Serverless Edge Computing: Vision and Challenges

Mohammad S. Aslanpour *
Faculty of Information Technology,
Monash University
CSIRO's DATA61
Australia
mohammad.aslanpour@monash.edu

Adel N. Toosi†
Faculty of Information Technology,
Monash University
Australia
adel.n.toosi@monash.edu

Claudio Cicconetti‡
IIT – National Research Council
Italy
c.cicconetti@iit.cnr.it

Bahman Javadi§
Western Sydney University
Australia
b.javadi@westernsydney.edu.au

Peter Sbarski¶
A Cloud Guru
Australia
pete@acloud.guru

Davide Taibi‖
Tampere University
Finland
davide.taibi@tuni.fi

Marcos Assuncao**
Inria, ENS de Lyon
France
marcos.dias.de.assuncao@ens-lyon.fr

Sukhpal Singh Gill††
Queen Mary University of London
United Kingdom
s.s.gill@qmul.ac.uk

Raj Gaire††
CSIRO's Data61
Australia
raj.gaire@data61.csiro.au

Schahram Dustdar††
Distributed Systems Group, Vienna
University of Technology
Austria
dustdar@dsg.tuwien.ac.at

ABSTRACT

Born from a need for a pure "pay-per-use" model and highly scalable platform, the "Serverless" paradigm emerged and has the potential to become a dominant way of building cloud applications. Although it was originally designed for cloud environments, Serverless is finding its position in the Edge Computing landscape, aiming to bring computational resources closer to the data source. That is, Serverless is crossing cloud borders to assess its merits in Edge computing, whose principal partner will be the Internet of Things (IoT) applications. This move sounds promising as Serverless brings particular benefits such as eliminating always-on services causing high electricity usage, for instance. However, the community is still hesitant to uptake Serverless Edge Computing because of the cloud-driven design of current Serverless platforms, and distinctive characteristics of edge landscape and IoT applications. In this paper, we evaluate both sides to shed light on the Serverless new territory. Our in-depth analysis promotes a broad vision for bringing Serverless to the Edge Computing. It also issues major challenges for Serverless to be met before entering Edge computing.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ACSW '21, February 1–5, 2021, Dunedin, New Zealand
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8956-3/21/02…$15.00
https://doi.org/10.1145/3437378.3444367

ACM Reference Format:
Mohammad S. Aslanpour, Adel N. Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar. 2021. Serverless Edge Computing: Vision and Challenges. In *Australasian Computer Science Week Multiconference (ACSW '21), February 1–5, 2021, Dunedin, New Zealand*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3437378.3444367

1 INTRODUCTION

The emergence of cloud computing was a huge step towards the paradigm of computing as a utility. On-demand resources along with seamless auto-scaling drastically reduced the management costs and improved the Quality of Service (QoS) experienced by end-users [7]. Nonetheless, there were burdens to accomplish a pure pay-per-use model and effortless scalability. Customers had to pay based on the allocated resources, not the resources actually consumed. Scalability was the problem of the customers, they had to configure autoscalers based on their load profile and the characteristics of their application; otherwise, they had to rely on general-purpose auto-scalers provided by cloud offerings, which by necessity could not be efficient for all kinds of (complex) deployments. The industry and research community were not silent and decided to remove those burdens.

On the development side, advances were rapid by moving from monolithic to service-oriented and then microservices application architecture. This move recognized the possibility of running small pieces of code as functions, leading to today Function-as-a-Service (FaaS) [8]. On the deployment side also the community was actively

investigating ways to relieve the burden carried by heavyweight virtualization, hence containerization came to accomplish it [22]. Well-timed, Serverless appeared. Serverless attended the reunion between technological advances such as microservices, FaaS [42], event-driven programming, containerization, and the idea of pure pay-per-use model. Performing well in cloud, it is expected that "Serverless will dominate the future of cloud computing" [22].

Demonstrating common features with the requirements of Internet of Things (IoT) applications, now the adaptation of Serverless in edge computing has attracted special attention both from public cloud providers and academia [4]. Edge computing refers to leveraging computation-enabled devices located at the edge of network. Devices can be conventional computer servers or sensor/actuator IoT devices augmented with computations such as Single Board Computers (SBCs), deemed as edge nodes, or a hybrid of both [28]. Although in its infancy, as one of its ultimate goals, edge computing is intended to enable computing on resource-limited devices and to reduce latency and bandwidth for IoT applications, as the biggest stakeholders [16, 28].

The main questions of this research include: "Is Serverless adaptable to edge computing? and how?" In principle, its scale-to-zero property (i.e., unused containers are deallocated from the platform) suits very well energy-aware IoT use cases with intermittent applications, while fine-grained scaling (i.e., at a function level) may easily accommodate the vastly heterogeneous requirements and execution environments at the edge. In fact, many IoT applications are based on events triggered through sensing/actuating, which is similar to the way functions in Serverless are triggered, and quite often they sense/actuate only occasionally while sleeping most of the time, instead of being always-on and wasting the energy, which is inline with the promise of ephemeral functions in Serverless, too [8]. Thus, at a first glance, Serverless looks like a near-perfect execution model. However, this is challenging due to the fact that Serverless has been originally designed for cloud environments which do not suffer from the limitations as computation devices at the edge [28].

Motivated by this potential, we identified the key gaps of integrating Serverless, edge computing and IoT applications. This is important to be filled as the reality is that although this adaptation appears essential and effective, its feasibility demands thorough investigations to avoid repercussions. Our goal is to fill this gap by expressing our early thoughts on identifying potential opportunities and steps which are needed to be taken to accomplish Serverless edge computing.

Our methodology is based on experts' collaboration and literature review. We brought together a group of active and expert researchers and practitioners in Serverless, edge computing and IoT domains to share their knowledge and critical thoughts on the topic. The result and key contributions include: a (1) simplified paradigm followed by a comprehensive list of (2) opportunities and (3) challenges facing Serverless edge computing identified and agreed by these experts.

The reminder of the paper is structured as follows. After giving a broad background on Serverless and edge computing in Section 2, we demonstrate the expected Serverless edge computing paradigm in Section 3. Following that, we categorize and discuss the opportunities as well as open issues in realizing the paradigm in Sections 4 and 5, respectively. We draw the final conclusions in Section 6.

2 BACKGROUND

2.1 Serverless

After the emergence of on-demand services in cloud, a pure pay-per-use model along with effortless scalability were missing. Customers had to implement their own auto-scaling methods, unless they decided to rely on general-purpose auto-scalers provided by cloud offerings. Further, immature monolithic application architecture and heavyweight virtualization were imposing substantial burdens.

Meanwhile, service-oriented architectures were evolving towards what we call *microservices*, i.e., small loosely coupled systems that are easier to deploy, manage, and monitor. Following that, decomposing applications into small pieces of codes acting as the basic unit of computation called *functions* became easier and led to today's *FaaS* [42]. This whitebox approach allows the controller to optimize intermediate subsets of the application, which is usually not possible in monolithic application deployments. Yet, the dependency and interaction between functions can exist and addressed by function composition practices. Functions can be composed to create more complex services [39]. Following this, *Event-driven programming* found its position in microservices to further enable fine-grained development and interaction of microservices, respectively. Due to their event-driven nature and subsequently distributed components, the necessity of *automation* and *CI/CD* (i.e., continuous integration and continuous deployment) were recognized as well. Finally, while the true potential in cgroup mechanism in Linux for isolating processes and resources was unseen for years, it became the starting point to realize today's *containerization*. Containers, led by Docker, enabled implementation of microservices as they became more popular.

Nonetheless, a technology to combine all theses advances and to fulfill a pure pay-per-use and effortless scalability was still missing, until *Serverless Computing* came into the picture. Serverless is one step forward in the abstraction staircase from Infrastructure-as-a-Service (IaaS) to Platform-as-a-Service (PaaS), since it offers customers a platform that allows the execution of software without providing any notion of the underlying Operating System (OS), not even in a virtualized manner, VM or container alike [34]. Therefore, not only the burden of monitoring the servers, but also that of managing the life-cycle of VMs and their images, is entirely assigned to the Serverless provider. In other words, Serverless attended the reunion between technological advances such as microservices, FaaS [42], event-driven programming, containerization, and the idea of pure pay-per-use model along with effortless scalability [34].

After its announcement by Amazon, the AWS Lambda gained considerable attention and following that other big IT companies took their Serverless platforms to the market. Microsoft, Google, IBM and Oracle announced Azure Functions, Google Cloud Functions, IBM Cloud Functions and Oracle Fn, respectively to penetrate into the growing market. Given the strong trends towards open collaboration, soon IBM (with OpenWhisk) and Oracle (with Fn project) released their open-source platform. Following that, Amazon Web Services (AWS) did so and currently their innovative Serverless platform, which is running microVMs by Firecracker, is open-sourced. The research community has also been actively developing easy-to-use platforms such as OpenFaaS, Fission, Kubeless, Knative, IronFunctions, etc.

To compare and draw an analogy between these projects, one can refer to their support for (1) variety in programming languages, (2) function triggering methods (e.g., HTTP, MQTT or cloud event), (3) cost and free tier offers, (4) resource provisioning and scaling specifications (e.g., CPU, memory or both), (5) function composition and communication patterns, and (6) resource abstractions or virtualization (e.g., containers or microVMs).

Serverless appears as a new technology for several enterprises to employ. The efficient deployment and organization of FaaS might not be easily available. Understanding this obstacle, Serverless frameworks such as SAM, Chalice, Serverless.com, etc. have appeared to help. Apart from this, the feasibility of deploying open-source platforms on the edge is highly debated due to their complexity and edge nodes' limitations [35], leading existing platforms to make themselves further portable and easy to deploy. An example of such portability is *faasd*, where OpenFaaS released a lighter version whose seamless implementation on SBCs such as Raspberry Pis is verified[1]. Along with these activities, cloud providers are also offering Serverless at the edge by services such as AWS Greengrass or Lambda@Edge provided by AWS.

2.2 Edge Computing

With the ever-increasing growth of IoT applications, the community was required to devote extra effort to address their requirements. Requirements include, but not limited to: low latency, real-time execution, event-driven developments, and efficient deployments [3]. Cloud computing with powerful computations could be the first available option, but cannot seem to satisfy all IoT requirements such as low latency due to being far away from data sources [7]. With IoT requirements and cloud struggle, the edge computing (or so-called fog computing) landscape is proposed as an complementary option [17]. With edge, the computation is closer to data source and real-time execution will be closer to accomplishment. Of course, edge resources are limited, but edge nodes will benefit from the distributed nature of edge to share resources and provide an ample pool of resources. Most importantly, IoT default sensing/actuating functionalities can expand to computation as well, enabling them to act as edge nodes. SBCs like Raspberry Pis, Beagles, PandaBoard, or even NVIDIA Jetson Nano are perfect examples of such advances.

Nonetheless, the potential benefits of edge computing can be obtained only if efficiently designed, implemented and deployed. Otherwise, energy- and computation-limited edge nodes are extremely vulnerable [28]. Firstly, if designed to rely on a central node while in a distributed network of IoT devices, it will fail, as edge computing is disposed to unwanted failures while powerful cloud data centers would not suffer from such constraints [28]. Secondly, if implemented using heavyweight resource abstractions such as VMs which occupy huge amount of available resources and containerization is ignored, it will fail to properly function and scale due to lack of space and lazy scalability, respectively which were not so problematic in cloud; Thirdly, if deployed as always-on applications on energy-limited edge nodes, the promise of edge computing will not probably fulfilled, and running in a event-driven manner on cloud may be even more efficient.

Overall, edge computing facilitates the process of shift from an cloud-only paradigm for IoT applications to a fully distributed cloud to edge continuum landscape. Motivated by edge-side challenges and above-mentioned characteristics of Serverless, it appears the integration of Serverless and edge computing for IoT applications will be an effective solution which will be discussed in the following section.

[1]https://blog.alexellis.io/faasd-for-lightweight-serverless/

3 SERVERLESS EDGE COMPUTING PARADIGM

Despite agreed signals on adapting Serverless-enabled edge computing, estimations show that, by 2017, IoT constituted less than 6% of Serverless use cases[2], and by 2020 it does not exceed beyond 10% [14]. Given the exponential growth of IoT market during recent years, such a slow adaptation and limited share of Serverless appears suffering from serious obstacles and ignorance of required changes in the perspective. It also seems that the question posed by Baldini et al. [5] in 2017 has remained unanswered yet, "Does serverless extend beyond traditional cloud platforms?" After Baldini's alert [5], feasibility of Serverless for IoT applications has incoherently been examined. Examples include: real-time data analytic [32], Transportation Planning [21, 38] and data processing [37], to name a few. In industry sector, the first commercial endeavour was Lambda@Edge offered by AWS in mid-2017, but critics [6] argue that the concept of edge is not fully accomplished by coarse grained distribution of AWS edge nodes (i.e., only 77 locations worldwide at the time of writing).

Motivated by this, a simplified architectural overview of Serverless edge computing is demonstrated in Fig. 1. With the literature efforts in mind [1, 4, 5, 10, 22, 23, 25, 34, 41], the figure provides essential components of Serverless and their placement in the edge as well as cloud. In brief, from edge view, IoT devices are in connection with edge nodes, each of which enabled with the whole Serverless functionality which is fully or partially shared across the edge network, depending on the considered implementation. From Serverless view, the key functionality of Serverless in this paradigm is to provide an efficient event processing platform in edge computing [5]. From IoT view, less adaptation is needed and it only has to follow function-based FaaS principles. And from cloud view, the original Serverless in the central cloud is intended to integrate itself with its distributed partner in edge computing. Under the hood, the life-cycle of an event can be illustrated as follows.

3.0.1 Edge and IoT Clients. Edge nodes will resemble a cloud data center characteristics, i.e., enabling computation, communication and storage. However, edge nodes are recognized to be extremely smaller, in greater numbers and fully distributed [28], which add the complexity to the control and management of the edge side.

IoT devices generate tasks by means of sensors and communication protocols such as HTTP and Application Programming Interfaces (APIs) by practicing event-driven principles. Given the distributed nature of IoT applications, such as sensors scattered across the urban area to monitor the transportation system in Smart City case, and the necessity of broadcasting messages, HTTP might not be always the practical solution. Hence, the Serverless-enabled edge is intended to support publish/subscribe communication protocols such as MQTT and DDS as well [12].

[2]https://www.serverless.com/blog/2018-serverless-community-survey-huge-growth-usage

Technically, if an IoT device itself has computation capability, for example a Raspberry Pi with interfaced sensors, it can act also as an edge node. Certain edge computing paradigms such as mist computing, or extreme edge, would follow this design [43].

3.0.2 Edge Network. Via the above-mentioned protocols the events are generated in IoT side and then edge nodes (as gateways or front-end devices) are in charge of handling events by triggering functions in the Serverless platform and returning appropriate actuation as another event if necessary. The Serverless platform is distributed over the edge nodes so that each node can handle the whole life-cycle of an incoming event or can offload to peers. In case of offloading, the initial node may return the event reply to the IoT layer or leave this responsibility to the remote node.

3.0.3 Serverless Platform. The Serverless platform consists of three main layers: *coordination*, *execution* and *computation*.

Coordination: The handler edge node will pass the event and its associated data such as authentication, API identifier, etc. to the *controller* component, bringing Serverless into gear. The *controller* acts as a load balancer. If the admitted event message is valid and the incoming load is not heavy, it is passed to the *scheduler* component, which based on the the triggers and rules sends the message to corresponding *execution engine* (or so-called worker [23]) associated to a function [5]; otherwise, it is kept in the *queue* component for certain reasons such as failure handling and mitigating message loss and in another iteration it is passed to the *scheduler*. The concept of queue here is deemed to keep messages for no more than a brief of milliseconds.

Execution: Once the message is handed to the *execution engine*, the engine acts on provisioning run-time environment for the corresponding function, either by reusing an already instantiated, but idle, function or by creating a new instance of the function. The core auto-scaler is also embedded in this component. Note that Serverless platforms would keep the function instance alive and avoid instant termination for a short period to serve upcoming events. The *monitoring* component is continuously babysitting the life-cycle of incoming and outgoing events to log the observations. The latest updates are kept in the *monitoring* component, and other layers, from computation to gateway, can interact with the monitoring component to collect required data.

Computation: To create a new instance, the requested computing resources are allocated by the *computing* component. The data, the function's state and the required libraries for executing a task in the functions are pulled from the *storage* component. To provision a new function, the *computing* component on an edge node will be able to interact with peers for offloading purposes as well. This is enabled by the *networking* component which is aware of peer's location (to provide server affinity), capacity, energy status, etc. The *monitor* component of each node is replying to the *computing* component of a demanding node. This offloading is essential, as the Serverless platform is intended to work collaboratively and in a distributed manner. Edge nodes build a pool of shared resources to compensate limited resources on each node.

3.0.4 Cloud Data Center. The Serverless paradigm can be also augmented with a cloud layer where they can seamlessly collaborate to share resources and execute functions by adopting a synchronized distributed computation layer. An orchestration for the computation layer may also be feasible (preferably located at the edge) acting as a gateway between edge and cloud executions. If fault tolerance is crucial, this component can be kept in the cloud. The cloud side can meanwhile serve traditional Serverless clients.

Overall: The Serverless edge computing paradigm creates many new possibilities and opportunities which are discussed in Section 4. The challenge to realize this paradigm, however, is that we need to overcome issues such as resource limitations on the edge nodes, complexity of executing multiple and dependent functions (and also function compositions), and distributed nature of edge nodes, to name a few. These and other open issues to realize Serverless edge computing are critically discussed in Section 5. Fig. 2 shows a taxonomy of identified opportunities and open issues.

4 OPPORTUNITIES

4.1 Pure Pay-per-use

Unpredictable workload in IoT applications appears inconsistent with static container provisioning which imposes charges even during idle times. Of course, dynamic auto-scaling is feasible to alleviate, but extra charges still exist, or QoS is endangered, due to highly likely imprecise over- and under-provisioning, respectively [7]. This is further problematic when the application, as well as consistent workload, is likely to encounter large spikes, wherein bulking up the capacity appears the only viable option, but at a high cost. By Serverless, however, the charge is driven by actual triggered events, i.e., the dedicated resource and number of times a function is triggered. By Serverless, fulfilling the price predictability for IoT is closer to reality, where the same (and minimal) rate is always paid per events/transactions, so no matter it is undergoing a spiky or consistent workload. Assuming a traffic control camera which is taking photos of offensive cars' plate and sending it to cloud for image processing. Such actions will happen consistently during the day, spiky in rush hours and occasionally at mid-night, making the resource provisioning highly challenging. If this processing is handled by precise scalability of Serverless which avoids over-provisioning and pay for idle times, such variability will not affect the imposed cost.

4.2 Always-on Mitigation

IoT use cases such as Smart Irrigation Systems are intended to operate on resource constraint devices in terms of power and computation, due to the environmental-related challenges in deployments. Their power will likely be supplied by batteries or renewable energy sources such as solar panels and computations, as well as orchestration will be handled by Single-board Computers (SBCs) such as Raspberry Pis. Add to this that they are running applications which periodically, not consistently, run to collect data such as soil humidity. Such limitations will not be consistent with conventional deployments which need the application to be always up and running, even when no event occurred. Also, this approach will always occupy certain amount of unused resources. Serverless with the scale to zero feature will provide the opportunity for edge nodes to save their limited resources and increase the lifetime [36]. Serverless can minimize the energy consumption by only provisioning when a trigger such as sensing is pulled and then de-provisioning

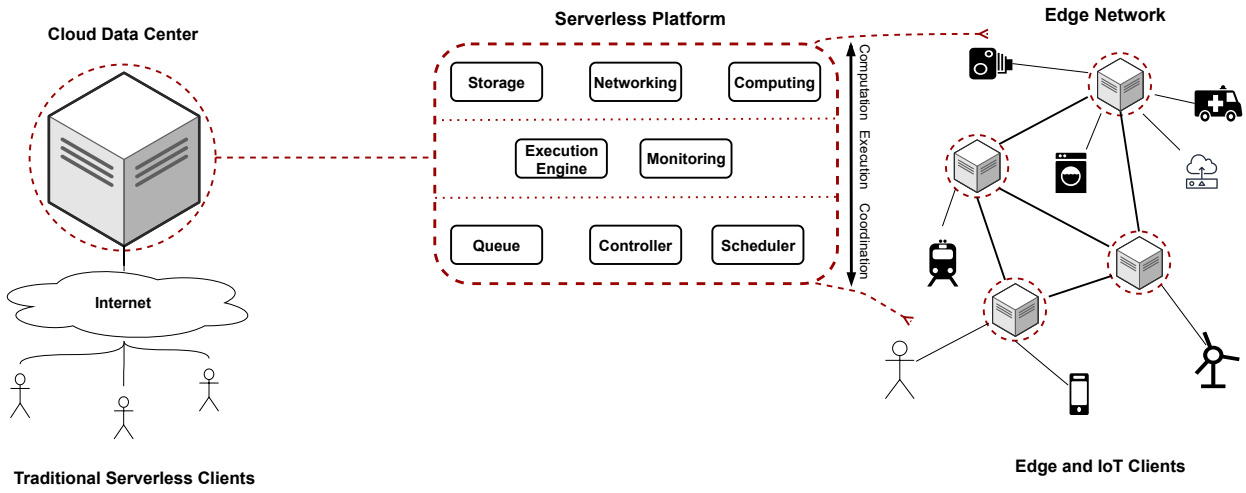

Figure 1: A simplified Serverless Edge Computing Paradigm.

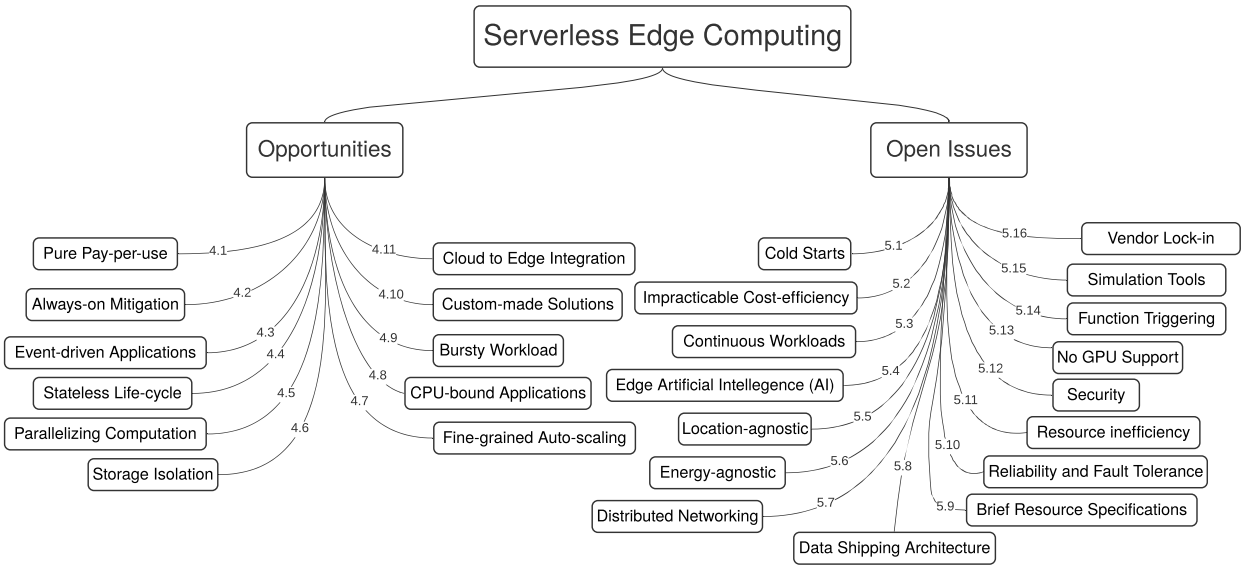

Figure 2: A Taxonomy of identified opportunities and open issues for Serverless Edge Computing.

idle resources [10]. This advantage of Serverless provides another view of its suitability for event-driven applications.

4.3 Event-driven Applications

Given the event-based life of sensor and actuation-based IoT applications, Serverless can perfectly suit them. In typical cloud services, e.g., back-end systems for mobile applications, events such as API calls and data storage are playing a key role in function invocation. On the other hand, most IoT applications are event-driven. In Smart Agricultural Farming, for instance, the decision-making about farm management is a perfect example of a sensing-driven application; the management system is triggered by temperature and humidity variability, otherwise it does nothing. In Serverless terms, this means that the sensor can make a preliminary decision on whether to trigger a chain of function invocations, based on local data acquisition; if not triggered, there are zero resources consumed on the Serverless platform. In another example involving both the cloud and an edge system, the remote sensor might always trigger the execution of a given function on the edge system (with humidity/temperature data), then the latter can take the decision on whether to escalate to the management system by triggering a function in the cloud platform or not. In all cases Serverless's design fits very well the sense-decide-act pattern, which is typical of many IoT applications of practical interest [10, 42].

4.4 Stateless Life-cycle

In edge designed for event-driven IoT applications, the invoked functions are running tasks such as image processing which essentially have not, and would not require, the knowledge of past occurrences. In a Road Vehicle Monitoring System, the monitor is interested in calling an image provisioning function to analyze the license plate number in an image input. This function will not require any knowledge of past tasks and only loads repetitive code and libraries, as if making it from scratch. Stateless functions have only one task to do, as IoT events demand. Statelessness is hence fundamentally transformable to the edge by means of Serverless. What can accelerate this transformation is the underlying resource abstraction in Serverless which rely on containers and unikernels. Such abstractions are purposely built to be stateless [31].

4.5 Parallelizing Computation

With the ability to execute independently, the ability of Serverless functions to parallelize computation is added value. By themselves, functions are usually quite under-powered, but if the problem is decomposited and processing is parallelized, this is where another advantage of Serverless for parallelized IoT use cases appears, especially if the problem space is suitable to this kind of decomposition. Take Defense IoT (DIoT) and particularly its edge computing enabled Equipment Tracking Systems as an example [16]. Once a hazard is perceived, it is crucial to quickly trigger an event to obtain the perception of equipment being tracked such as cars, ambulance, tactical devices, etc., before any action. Such equipment will need to run independent processing and generate their own response quickly while they might be running mission-related tasks. Without parallelizm, mission-critical IoT applications are highly likely to fail to react promptly. Serverless-enabled equipment will leverage decomposition of such tasks and hence parallelism thereof. Serverless is further practical here as the scalability of invoked function will come into the play when multiple similar alarms are received by peers.

4.6 Storage Isolation

IoT applications such as Smart Traffic Monitoring are continuously collecting data. In case of bursty workload, a massive amount of data is shipped with the application to be scaled up in conventional deployments. This approach can be a hassle to timely launch the new instance, regardless of storage wastage. This is further problematic when the machine is provisioned from remote devices (i.e., task offloading), demanding to ship the data and imposing further traffic congestion. Serverless provides decoupled computation and storage which can be scaled and provisioned separately and hence quickly [22].

4.7 Fine-grained Auto-scaling

One of the biggest challenges in edge devices is resource limitations. Obviously, conventional virtual machines were not a perfect solution due to large memory footprint, and difficult scalability (i.e., creating more replicas of the same service on a device) which comes at the expense of duplicating large amount of data, including the operating system and application environment. Relying on lightweight abstractions, including containers and, more recently, unikernels [31] such as microVM[3], Serverless will meet small footprint and fine-grained auto-scaling, because the overhead of creating/terminating replicas is very small compared to full-fledged virtual machines. This is further promising when Serverless is following function as computation principles inherited from the advances in microservices architecture, instead of considering the whole application as a black box [1, 8].

4.8 CPU-bound Applications

To deploy a function in Serverless environments, the amount of CPU required for each instance is pre-defined. Hence, a Serverless-enabled platform is always aware of the CPU requirements of an application. Backed by automatic scalability, this CPU specification has made the Serverless well-suited for CPU-bound applications in clouds [5]. IoT use cases driven by CPU-intensive functionalities can leverage this enabler as well to avoid unnecessary resource (e.g., memory or data) provisioning in scaling practices. Self-driving cars employing a Pedestrian Monitoring System can be a perfect example. The system functionality is to trigger events containing images of the surrounding area which need to be processed using the edge device embedded in the car (e.g., Engine Control Unit-ECU). The processing is to detect pedestrians who might cross the street, or trajectories of other vehicles that may eventually collide with us, etc. Such CPU-bound applications can effectively be made Serverless-enabled and embedded in edge nodes so that they are able to proportionally scale up/down the CPU-bound functions.

4.9 Bursty Workloads

In terms of workload type, 81% of Serverless use cases constitute bursty, i.e., spiky or flash crowd, workloads [13]. This is mostly due to effortless and fine-grained scalability embedded in Serverless, which avoids long latencies and provides precise resource provisioning, respectively. Motivated by this, IoT use cases with potential to encounter bursty workloads will fit very well with Serverless deployments. Use cases such as traffic or crowd control systems are continuing monitoring and are exposed to bursty events. Failing to timely reacting can cause an important event to be unnoticed [33], for instance. Traditional IaaS/PaaS systems may only handle bursts by overprovisioning the resources, e.g., in terms of service replicas always active, because their auto-scale functions may yield a latency that is incompatible with the application requirements. This is aggravated in edge domains, because edge nodes have more limited resources than their cloud counterparts, hence overprovisioning is even more undesirable there. Serverless brings significant benefits to use cases subjects to flash crowd effects since it allows to exploit statistical multiplexing of bursty workloads through scale-to-zero during idle times, and fast up-scale at peaks. Given that, edge computing orchestrators will utilize Serverless functions, instead of lazy heavyweight virtualization and inefficient resource reservation solutions to handle such burstiness.

4.10 Custom-made Solutions

Edge frameworks and their requirements vary depending on the IoT use case, environmental limitations, locality requirements, etc.

[3] https://firecracker-microvm.github.io

With this in mind, edge nodes provided by public cloud offerings with limited geographically distributed locations will not satisfy the need for purpose-built edge frameworks in the near term. For instance, while transmitting data between agricultural facilities is already challenging, reaching out the cloud-hosted Serverless can be an even bigger challenge. Fortunately, custom-made Serverless solutions such as OpenFaaS[4] can remove the burden for Serverless at the edge whereby no interaction to inaccessible cloud is required. Custom-made solutions are also adapting themselves with resource-limited edge nodes, with *faasd* as a lightweight platform which even works without Kubernetes orchestrator. Estimations also confirm their satisfactory performance, where they are involved in 53% of Serverless use cases, gaining more popularity than cloud offerings [14].

4.11 Cloud to Edge Integration

While certain IoT use cases tend to stay isolated at the edge of network such as Smart Farming, several use cases require seamless integration and interaction with cloud as well. This is to enable data persistent or access infinite-computation, to name a few [28], and particularly essential for mobile edge computing which demands both cloud and edge computations, for example for task offloading purposes [36]. Existing Serverless platforms, although originally designed for cloud, are now adopting themselves to this requirement and enable a Serverless-enabled edge platform integrated with cloud [15]. A successful example is AWS IoT Greengrass[5] which runs Lambda Functions under the hood. This opportunity will help clients executing their tasks on edge nodes until there are resources available, then it starts offloading to the cloud. Transition between edge and cloud is designed to be transparent to the client. The system is also robust to intermittent cloud connectivity. The integration is no longer an obstacle for Serverless to be adapted in edge computing.

5 OPEN ISSUES

5.1 Cold Starts

Although it is generally considered a successful achievement in the cloud landscape, scale to zero technique and subsequent cold start of Serverless functions are not suited for some latency-sensitive IoT applications [13]. Said that, scale to zero appears a double-edged sword. On the positive side, it reduces energy consumption. On the other side, the first invocation of the function will sustain a cold startup which imposes tens or hundreds milliseconds of delays with current technologies. This delay is not an issue for batch applications which are intended to resize an image stored in the storage, for instance. However, such delays can have performance degradation for latency-sensitive IoT applications managed by an embedded edge device. For instance, an edge device in a driverless car surely prefers to reach timely decisions (e.g., activating the break pedal) to avoid accidents than to save energy, which is abundant in an electric car and, thus, its reduction is deemed a secondary requirement. Hence, the cold start must be addressed before adopting Serverless Edge Computing if high latency (or jitter) avoidance is a must. The reason for such popularity of Serverless despite the cold startup bottleneck appears that the today customers are concerned about the cost, not latency. Estimations show that only 2% of Serverless use cases are intended for real-time applications [14]. The research community, however, has not been silent and proposed many promising solutions such as warm starts [1], function pinging [25], pre-loading critical packages for the container, utilizing further agile unikernels (or Web Assembly-based resources [19]) instead of containers [1], and overhead reduction in development phase [42], to name a few.

5.2 Impracticable Cost-efficiency

Cloud-domain estimations [14] revealed that a huge share of Serverless popularity comes from its cost-efficiency (41%) rather than its performance (23%), which is compelling with its pure pay-as-use model. However, (1) mission-critical IoT use cases such as healthcare require high performance to predict significant incidents before they actually occur. (2) Edge computing frameworks are intended to be purpose-built, hence likely to adopt open-source and custom-made solutions which would not come behind monetary cost matters. This also may reduce the role of cloud offerings in the future market, thereby eliminating the monetary matters in public cloud offerings. For instance, a private Serverless-enabled farm with local resources will not pay money to themselves. Instead, their deployment concerns will probably shift from cost-efficiency in cloud to high performance in edge.

5.3 Continuous Workloads

Although event-driven execution applications are identified as suitable use cases of IoT to run Serverlessly, there are several IoT use cases encountering continuous executions such as traffic control systems. The cost-efficiency of Serverless for such use cases is under doubt [1]. Assume that the edge node has to continuously respond to incoming workload and no idle times are expected. Such executions will continuously call functions. The reality is that cost-efficiency of Serverless will not be accomplished, but degraded, by continuous function invocations. Investigations shows that, always-on resources will work more efficient for continuous executions [1], until more efficient resource scheduling schemes are found for Serverless platforms [27]. The cost-inefficiency becomes further apparent if the functions are long-running, as the billing is based on both allocated resource and execution time. Regardless of the cost, the obtained latency will also increase due to stateless executions. The statelessness causes a resource provisioner to look for new resources for an instance creation per execution.

5.4 Edge Artificial Intelligence (AI)

Edge AI is a booming area for predictive IoT solutions by utilizing machine learning at the edge [11], whether central or federated learning. With long-running AI tasks, utilizing cloud offering Serverless will not be cost-efficient, as investigations show that both training and testing machine learning at the edge with computation-limited resources will be I/O intensive and hence time-consuming [20]. Edge AI processing will involve significant I/O-intensive long-running tasks while Serverless is intended to work well for CPU-bound short-running CPU-bound functions. Lack of awareness appears deep, though. In fact, while I/O-intensive applications are deemed unworthy of delivering to Serverless, studies

[4] https://www.openfaas.com/
[5] https://aws.amazon.com/greengrass/

show that they comprise the majority of Serverless use cases, even more than CPU-bound [14].

5.5 Location-agnostic

In the edge landscape, the devices will be distributed across the environment and communicate with each other. Minimizing the communication overhead has always been a controversial issue and efforts are needed to make the edge nodes aware of their offloading decisions, for instance [22]. Serverless platforms, though, care more about resource customizability and are agnostic to the network communications. In the cloud, a logically centralized controller deciding on resource provisioning on unlimited resources, but at the edge, this can be fully distributed and task offloading or load distribution exist actually. Moreover, due to the statelessness, executions deprive new instances of past locations of peers and dismiss server affinity. Assume that the edge node decides to offload the resource provisioning for a function to peers, in favour of energy saving or due to lack of available resources. In this case, the execution time, communication overhead and energy consumption for running the function is highly dependent on the edge node undertaking the function. Function composition (i.e., interaction between functions) can also make resource optimization even more complex. Serverless appears not to be designed for such considerations, whereas location-awareness resource provisioning is essential in many edge applications [18].

5.6 Energy-agnostic

One of the main promises of Serverless is resource-efficiency by fine-grained embedded scalability in cloud. Cloud offerings are conscious of subscribed CPU and memory on the underlying Serverless platform. However, when it comes to the edge, energy and delay are matter of concern, and even the most important in particular IoT use cases such as Smart Farming which are driven by battery-operated sensors and struggle with their limited lifetime. To adopt Serverless at the edge, energy-aware function provisioning is essential, otherwise the Serverless-enabled edge device might sacrifice the energy in the interest of finding edge nodes with less provisioned CPU and memory. It also seems that cloud-driven performance metrics such as scalability, concurrent requests and boot time are needed to be revisited when Serverless travels to edge computing [30].

5.7 Distributed Networking

Given the distributed nature of many IoT applications, the search for peer-to-peer communications to share state, offload tasks, and send data is important. For instance, a vehicular ad-hoc network (VANET) requires interactions between vehicles to find out the best routing, demanding remote function calls on peers. However, cloud-based designs for Serverless are dismissing this necessity, resulting in inefficient function communications. With present Serverless architecture, functions have to communicate through intermediary functions or storage systems [38]. Fortunately, this issue appears temporary, as projects such as Serverless Supercomputing will explore it [9].

5.8 Data-shipping Architecture

Regardless of the networking limitations, data and storage are to be taken care of in Serverless edge computing. Serverless would more emphasize on CPU and memory specifications and would disregard storage provisioning techniques which are essential in the distributed edge. Stateless performance followed by lack of server affinity would provoke the problem. In cloud also, the applications maintain the (remote) state by storing in storage like DynamoDB, but this state must be maintained closer in the edge case. This weakness can make Serverless-enabled edge devices send huge amount of data per function invocation to the remote device [20]. This barrier is seen also when functions are interacting with each other as function composability: for instance, if an application consists of a chain of invocations of functions from the initial input from the user to the final output, the transfer cost is paid only once in case of a monolithic application in a VM/container, but may have to be paid multiple times with serverless, i.e., between any two consecutive function invocations if each is executed on a different (edge) node. In cloud, such limitations are seamlessly handled by powerful data centers and high speed communications, but in distributed edge, such data shipping is problematic, particularly when it comes to mist computing [28]. Mist refers to a fully distributed edge network without a central controller. As possible solutions, it is suggested that the data transmission must be dependent on all CPU, RAM and data requirements. If the CPU on an edge device is oversubscribed, the processing can move to data, or if a huge amount of memory needs to be read, the processing can move to the data location [2].

5.9 Brief Resources Specifications

AWS Lambda as the pioneer revealed that each invoked function can execute a task for up to 15 minutes. McGrath et al. [30] also raised the question that "Is function termination time of 15 minutes efficient?" Although regarded quite sufficient for IoT with event-driven and short-running tasks, yet particular use cases such as data analytics may not be adopted with this limitation. Oppositely, keeping a function warm for 15 minutes, for instance, which is followed by cloud providers may not be always necessary. Additionally, function size and concurrency level may prove cumbersome when it comes to the edge [40]. Fortunately, this will not encounter a major obstacle as open-source solutions will ease the burden since they will allow full configuration to adapt the platform to the specific use case. The open source community in this area is vibrant and it is making huge progresses towards the goal of customized serverless platforms for all needs. For instance, OpenFaaS, already mentioned above, is now offering a lighter version as faasd which can run smoothly even on extremely resource-limited edge devices such as Raspberry Pis.

5.10 Reliability and Fault Tolerance

Failures always exist, whether in cloud or edge, but the difference is at the degree of reliability. In cloud data centers, such occurrences are handled by multiple layers of underlying virtualization and continuous and fine monitoring of servers and applications. In the edge, however, fault rolerance techniques are very limited and we envision more robust solutions which have to be defined to achieve the desired level of resiliency. Palade et al. [35] raise the concern that existing Serverless platforms would not sufficiently care about fault tolerance, which is critical in mission-critical IoT applications [16], for instance. If a function's container failed, or

more seriously, if a function's edge node failed, how the lack of that is covered by Serverless platforms? Given the real-time nature IoT applications, not handling such failures can cause a disaster.

5.11 Resource Inefficiency

In IoT domain, efficient and agile resources to be provisioned, booted-up and ran are essential for real-time and latency-sensitive use cases such as Fleet Monitoring in DIoT [16]. The computation-limited nature of edge nodes also doubles the challenge [7]. This is because while cloud benefits from infinite and powerful resources, the occurrence of resource saturation is highly likely in edge [28]. Leveraging container provisioning was a big step towards efficiency where edge devices such as SBCs are unable to launch heavyweight hypervisor-based machines. Containers' negative side, though, is that although a purpose-built container itself cannot be heavy, pre-requisites such as Docker engines are still heavy, particularly when the Serverless platform is launched on top thereof. To alleviate the boot-up time, unikernels are proposing effective solutions by omitting the intermediary layer such as Docker engine. The microVM introduced by AWS Firecracker[6] can realize reasonable boot-up time (e.g., 125ms) and efficient memory footprint, although the latter and problematic orchestration thereof are debated in [29]. Having said that, not only virtualization, but also the Serverless platforms are inevitably needed to be more agile.

5.12 Security

In the cloud, Serverless is backed by the firewall and Trusted Execution Environment (TEE) platforms [24], while in vulnerable edge network the devices are exposed [18]. The distributed nature of microservices, as well as expected multi-tenancy, make it further vulnerable, particularly when functions are deployed on containers which exhibit weaker isolation than VMs. There are attempts to reasonably isolate microservices by developing unikernel or microVMs [29]. The latter is particularly designed to ensure an isolated environment by combining technologies such as *cgroups*, *namespaces*, *seccomp-bpf*, *iptables*, and *chroot*. Moreover, given the functions are deemed to be distributed across the edge network, the transmission pattern between them can make them further vulnerable, although their ephemerality will stop continuous data leakage at least [22]. Having said that, security guarantees are still necessary to be given prior to enabling Serverless at the edge.

5.13 No GPU Support

Most commercial Serverless platforms do not offer APIs that allow the offloading of computation to specialized hardware, such as GPUs and FPGAs, that might be more efficient to the task at hand. This limitation makes it impossible to adopt Serverless edge for GPU-based real-time streaming applications, such as Smart Supply Chain Systems, which are equipped with real-time information and monitor the entire supply chain, from suppliers to distributors and retails [16]. Serverless architecture appears to need revisiting for such sectors, especially at the edge, by extending the programming paradigms so as to be more comprehensive towards different hardware targets, while keeping the same level of abstraction that we have today. For commercial systems, this might also require differentiated pricing plans and more flexible billing and accounting schemes compared to what are available today.

[6] https://firecracker-microvm.github.io

5.14 Function Triggering

Cloud-driven Serverless manages the function triggering through HTTP APIs, cloud events, scheduled events, etc, with HTTP as the widely-popular type with 46% share [14]. The reality is that HTTP APIs are of request/reply family of communication patterns while in IoT the subscribe/reply pattern (e.g., MQTT) is more widely adopted. Cloud offerings and open-source platforms such as AWS GreenGrass and KubeEdge[7], respectively, have recognized this necessity and are filling this gap.

5.15 Simulation Tools

Serverless applications include many moving parts, that cannot be tested locally [26]. Testing a Serverless application on the edge requires extra effort and new practices to test the system in production. The analysis of log tracing is a possible approach [26]. Given that, simulation tools providing the testing and evaluation of Serverless edge computing are essential to alleviate the complexities.

5.16 Vendor lock-in

Adopting a Serverless-based solution with one vendor makes it very difficult to switch other vendors. As an example, the adoption of vendor-specific message buses, API-Gateways or other native technologies, increase even more the vendor lock-in [41].

6 CONCLUSIONS

Serverless computing has established its position in cloud computing and is known as a cost-efficient, agile, low footprint solution for short-running applications. However, its suitability for edge computing driven by IoT applications is yet to be proven. In this paper, we carefully analyzed the advantages of bringing Serverless to the edge and potential obstacles for such accomplishments.

After a thorough evaluation of both sides, we can admire the suitability of Serverless in terms of (1) offering pure pay-per-use, (2) mitigating always-on resource provisioning, (3) consistency with event-driven nature of IoT, (4) easy to parallelize stateless functions, (5) fulfilling storage isolation, (6) fine-grained scalability for resource-limited edge devices, (7) moving inline with bursty workloads and CPU-bound applications, (8) existing custom-made platforms, and (9) viability of cloud to edge integration.

However, our evaluation voices serious challenges about the feasibility of Serverless edge computing with today's technology in terms of (1) cold startup provoking long latencies, (2) impractical cost-efficiency designed for cloud, (3) unsuitability for continuous workloads and edge AI applications along with no support for GPU considerations (4) lack of location- and-energy-awareness, (5) unconsidered distributed networking, (6) inefficient data shipping, (7) brief resource specification, (8) reliability and fault tolerance concerns, (9) possibility of provoking resource inefficiency, (10) security concerns, (11) immature function triggering and (12) lack of simulation tools.

[7] https://kubeedge.io/en/

We believe Serverless will bring additional benefits to edge computing if we can overcome these challenges, which requires joint efforts from academia, industry, and the relevant open source communities.

REFERENCES

[1] Paarijaat Aditya, Istemi Ekin Akkus, Andre Beck, Ruichuan Chen, Volker Hilt, Ivica Rimac, Klaus Satzke, and Manuel Stein. 2019. Will Serverless Computing Revolutionize NFV? *Proc. IEEE* 107, 4 (2019), 667–678.

[2] Zaid Al-Ali, Sepideh Goodarzy, Ethan Hunter, Sangtae Ha, Richard Han, Eric Keller, and Eric Rozner. 2018. Making serverless computing more serverless. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 456–459.

[3] Mohammad S. Aslanpour, Sukhpal Singh Gill, and Adel N. Toosi. 2020. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet of Things* 12 (2020), 100273. https://doi.org/10.1016/j.iot.2020.100273

[4] Javadi Bahman, Sun Jingtao, and Ranjan Rajiv. 2020. Serverless architecture for edge computing. In *Edge Computing: Models, technologies and applications*. Institution of Engineering and Technology, 249–264. https://doi.org/10.1049/pbpc033e_ch12

[5] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, and Aleksander Slominski. 2017. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*. Springer, 1–20.

[6] Luciano Baresi, Danilo Filgueira Mendonça, and Martin Garriga. 2017. Empowering low-latency applications through a serverless edge computing architecture. In *European Conference on Service-Oriented and Cloud Computing*. Springer, 196–210.

[7] Rajkumar Buyya, Satish Narayana Srirama, Giuliano Casale, Rodrigo Calheiros, Yogesh Simmhan, Blesson Varghese, Erol Gelenbe, Bahman Javadi, Luis Miguel Vaquero, and Marco A S Netto. 2018. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–38.

[8] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. The rise of serverless computing. *Commun. ACM* 62, 12 (2019), 44–54.

[9] Ryan Chard, Tyler J Skluzacek, Zhuozhao Li, Yadu Babuji, Anna Woodard, Ben Blaiszik, Steven Tuecke, Ian Foster, and Kyle Chard. 2019. Serverless supercomputing: High performance function as a service for science. *arXiv preprint arXiv:1908.04907* (2019).

[10] Claudio Cicconetti, Marco Conti, Andrea Passarella, and Dario Sabella. 2020. Toward Distributed Computing Environments with Serverless Solutions in Edge Systems. *IEEE Communications Magazine* 58, 3 (2020), 40–46.

[11] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. 2020. Edge intelligence: the confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal* (2020).

[12] Jasenka Dizdarević, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. 2019. A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. *ACM Computing Surveys (CSUR)* 51, 6 (2019), 1–29.

[13] Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina Abad, and Alexandru Iosup. 2020. Serverless Applications: Why, When, and How? *IEEE Software* (2020).

[14] Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L Abad, and Alexandru Iosup. 2020. A review of serverless use cases and their characteristics. *arXiv preprint arXiv:2008.11110* (2020).

[15] Nabil El Ioini, David Hästbacka, Claus Pahl, and Davide Taibi. 2020. Platforms for Serverless at Edge: A Review. In *1st International Workshop on Edge Migration and Architecture*.

[16] Paula Fraga-Lamas, Tiago M Fernández-Caramés, Manuel Suárez-Albela, Luis Castedo, and Miguel González-López. 2016. A review on internet of things for defense and public safety. *Sensors* 16, 10 (2016), 1644.

[17] Mostafa Ghobaei-Arani, Alireza Souri, and Ali A Rahmanian. 2020. Resource Management Approaches in Fog Computing: a Comprehensive Review. *Journal of Grid Computing* 18, 1 (2020), 1–42. https://doi.org/10.1007/s10723-019-09491-1

[18] Alex Glikson, Stefan Nastic, and Schahram Dustdar. 2017. Deviceless edge computing: extending serverless computing to the edge of the network. In *Proceedings of the 10th ACM International Systems and Storage Conference*. 1.

[19] Adam Hall and Umakishore Ramachandran. 2019. An execution model for serverless functions at the edge. In *Proceedings of the International Conference on Internet of Things Design and Implementation*. 225–236.

[20] Joseph M Hellerstein, Jose Faleiro, Joseph E Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2018. Serverless computing: One step forward, two steps back. *arXiv preprint arXiv:1812.03651* (2018).

[21] Luis Felipe Herrera-Quintero, Julian Camilo Vega-Alfonso, Klaus Bodo Albert Banse, and Eduardo Carrillo Zambrano. 2018. Smart its sensor for the transportation planning based on iot approaches using serverless and microservices architecture. *IEEE Intelligent Transportation Systems Magazine* 10, 2 (2018), 17–27.

[22] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, and Neeraja Yadwadkar. 2019. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383* (2019).

[23] Young Ki Kim, M Reza HoseinyFarahabady, Young Choon Lee, and Albert Y Zomaya. 2020. Automated Fine-Grained CPU Cap Control in Serverless Computing Platform. *IEEE Transactions on Parallel and Distributed Systems* 31, 10 (2020), 2289–2301.

[24] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.

[25] Philipp Leitner, Erik Wittern, Josef Spillner, and Waldemar Hummer. 2019. A mixed-method empirical study of Function-as-a-Service software development in industrial practice. *Journal of Systems and Software* 149 (2019), 340–359.

[26] V Lenarduzzi and A Panichella. 2021. Serverless Testing: Tool Vendors and Experts Point of View. *IEEE Software* (2021), 0. https://doi.org/10.1109/MS.2020.3030803

[27] C Lin and H Khazaei. 2021. Modeling and Optimization of Performance and Cost of Serverless Applications. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (mar 2021), 615–632. https://doi.org/10.1109/TPDS.2020.3028841

[28] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2020. Application Management in Fog Computing Environments: A Taxonomy, Review and Future Directions. *ACM Comput. Surv.* 53, 4 (jul 2020). https://doi.org/10.1145/3403955

[29] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. 2017. My VM is Lighter (and Safer) than your Container. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 218–233.

[30] Garrett McGrath and Paul R Brenner. 2017. Serverless computing: Design, implementation, and performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 405–410.

[31] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jorg Ott. 2018. Consolidate IoT edge computing with lightweight virtualization. *IEEE Network* 32, 1 (2018), 102–111.

[32] Stefan Nastic, Thomas Rausch, Ognjen Scekic, Schahram Dustdar, Marjan Gusev, Bojana Koteska, Magdalena Kostoska, Boro Jakimovski, Sasko Ristov, and Radu Prodan. 2017. A serverless real-time data analytics platform for edge computing. *IEEE Internet Computing* 21, 4 (2017), 64–71.

[33] Hai Duc Nguyen, Chaojie Zhang, Zhujun Xiao, and Andrew A Chien. [n.d.]. Real-time Serverless: Cloud Resource Management for Bursty, Real-time Workloads. ([n. d.]).

[34] Jussi Nupponen and Davide Taibi. 2020. Serverless: What it is, what to do and what not to do. In *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 49–50.

[35] Andrei Palade, Aqeel Kazmi, and Siobhán Clarke. 2019. An evaluation of open source serverless computing frameworks support at the edge. In *2019 IEEE World Congress on Services (SERVICES)*, Vol. 2642. IEEE, 206–211.

[36] Duarte Pinto, João Pedro Dias, and Hugo Sereno Ferreira. 2018. Dynamic allocation of serverless functions in IoT environments. In *2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE, 1–8.

[37] R Arokia Paul Rajan. 2020. A review on serverless architectures-function as a service (FaaS) in cloud computing. *TELKOMNIKA* 18, 1 (2020), 530–537.

[38] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. 2019. Serverless Computing: A Survey of Opportunities, Challenges and Applications. arXiv:1911.01296 [cs.NI]

[39] Davide Taibi, Nabil El Ioini, Claus Pahl, and Jan Raphael Schmid Niederkofler. 2020. Patterns for Serverless Functions (Function-as-a-Service): A Multivocal Literature Review. In *10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*. 181–192.

[40] Davide Taibi, Nabil El Ioini, Claus Pahl, and Jan Raphael Schmid Niederkofler. 2020. Serverless Cloud Computing (Function-as-a-Service) Patterns: A Multivocal Literature Review. In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER'20)*.

[41] Davide Taibi, J. Spillner, and K. Wawruch. 2021. Serverless Where are we now and where are we heading? *IEEE Software* 38, 1 (2021).

[42] Erwin Van Eyk, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Uță, and Alexandru Iosup. 2018. Serverless is more: From paas to present cloud computing. *IEEE Internet Computing* 22, 5 (2018), 8–17.

[43] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. 2019. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture* 98 (2019), 289–330.