, across 6 different locations. The selected infrastructures include local devices and a server, simulating the network edge, as well as both conventional and serverless cloud services from multiple service providers. Interestingly, to better illustrate the real-world network latency, measurements are performed from different physical locations, and surrounding conditions. Results show that the previously described cold start problem plays a significant role in the overall function execution delay, leading to increases of over 40 times in the most extreme case.

Migration Guidelines & Benefits

Through the analysis of the selected papers, we have identified a significant number of entries that provide implementation results and outline the benefits of the new implementation by comparing it to existing alternatives. This is of course, highly beneficial, and contributes to the body of knowledge regarding performance characteristics of the various available options. However, one major issue is still present in this area, and that is the portability of the devised benchmarking suites. Without any uniform API for function deployment and connection to supporting services, researchers have to manually adapt their implementations to the different offerings, in most cases prioritizing popular and well-established services, hindering the adoption of newer and perhaps more efficient alternatives. This issue also makes it difficult and very time consuming to directly compare commercial and open-source serverless solutions, because of their different APIs. The best solution to this problem would of course be the adoption of a standardized abstraction layer, which would allow cross-platform portability of functions.

Serverless Security, Integrity, and Policy

Even though the serverless paradigm aims to simplify feature development and provide a more agile workflow without burdening the user with infrastructure management, there are still security concerns that must be considered, both by the developers themselves, as well as the platform providers. This is especially true for multi-tenant environments where different users might share the same underlying infrastructure, such as at the network edge. Current research aims not only to mitigate threats from other functions submitted by malicious actors, but also from hostile platform operators as well. Unfortunately, there is no consensus on the recommended security and data integrity features implemented by different platforms, which

combined with the absence of a uniform API, leads to issues when searching for alternative providers. **Table** 7 provides a summary of the relevant topics and associated papers in terms of serverless security, data integrity, and policy, before continuing with a more in-depth discussion of these issues below.

Table 7. Overview of Research Topics and Related Papers from a Security, Integrity, and Privacy Perspectives

Topic	Papers	Total
Vendor Lock-In	[16], [24], [50], [89]	4
Trustworthiness	[28], [89], [104], [105]	4
Data Repair	[85]	1

The Vendor Lock-In Problem

One of the top problems currently facing serverless adoption is the associated vendor-lock in resulting from the different provider implementations. This is especially true for IoT environments, where the diverse nature of devices with varying hardware configurations narrows down the set of platforms that can be used, resulting in scenarios where the same logic needs to be implemented multiple times, for different serverless platforms, or even different runtimes and programming languages. While a unified, cross-platform API is currently lacking, there are efforts to independently overcome these imposed limitations. Serveless.com [106], [107] offers the ability to translate serverless functions and deploy them to different cloud providers using a unified client utility. Of course, this does not equal an official support by the end-providers, making exclusive features hard to support. However, by relying on the support of the open-source community, it is possible to scale the effort of supporting new platforms, and in some cases this has led to the development of compatibility layers for open-source serverless solutions as well [108].

The latest research in this field is not focused only on providing a cross-platform layer, but also of devising an intelligent way of scheduling the required functions across the different platforms, taking into account either programming environment restrictions, or user preferences. Pelle et al. [24] provide an abstraction layer that can be used for programming serverless functions, allowing such implementations to be later instantiated on commercial platforms, based on a scheduling logic. The authors of [50] also extend this idea by implementing a platform providing a unified API capable of translating the published functions to different platform implementations, establishing an edge-cloud continuum through intelligent function scheduling.

There are also examples of platform implementations that natively support functions developed for popular cloud providers' APIs, thus offering easier adoption by existing users of these public platforms. One such example is CSPOT [61], [62], which supports functions written for AWS Lambda.

Function Marketplaces

Having a unified API for serverless function development, would not only avoid any vendor-lock in and allow easier migration between platforms, but would also promote the use of public function marketplaces, where developers can share their functions with others. Combining this approach with the mash-up development paradigm would lead to decreased development

times, and arguably better code quality, since developers can collaborate on a common implementation of a function, used by many people, instead of devising their own solutions. Of course, such function sharing does not imply that it will solely be done on a free-of-charge basis, developers should be able to opt to monetize their code. This might lead to even new business models, where developers not only charge a one-time fee for the function code itself, but rather per invocation of the code, further blurring the lines between the various * as a service offerings. The first function marketplaces are already available [109], but unfortunately they are platform exclusive, lacking a standardized API.

Isolation & Security

In any code sharing scenario, one of the first issues that arises is the question of security. It is not hard to envision malicious functions posted on the previously described function marketplaces, aiming to compromise either the underlying infrastructure where the code is run, or the processed data itself. Datta et al. [104] propose a solution to this problem by introducing Valve, a security framework for serverless functions that is capable of establishing a security baseline, and enforcing policies based on this baseline. The concepts of security baselining and policy definition is well known from other areas and is the way in which the popular SELinux framework is implemented. In this manner, data exfiltration and infrastructure abuse can be averted, albeit with a performance hit to the execution performance, because of the real-time policy enforcement. Function size is also increased, as a consequence of the introduced sidecar applications, responsible for monitoring the function's behavior and intercepting any file or network access. Another approach discussed in literature [28], [105] is the use of secure enclaves, such as Intel Software Guard Extensions (SGX), protecting the runtime environment of the functions and allowing execution on untrusted infrastructures, by relying on the on-the-fly memory encryption and decryption capabilities of the CPU. The authors of [89] review further serverless security issues and evaluate alternate ways in which higher levels of function isolation can be achieved, such as the the modification of existing JavaScript engines to ensure separation.

Data Integrity

The FaaS serverless paradigm of chaining multiple simple functions together to complete a more complex task leads to the question of data integrity as it is passed between the various intermediary processing steps. One area in which this issue is present is IoT, where the real-time nature of the output data makes it hard to repeat any computations in case there is a data corruption leading to a computational error. SANS-SOUCI [85] is an extension to the previously mentioned CSPOT [61], which allows data repair by implementing an append-only log of all data processed by functions. In this way, any errors can be mitigated by replaying the computation after the necessary changes have been made to the function code, or without any alternations, but simply at a later time. This concept of an append-only storage also allows data gathered in the past to be replaced with more precise data in the future, or to process real-time data using a different time resolution.

Open-Source Solutions

Many of the presented research papers focusing on development of new edge serverless solutions used an existing open-source serverless platform and by extending it, adapted it to run at the edge.

Table 8 outlines these papers, as well as categorizing them whether they have used an open-source software (OSS) platform or have in turn open-sourced their code.

Table 8. Overview of Research Topics and Related Papers from an Open-Source Software Perspective

Topic	Papers	Total
Based on an OSS Platform	[16], [26], [28], [30], [38]–[41], [45], [46], [48], [49], [54], [55], [64], [72], [74], [83], [85], [87], [89], [100], [104], [105]	24
Published as OSS	[8], [20], [22], [26], [46], [47], [58], [61], [64], [72], [76], [85], [87], [92], [96], [98]	16

To better understand the varying levels of popularity of the different platforms, we have classified the analyzed papers in terms of which platform they based their implementation on. Figure 2 shows the popularity distribution of each mentioned platform, while
Table 9 links the associated papers with the underlying platform of their choice. OpenWhisk is the most popular base platform, chosen by 9 papers in total. This is not surprising since even commercial offerings have contributed to the wide popularity that it currently enjoys by providing compatible solutions. One such example is IBM Cloud Functions [110]. Following it is OpenFaaS, another versatile solution with a wide array of supported programming languages in which serverless functions can developed, as well as offering the possibility of using custom container images, allowing developers to specify the execution environment. Kubeless and Knative follow next in third and fourth place, respectively, both being Kubernetes centric solutions. In comparison, the other solutions have more versatile deployment options, supporting other container orchestrators as well. Finally, Calvin, FogFlow, and TinyFaaS have only been utilized once, both representing platforms that have been presented in previous research, and later adapted to execute serverless functions at the edge.

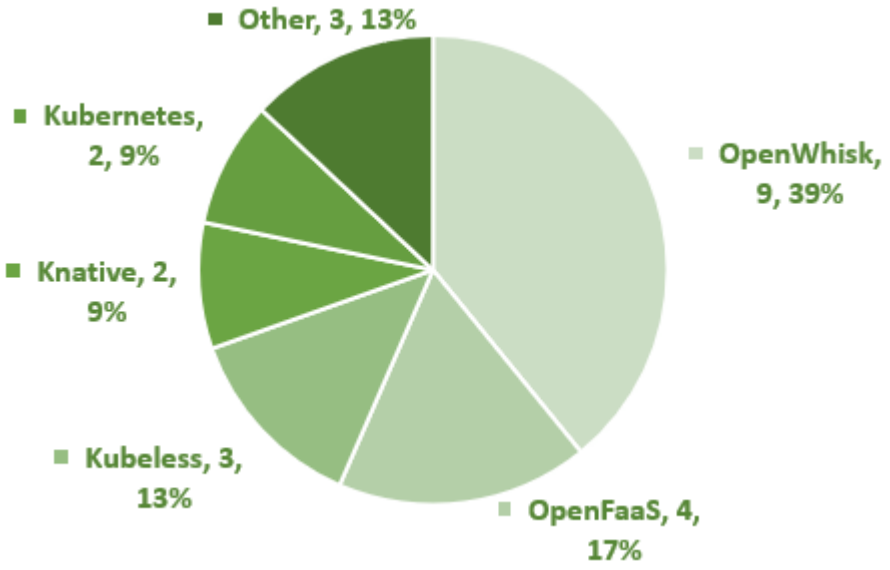

Figure 2. Most Popular Open-Source Serverless Platforms

In total, 40 of the 67 analyzed papers have directly or indirectly contributed to the open-source community, by either extending an existing open-source platform (24 papers) or publicly publishing the source code for their implementation (16 papers).

Table 9. Usage Distribution of Open-Source Serverless Platforms

Platform Name	Used By	Total
OpenWhisk	[26], [28], [38], [40], [46], [48], [55], [83], [100]	9
OpenFaaS	[58], [87], [100], [104]	4
Kubeless	[41], [45], [100]	3
Knative	[39], [100]	2
Kubernetes	[30], [64]	2
Calvin	[74]	1
FogFlow	[54]	1
TinyFaaS	[8]	1

Summary of Open Issues

Considering the complex relationships between the various topics discussed in the analyzed papers, **Figure** 3 provides a visual representation of the number of entries related to each identified open issue, while also serving as a reference to the related papers that discuss it. This figure aims to extend the tables present in the previous sections, focusing on the global view instead of on the local view which was provided within the sections discussing the respective categories above. The numbers on the lines connecting the open issues with their subtopics are color-coded according to the color of the subtopic on the right-hand side and relate to the references available at the end of this paper. For clarity, review papers as secondary literature are explicitly listed at the end.

CONCLUSION

By searching 6 popular research databases, we have analyzed 67 papers related to the novel trend of applying serverless computing at the network edge. Through this analysis we have derived a classification framework consisting of 8 distinct categories and 30 unique subcategories. Current research trends are focused on development of new serverless platforms deployable across the whole edge-fog-cloud continuum by using advanced scheduling algorithms and optimizing either latency, price, or bandwidth utilization.

Serverless computing has also recently been applied to real-world problems at the network edge as well, primarily aimed at event based IoT applications. However, one persistent issue being faced is the runtime efficiency and incurred start up delays when presented with a high frequency of function invocations. Recent efforts made using WebAssembly and the idea of adopting unikernels as a possible more lightweight alternative to containers is promising, but further work is needed on increasing the number of supported programming languages and tooling in terms of WebAssembly and concrete implementations for unikernels. Functions executing complex AI algorithms would also benefit from specialized hardware, support for which is currently lacking in many platforms [90].

In the coming years, with the increase in IoT devices utilized by customers and industry alike, there will be an even greater focus on long-term security and privacy protecting measures that can be implemented to safeguard critical information. Strict function isolation [91] with the

help of novel hardware features, along with sane data processing policies are inevitable in this aspect. One of the prominent advantages of edge computing, apart from the reduced latency is exactly the benefit of reducing private information flow to third parties before it is preprocessed and anonymized.

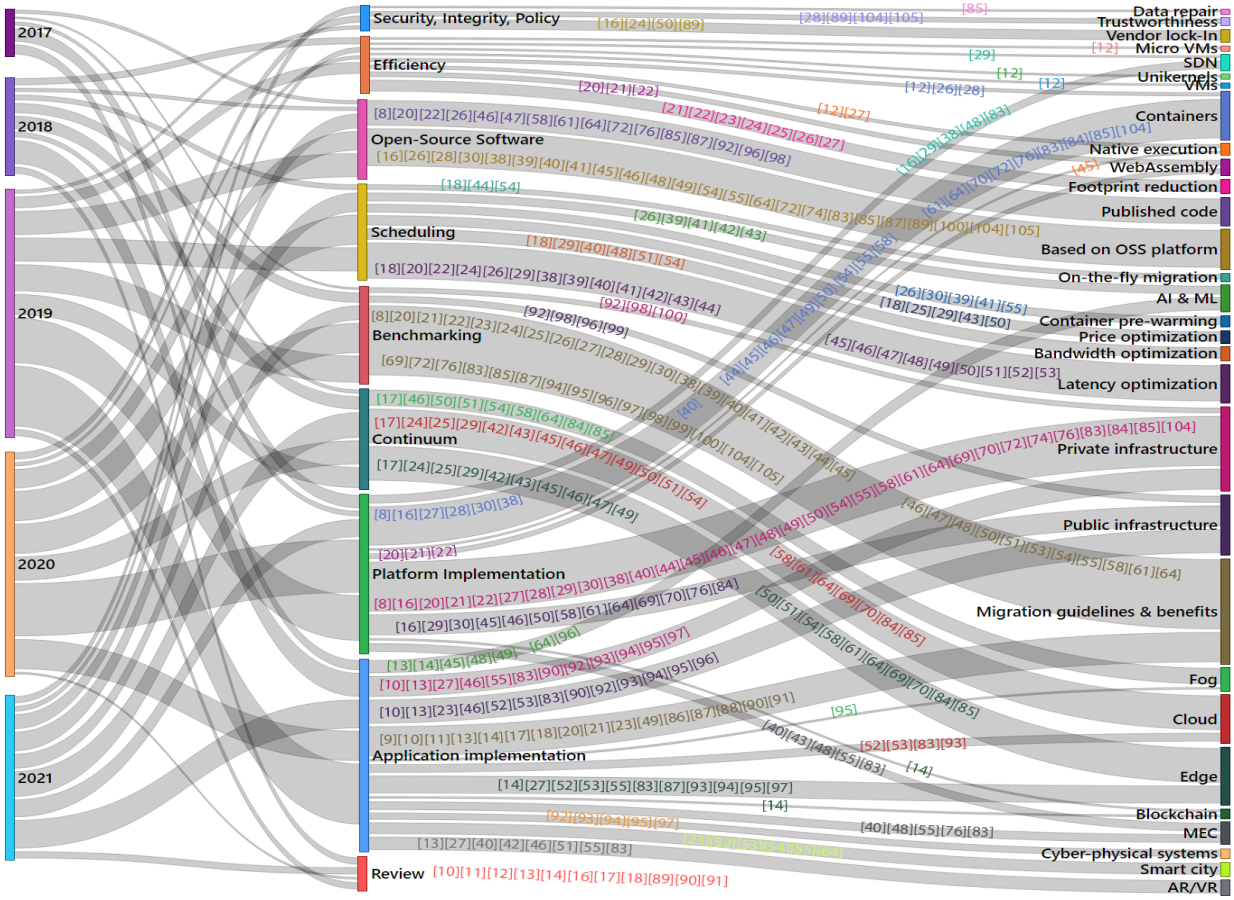

Figure 3. Topics and Subtopics Covered by each Analyzed Paper

References

[1] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, 'Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends', in *2015 IEEE 8th International Conference on Cloud Computing*, Jun. 2015, pp. 621–628. doi: 10.1109/CLOUD.2015.88.

[2] S. Eismann *et al.*, 'Serverless Applications: Why, When, and How?', *IEEE Software*, vol. 38, no. 1, pp. 32–39, Jan. 2021, doi: 10.1109/MS.2020.3023302.

[3] 'AWS Lambda – Serverless Compute - Amazon Web Services', *Amazon Web Services, Inc.* https://aws.amazon.com/lambda/ (accessed Apr. 26, 2021).

[4] 'AWS IoT Greengrass - Amazon Web Services', *Amazon Web Services, Inc.* https://aws.amazon.com/greengrass/ (accessed Apr. 26, 2021).

[5] 'IoT Hub | Microsoft Azure'. https://azure.microsoft.com/en-us/services/iot-hub/ (accessed Apr. 26, 2021).

[6] P. Kravchenko, *kpavel/openwhisk-light*. 2020. Accessed: May 09, 2021. [Online]. Available: https://github.com/kpavel/openwhisk-light

[7] O. Ltd, 'Meet faasd - portable Serverless without the complexity of Kubernetes', *OpenFaaS - Serverless Functions Made Simple*, Apr. 17, 2020. https://www.openfaas.com/blog/introducing-faasd/ (accessed Sep. 06, 2021).

[8] T. Pfandzelter and D. Bermbach, 'tinyFaaS: A Lightweight FaaS Platform for Edge Environments', in *2020 IEEE International Conference on Fog Computing (ICFC)*, Sydney, Australia, Apr. 2020, pp. 17–24. doi: 10.1109/ICFC49376.2020.00011.

[9] M. S. Aslanpour *et al.*, 'Serverless Edge Computing: Vision and Challenges', in *2021 Australasian Computer Science Week Multiconference*, New York, NY, USA, Feb. 2021, pp. 1–10. doi: 10.1145/3437378.3444367.

[10] B. Varghese and R. Buyya, 'Next generation cloud computing: New trends and research directions', *Future Generation Computer Systems*, vol. 79, pp. 849–861, Feb. 2018, doi: 10.1016/j.future.2017.09.020.

[11] R. Buyya *et al.*, 'A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade', *ACM Comput. Surv.*, vol. 51, no. 5, pp. 1–38, Jan. 2019, doi: 10.1145/3241737.

[12] N. Kratzke, 'A Brief History of Cloud Application Architectures', *Applied Sciences*, vol. 8, no. 8, p. 1368, Aug. 2018, doi: 10.3390/app8081368.

[13] H Shafiei, A Khonsari, and P Mousavi, 'Serverless Computing: A Survey of Opportunities, Challenges and Applications', 2019, doi: 10.13140/RG.2.2.32882.25286.

[14] S. S. Gill *et al.*, 'Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges', *Internet of Things*, vol. 8, p. 100118, Dec. 2019, doi: 10.1016/j.iot.2019.100118.

[15] I. Baldini *et al.*, 'Serverless Computing: Current Trends and Open Problems', in *Research Advances in Cloud Computing*, S. Chaudhary, G. Somani, and R. Buyya, Eds. Singapore: Springer, 2017, pp. 1–20. doi: 10.1007/978-981-10-5026-8_1.

[16] N. El Ioini, D. Hästbacka, C. Pahl, and D. Taibi, 'Platforms for Serverless at the Edge: A Review', in *Advances in Service-Oriented and Cloud Computing*, vol. 1360, C. Zirpins, I. Paraskakis, V. Andrikopoulos, N. Kratzke, C. Pahl, N. El Ioini, A. S. Andreou, G. Feuerlicht, W. Lamersdorf, G. Ortiz, W.-J. Van den Heuvel, J. Soldani, M. Villari, G. Casale, and P. Plebani, Eds. Cham: Springer International Publishing, 2021, pp. 29–40. Accessed: Apr. 09, 2021. [Online]. Available: http://link.springer.com/10.1007/978-3-030-71906-7_3

[17] L. Bittencourt *et al.*, 'The Internet of Things, Fog and Cloud continuum: Integration and challenges', *Internet of Things*, vol. 3–4, pp. 134–155, Oct. 2018, doi: 10.1016/j.iot.2018.09.005.

[18] M. Adhikari, T. Amgoth, and S. N. Srirama, 'A Survey on Scheduling Strategies for Workflows in Cloud Environment and Emerging Trends', *ACM Comput. Surv.*, vol. 52, no. 4, pp. 1–36, Sep. 2019, doi: 10.1145/3325097.

[19] V. Kjorveziroski, S. Filiposka, and V. Trajkovik, 'IoT Serverless Computing at the Edge: A Systematic Mapping Review', *Computers*, vol. 10, no. 10, Art. no. 10, Oct. 2021, doi: 10.3390/computers10100130.

[20] P. K. Gadepalli, S. McBride, G. Peach, L. Cherkasova, and G. Parmer, 'Sledge: a Serverless-first, Light-weight Wasm Runtime for the Edge', in *Proceedings of the 21st International Middleware Conference*, New York, NY, USA, Dec. 2020, pp. 265–279. doi: 10.1145/3423211.3425680.

[21] A. Hall and U. Ramachandran, 'An execution model for serverless functions at the edge', in *Proceedings of the International Conference on Internet of Things Design and Implementation*, New York, NY, USA, Apr. 2019, pp. 225–236. doi: 10.1145/3302505.3310084.

[22] P. K. Gadepalli, G. Peach, L. Cherkasova, R. Aitken, and G. Parmer, 'Challenges and Opportunities for Efficient Serverless Computing at the Edge', in *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, Oct. 2019, pp. 261–2615. doi: 10.1109/SRDS47363.2019.00036.

[23] A. Christidis, R. Davies, and S. Moschoyiannis, 'Serving Machine Learning Workloads in Resource Constrained Environments: a Serverless Deployment Example', in *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*, Kaohsiung, Taiwan, Nov. 2019, pp. 55–63. doi: 10.1109/SOCA.2019.00016.

[24] I. Pelle, J. Czentye, J. Doka, A. Kern, B. P. Gero, and B. Sonkoly, 'Operating Latency Sensitive Applications on Public Serverless Edge Cloud Platforms', *IEEE Internet Things J.*, pp. 1–1, 2020, doi: 10.1109/JIOT.2020.3042428.

[25] T. Elgamal, 'Costless: Optimizing Cost of Serverless Computing through Function Fusion and Placement', in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Seattle, WA, USA, Oct. 2018, pp. 300–312. doi: 10.1109/SEC.2018.00029.

[26] B. Wang, A. Ali-Eldin, and P. Shenoy, 'LaSS: Running Latency Sensitive Serverless Computations at the Edge', in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, New York, NY, USA: Association for Computing Machinery, 2020, pp. 239–251. Accessed: Sep. 02, 2021. [Online]. Available: https://doi.org/10.1145/3431379.3460646

[27] M. Salehe, Z. Hu, S. H. Mortazavi, I. Mohomed, and T. Capes, 'VideoPipe: Building Video Stream Processing Pipelines at the Edge', in *Proceedings of the 20th International Middleware Conference Industrial Track*, Davis CA USA, Dec. 2019, pp. 43–49. doi: 10.1145/3366626.3368131.

[28] B. Trach, O. Oleksenko, F. Gregor, P. Bhatotia, and C. Fetzer, 'Clemmys: towards secure remote execution in FaaS', in *Proceedings of the 12th ACM International Conference on Systems and Storage*, New York, NY, USA, May 2019, pp. 44–54. doi: 10.1145/3319647.3325835.

[29] I. Pelle, F. Paolucci, B. Sonkoly, and F. Cugini, 'Latency-Sensitive Edge/Cloud Serverless Dynamic Deployment Over Telemetry-Based Packet-Optical Network', *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 9, pp. 2849–2863, Sep. 2021, doi: 10.1109/JSAC.2021.3064655.

[30] W. Ling, L. Ma, C. Tian, and Z. Hu, 'Pigeon: A Dynamic and Efficient Serverless and FaaS Framework for Private Cloud', in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, Dec. 2019, pp. 1416–1421. doi: 10.1109/CSCI49370.2019.00265.

[31] *gwsystems/sledge-serverless-framework*. The Embedded and Operating Systems group at GWU, 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/gwsystems/sledge-serverless-framework

[32] *gwsystems/aWsm*. The Embedded and Operating Systems group at GWU, 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/gwsystems/aWsm

[33] 'Nuclio', *nuclio*. https://nuclio.io/ (accessed Apr. 27, 2021).

[34] 'Firecracker – Lightweight Virtualization for Serverless Computing', *Amazon Web Services*, Nov. 26, 2018. https://aws.amazon.com/blogs/aws/firecracker-lightweight-virtualization-for-serverless-computing/ (accessed Apr. 27, 2021).

[35] 'Firecracker – Secure and fast microVMs for serverless computing'. https://firecracker-microvm.github.io/ (accessed Apr. 27, 2021).

[36] F. Manco *et al.*, 'My VM is Lighter (and Safer) than your Container', in *Proceedings of the 26th Symposium on Operating Systems Principles*, New York, NY, USA, Oct. 2017, pp. 218–233. doi: 10.1145/3132747.3132763.

[37] 'Projects | Unikernels'. http://unikernel.org/projects/ (accessed Apr. 27, 2021).

[38] C. Cicconetti, M. Conti, and A. Passarella, 'A Decentralized Framework for Serverless Edge Computing in the Internet of Things', *IEEE Transactions on Network and Service Management*, pp. 1–1, 2020, doi: 10.1109/TNSM.2020.3023305.

[39] I. Wang, E. Liri, and K. K. Ramakrishnan, 'Supporting IoT Applications with Serverless Edge Clouds', in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, Nov. 2020, pp. 1–4. doi: 10.1109/CloudNet51028.2020.9335805.

[40] C. Cicconetti, M. Conti, and A. Passarella, 'Low-latency Distributed Computation Offloading for Pervasive Environments', in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*, Kyoto, Japan, Mar. 2019, pp. 1–10. doi: 10.1109/PERCOM.2019.8767419.

[41] S. Agarwal, M. A. Rodriguez, and R. Buyya, 'A Reinforcement Learning Approach to Reduce Serverless Function Cold Start Frequency', in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, May 2021, pp. 797–803. doi: 10.1109/CCGrid51090.2021.00097.

[42] J. Patman, D. Chemodanov, P. Calyam, K. Palaniappan, C. Sterle, and M. Boccia, 'Predictive Cyber Foraging for Visual Cloud Computing in Large-Scale IoT Systems', *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2380–2395, Dec. 2020, doi: 10.1109/TNSM.2020.3010497.

[43] C. Cho, S. Shin, H. Jeon, and S. Yoon, 'QoS-Aware Workload Distribution in Hierarchical Edge Clouds: A Reinforcement Learning Approach', *IEEE Access*, vol. 8, pp. 193297–193313, 2020, doi: 10.1109/ACCESS.2020.3033421.

[44] P. Karhula, J. Janak, and H. Schulzrinne, 'Checkpointing and Migration of IoT Edge Functions', in *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking*, New York, NY, USA, Mar. 2019, pp. 60–65. doi: 10.1145/3301418.3313947.

[45] M. Zhang, C. Krintz, and R. Wolski, 'STOIC: Serverless Teleoperable Hybrid Cloud for Machine Learning Applications on Edge Device', in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Austin, TX, USA, Mar. 2020, pp. 1–6. doi: 10.1109/PerComWorkshops48775.2020.9156239.

[46] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi, 'A Unified Model for the Mobile-Edge-Cloud Continuum', *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–21, Apr. 2019, doi: 10.1145/3226644.

[47] C. Avasalcai, C. Tsigkanos, and S. Dustdar, 'Resource Management for Latency-Sensitive IoT Applications with Satisfiability', *IEEE Transactions on Services Computing*, pp. 1–1, 2021, doi: 10.1109/TSC.2021.3074188.

[48] S. Yang, K. Xu, L. Cui, Z. Ming, Z. Chen, and Z. Ming, 'EBI-PAI: Towards An Efficient Edge-Based IoT Platform for Artificial Intelligence', *IEEE Internet Things J.*, pp. 1–1, 2020, doi: 10.1109/JIOT.2020.3019008.

[49] T. Rausch, W. Hummer, V. Muthusamy, A. Rashed, and S. Dustdar, 'Towards a Serverless Platform for Edge {AI}', presented at the 2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19), 2019. Accessed: Mar. 23, 2021. [Online]. Available: https://www.usenix.org/conference/hotedge19/presentation/rausch

[50] Z. Huang, Z. Mi, and Z. Hua, 'HCloud: A trusted JointCloud serverless platform for IoT systems with blockchain', *China Communications*, vol. 17, no. 9, pp. 1–10, Sep. 2020, doi: 10.23919/JCC.2020.09.001.

[51] M. Zhang, F. Wang, Y. Zhu, J. Liu, and Z. Wang, 'Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines', in *Proceedings of the 12th ACM Multimedia Systems Conference*, New York, NY, USA: Association for Computing Machinery, 2021, pp. 80–93. Accessed: Sep. 02, 2021. [Online]. Available: https://doi.org/10.1145/3458305.3463377

[52] E. Al-Masri, I. Diabate, R. Jain, M. H. Lam, and S. Reddy Nathala, 'Recycle.io: An IoT-Enabled Framework for Urban Waste Management', in *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, Dec. 2018, pp. 5285–5287. doi: 10.1109/BigData.2018.8622117.

[53] L. F. Herrera-Quintero, J. C. Vega-Alfonso, K. B. A. Banse, and E. C. Zambrano, 'Smart ITS Sensor for the Transportation Planning Based on IoT Approaches Using Serverless and Microservices Architecture', *IEEE Intelligent Transportation Systems Magazine*, vol. 10, no. 2, pp. 17–27, Summer 2018, doi: 10.1109/MITS.2018.2806620.

[54] B. Cheng, J. Fuerst, G. Solmaz, and T. Sanada, 'Fog Function: Serverless Fog Computing for Data Intensive IoT Services', in *2019 IEEE International Conference on Services Computing (SCC)*, Jul. 2019, pp. 28–35. doi: 10.1109/SCC.2019.00018.

[55] L. Baresi and D. Filgueira Mendonca, 'Towards a Serverless Platform for Edge Computing', in *2019 IEEE International Conference on Fog Computing (ICFC)*, Prague, Czech Republic, Jun. 2019, pp. 1–10. doi: 10.1109/ICFC.2019.00008.

[56] C. Cicconetti, *ccicconetti/serverlessonedge*. 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/ccicconetti/serverlessonedge

[57] M. Zhang, *Heronalps/STOIC*. 2020. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/Heronalps/STOIC

[58] D. Pinto, J. P. Dias, and H. S. Ferreira, 'Dynamic Allocation of Serverless Functions in IoT Environments', in *2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC)*, Oct. 2018, pp. 1–8. doi: 10.1109/EUC.2018.00008.

[59] D. Pinto, *duartepinto/serverless-iot*. 2018. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/duartepinto/serverless-iot

[60] smartfog, *smartfog/fogflow*. 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/smartfog/fogflow

[61] R. Wolski, C. Krintz, F. Bakir, G. George, and W.-T. Lin, 'CSPOT: portable, multi-scale functions-as-a-service for IoT', in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, Arlington Virginia, Nov. 2019, pp. 236–249. doi: 10.1145/3318216.3363314.

[62] *MAYHEM-Lab/cspot*. MAYHEM-Lab, 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/MAYHEM-Lab/cspot

[63] 'deib-polimi/A3-E', *GitHub*. https://github.com/deib-polimi (accessed Apr. 27, 2021).

[64] S. Risco, G. Moltó, D. M. Naranjo, and I. Blanquer, 'Serverless Workflows for Containerised Applications in the Cloud Continuum', *J Grid Computing*, vol. 19, no. 3, p. 30, Jul. 2021, doi: 10.1007/s10723-021-09570-2.

[65] *SCAR - Serverless Container-aware ARchitectures*. GRyCAP, 2021. Accessed: Sep. 04, 2021. [Online]. Available: https://github.com/grycap/scar

[66] *OSCAR - Open Source Serverless Computing for Data-Processing Applications*. GRyCAP, 2021. Accessed: Sep. 04, 2021. [Online]. Available: https://github.com/grycap/oscar

[67] 'scar/examples/mask-detector-workflow at master · grycap/scar', *GitHub*. https://github.com/grycap/scar (accessed Sep. 04, 2021).

[68] cavasalcai, *Decentralized Resource Management for Latency-Sensitive IoT Applications with Satisfiability*. 2021. Accessed: Sep. 04, 2021. [Online]. Available: https://github.com/cavasalcai/Decentralized-Resource-Management

[69] T. Quang and Y. Peng, 'Device-driven On-demand Deployment of Serverless Computing Functions', in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Mar. 2020, pp. 1–6. doi: 10.1109/PerComWorkshops48775.2020.9156140.

[70] S. Nastic *et al.*, 'A Serverless Real-Time Data Analytics Platform for Edge Computing', *IEEE Internet Computing*, vol. 21, no. 4, pp. 64–71, 2017, doi: 10.1109/MIC.2017.2911430.

[71] *OpenFogStack/tinyFaaS*. OpenFogStack, 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/OpenFogStack/tinyFaaS

[72] G. Tricomi, Z. Benomar, F. Aragona, G. Merlino, F. Longo, and A. Puliafito, 'A NodeRED-based dashboard to deploy pipelines on top of IoT infrastructure', in *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, Bologna, Italy, Sep. 2020, pp. 122–129. doi: 10.1109/SMARTCOMP50058.2020.00036.

[73] *MDSLab/stack4things*. MDSLab, 2019. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/MDSLab/stack4things

[74] P. Persson and O. Angelsmark, 'Kappa: serverless IoT deployment', in *Proceedings of the 2nd International Workshop on Serverless Computing*, New York, NY, USA, Dec. 2017, pp. 16–21. doi: 10.1145/3154847.3154853.

[75] *Sched-Sim*. Accessed: Apr. 27, 2021. [Online]. Available: https://git.dsg.tuwien.ac.at/serverless-edge-ai/sched-sim

[76] C. Cicconetti, M. Conti, A. Passarella, and D. Sabella, 'Toward Distributed Computing Environments with Serverless Solutions in Edge Systems', *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 40–46, Mar. 2020, doi: 10.1109/MCOM.001.1900498.

[77] C. Cicconetti, *ccicconetti/etsimec*. 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/ccicconetti/etsimec

[78] 'Qinling - OpenStack'. https://wiki.openstack.org/wiki/Qinling (accessed Apr. 27, 2021).

[79] 'K3s: Lightweight Kubernetes'. https://k3s.io/ (accessed Sep. 05, 2021).

[80] 'MicroK8s - Zero-ops Kubernetes for developers, edge and IoT | MicroK8s', *microk8s.io*. http://microk8s.io (accessed Sep. 05, 2021).

[81] *EricssonResearch/calvin-base*. Ericsson Research, 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/EricssonResearch/calvin-base

[82] S. Dahmen-Lhuissier, 'ETSI - Multi-access Edge Computing - Standards for MEC', *ETSI*. https://www.etsi.org/technologies/multi-access-edge-computing (accessed Apr. 27, 2021).

[83] L. Baresi, D. Filgueira Mendonça, and M. Garriga, 'Empowering Low-Latency Applications Through a Serverless Edge Computing Architecture', in *Service-Oriented and Cloud Computing*, Cham, 2017, pp. 196–210. doi: 10.1007/978-3-319-67262-5_15.

[84] A. Luckow, K. Rattan, and S. Jha, 'Pilot-Edge: Distributed Resource Management Along the Edge-to-Cloud Continuum', in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Jun. 2021, pp. 874–878. doi: 10.1109/IPDPSW52791.2021.00130.

[85] W.-T. Lin, F. Bakir, C. Krintz, R. Wolski, and M. Mock, 'Data Repair for Distributed, Event-based IoT Applications', in *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*, Darmstadt Germany, Jun. 2019, pp. 139–150. doi: 10.1145/3328905.3329511.

[86] T. Pfandzelter and D. Bermbach, 'IoT Data Processing in the Fog: Functions, Streams, or Batch Processing?', in *2019 IEEE International Conference on Fog Computing (ICFC)*, Prague, Czech Republic, Jun. 2019, pp. 201–206. doi: 10.1109/ICFC.2019.00033.

[87] M. Großmann, C. Ioannidis, and D. T. Le, 'Applicability of Serverless Computing in Fog Computing Environments for IoT Scenarios', in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, New York, NY, USA, Dec. 2019, pp. 29–34. doi: 10.1145/3368235.3368834.

[88] J. Wen *et al.*, 'An empirical study on challenges of application development in serverless computing', in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, New York, NY, USA, Aug. 2021, pp. 416–428. doi: 10.1145/3468264.3468558.

[89] A. Bocci, S. Forti, G.-L. Ferrari, and A. Brogi, 'Secure FaaS orchestration in the fog: how far are we?', *Computing*, vol. 103, no. 5, pp. 1025–1056, May 2021, doi: 10.1007/s00607-021-00924-y.

[90] J. M. Hellerstein *et al.*, 'Serverless Computing: One Step Forward, Two Steps Back', presented at the Conference on Innovative Data Systems Research, Monterey, CA, Dec. 2018. Accessed: Apr. 09, 2021. [Online]. Available: http://arxiv.org/abs/1812.03651

[91] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, 'Survey on serverless computing', *J Cloud Comp*, vol. 10, no. 1, p. 39, Dec. 2021, doi: 10.1186/s13677-021-00253-7.

[92] Y. Gan *et al.*, 'An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems', in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, Apr. 2019, pp. 3–18. doi: 10.1145/3297858.3304013.

[93] S. Zhang, X. Luo, and E. Litvinov, 'Serverless computing for cloud-based power grid emergency generation dispatch', *International Journal of Electrical Power & Energy Systems*, vol. 124, p. 106366, Jan. 2021, doi: 10.1016/j.ijepes.2020.106366.

[94] F. Huber and M. Mock, 'Toci: Computational Intelligence in an Energy Management System', in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, Canberra, ACT, Australia, Dec. 2020, pp. 1287–1296. doi: 10.1109/SSCI47803.2020.9308324.

[95] A. Albayati, N. F. Abdullah, A. Abu-Samah, A. H. Mutlag, and R. Nordin, 'A Serverless Advanced Metering Infrastructure Based on Fog-Edge Computing for a Smart Grid: A Comparison Study for Energy Sector in Iraq', *Energies*, vol. 13, no. 20, p. 5460, Oct. 2020, doi: 10.3390/en13205460.

[96] J. Kim and K. Lee, 'FunctionBench: A Suite of Workloads for Serverless Cloud Function Service', in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, Milan, Italy, Jul. 2019, pp. 502–504. doi: 10.1109/CLOUD.2019.00091.

[97] R. F. Hussain, M. A. Salehi, and O. Semiari, 'Serverless Edge Computing for Green Oil and Gas Industry', in *2019 IEEE Green Technologies Conference(GreenTech)*, Apr. 2019, pp. 1–4. doi: 10.1109/GreenTech.2019.8767119.

[98] A. Das, S. Patterson, and M. Wittie, 'EdgeBench: Benchmarking Edge Computing Platforms', in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, Zurich, Dec. 2018, pp. 175–180. doi: 10.1109/UCC-Companion.2018.00053.

[99] M. Gorlatova, H. Inaltekin, and M. Chiang, 'Characterizing task completion latencies in multi-point multi-quality fog computing systems', *Computer Networks*, vol. 181, p. 107526, Nov. 2020, doi: 10.1016/j.comnet.2020.107526.

[100] A. Palade, A. Kazmi, and S. Clarke, 'An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge', in *2019 IEEE World Congress on Services (SERVICES)*, Jul. 2019, vol. 2642–939X, pp. 206–211. doi: 10.1109/SERVICES.2019.00057.

[101] A. Das, *akaanirban/edgebench*. 2020. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/akaanirban/edgebench

[102] *kmu-bigdata/serverless-faas-workbench*. BigData Lab. in KMU, 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/kmu-bigdata/serverless-faas-workbench

[103] 'DeathStarBench: An Open-Source End-to-End Microservices Benchmark Suite'. http://microservices.ece.cornell.edu/ (accessed Oct. 03, 2021).

[104] P. Datta, P. Kumar, T. Morris, M. Grace, A. Rahmati, and A. Bates, 'Valve: Securing Function Workflows on Serverless Computing Platforms', in *Proceedings of The Web Conference 2020*, Taipei Taiwan, Apr. 2020, pp. 939–950. doi: 10.1145/3366423.3380173.

[105] S. Brenner and R. Kapitza, 'Trust more, serverless', in *Proceedings of the 12th ACM International Conference on Systems and Storage*, New York, NY, USA, May 2019, pp. 33–43. doi: 10.1145/3319647.3325825.

[106] 'The Serverless Application Framework | Serverless.com', *serverless*. https://serverless.com/ (accessed Apr. 27, 2021).

[107] *serverless/serverless on GitHub*. Serverless, 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/serverless/serverless

[108] *Kubeless Serverless Plugin*. Serverless, 2021. Accessed: Nov. 10, 2021. [Online]. Available: https://github.com/serverless/serverless-kubeless

[109] 'AWS Serverless Application Repository - Amazon Web Services', *Amazon Web Services, Inc.* https://aws.amazon.com/serverless/serverlessrepo/ (accessed Apr. 27, 2021).

[110] 'IBM Cloud Functions - Overview', Feb. 11, 2021. https://www.ibm.com/cloud/functions (accessed Apr. 26, 2021).