Serverless Computing: A Survey of Opportunities, Challenges, and Applications

HOSSEIN SHAFIEI, K. N. Toosi University of Technology, Iran
AHMAD KHONSARI, University of Tehran, Iran
PAYAM MOUSAVI, University of Tehran, Iran

The emerging serverless computing paradigm has attracted attention from both academia and industry. This paradigm brings benefits such as less operational complexity, a pay-as-you-go pricing model, and an auto-scaling feature. The paradigm opens up new opportunities and challenges for cloud application developers. In this paper, we present a comprehensive overview of the past development as well as the recent advances in research areas related to serverless computing. First, we survey serverless applications introduced in the literature and summarize the challenges they have faced. We categorize applications in eight domains and separately discuss the objectives and the viability of the serverless paradigm in each of those domains. We then classify those challenges into nine topics and survey the proposed solutions for each of them. Finally, we present the areas that need further attention from the research community and identify open problems.

CCS Concepts: • **Computer systems organization → Cloud computing**.

Additional Key Words and Phrases: Cloud Services, Serverless Computing, Function-as-a-Service (FaaS)

ACM Reference Format:
Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. 2021. Serverless Computing: A Survey of Opportunities, Challenges, and Applications. *ACM Comput. Surv.* 1, 1 (June 2021), 27 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Large technology companies such as Amazon, Google, and Microsoft offer serverless platforms under various brand names. Although the specifics of the services may differ the essential idea behind the offered services is almost the same i.e. by rendering computation to the pay-as-you-go model, serverless computing tries to achieve auto-scaling while providing affordable computation services [82].

Serverless computing differs from traditional cloud computing concepts (we refer to them as serverful in this paper) in the sense that the infrastructure and the platforms in which the services are running are hidden from customers. In this approach, the customers are only concerned with the desired functionality of their application and the rest is delegated to the service provider.

Authors' addresses: Hossein Shafiei, shafiei@kntu.ac.ir, shafiei@kntu.ac.ir, K. N. Toosi University of Technology, P.O. Box 15875-4416, Tehran, Iran, 43017-6221; Ahmad Khonsari, University of Tehran, Tehran, Iran, ak@ipm.ir; Payam Mousavi, University of Tehran, Tehran, Iran, pa.mousavi@ut.ac.ir.

0360-0300/2021/6-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

There are successful commercial implementations of this model. Amazon introduced Lambda [1] in 2014 and later Google Cloud Functions [2], Microsoft Azure Functions [3] and IBM OpenWhisk [4] were launched in 2016. Since then, many studies have focused on the challenges and open problems of this concept. Some of the previous studies are skeptical about the potentials of serverless computing due to the poor performance of their case studies [66]. In contrast, others believe that serverless computing will become the face of cloud computing and the performance issues will be addressed eventually [82].

The aim of the serverless services is threefold: (1) relieve the users of cloud services from dealing with the infrastructures or the platforms, (2) convert the billing model to the pay-as-you-go model, (3) auto-scale the service per customers' demand. As a result in a truly serverless application, the execution infrastructure is hidden from the customer and the customer only pays for the resources they actually use. The service is designed such that it can handle request surges rapidly by scaling automatically. The basic entities in serverless computing are functions. The customer registers their functions in the service provider. Then, those functions can be invoked either by an event or per users' request. The execution results are sent back to the customer. The invocation of the functions is delegated to one of the available computation nodes inside the service provider. Usually, these nodes are cloud containers such as Docker [100] or an isolated runtime environment [67].

Though the concept of serverless computing is relatively new it has paved its way into many real-world applications ranging from online collaboration tools to the Internet of Things (IoT). We survey the papers that introduce real-world serverless applications and categorize them into eight different domains. We summarize the objectives (as justified by the authors) for migrating to serverless services for each of the application domains. We further assess the aptness of the serverless paradigm for each application domain based on the arguments made by the authors, the obtained results, and the challenges they reported. Table 1 lists the application domains, migration objectives, and assessments.

There are several challenges that serverless services are currently facing. There exist some surveys and literature reviews that discuss those challenges [30, 39, 41, 82, 148]. Our approach is different, in that, instead of focusing on the inherent obstacles and shortcomings of the concept, we analyze the challenges reported in each of the surveyed application domains. Then, we categorize and discuss the existing solutions for those challenges. We bring out the areas that need further attention from the research community. We also discuss the limitations of those solutions and identify open problems. Some of the challenges surveyed in this paper are common between various application domains such as the topic of providing security and privacy in serverless services. And some of them are domain-specific, such as the issues of scheduling, pricing, caching, provider management, and function invocation.

In this paper, we also discuss the opportunities presented by serverless computing. We emphasize that serverless services are more customer-friendly as they relieve customers from the intricacies of deployment. They are also more affordable in some cloud computing scenarios that we will discuss later in this paper. We argue that new market places are emerging around these services, which implies new business opportunities.

The rest of this paper is organized as follows. Section 2 presents definitions and characteristics of serverless services. Section 3 focuses on the opportunities that the serverless computing model offers. Section 4 discusses the application domains. Section 5 surveys the challenges toward the vast adoption of the concept. Section 6 concludes the paper.

[1] https://aws.amazon.com/lambda/
[2] https://cloud.google.com/functions/
[3] https://azure.microsoft.com/
[4] https://openwhisk.apache.org/

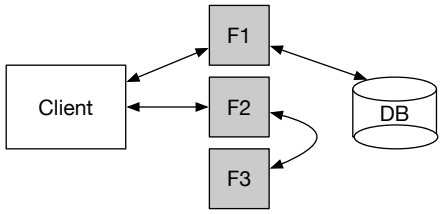

Fig. 1. An example of a serverless application.

2 DEFINITION AND CHARACTERISTICS

2.1 Definitions

There is no formal definition for the concept of serverless computing and its various services. So, here we present the most commonly acknowledged definitions.

2.1.1 FaaS. Function as a service (FaaS) is a paradigm in which the customers are able to develop, run, and manage application functionalities without the trouble of building and maintaining the infrastructure.

2.1.2 BaaS. Backend as a Service (BaaS) is an online service that handles a specific task over the cloud, such as authentication or notification. Both BaaS and FaaS require no resource management from the customers. While FaaS only offers to execute users' functions, BaaS offers a complete online service.

2.1.3 Serverless service. A serverless service can be viewed as a generalization of FaaS and BaaS that incorporates the following characteristics:

(1) The execution environment should be hidden from the customer i.e. the computation node, the virtual machine, the container, its operating system and etc. are all hidden from the customer.
(2) The provider[5] should provide an auto-scaling service i.e. the resources should be made available to the customer instantly per demand.
(3) The billing mechanism should only reflect the number of resources the customer actually uses i.e. pay-as-you-go billing model.
(4) The provider does its best effort to complete the customer's task as soon as it receives the request and the execution duration is bounded.
(5) The basic elements in serverless services are functions. The functions are not hidden from the provider. The provider knows their dependencies to external libraries, run-time environments, and state during and after execution.

A serverless application is usually comprised of two parts:

- A client [6]. The client implements most of the application logic. It interacts with two sides i.e. the end-user and the provider, invoking functions on one side and translating the results into usable views for the other side.
- Registered functions on a provider. Functions are uploaded to the provider. The provider invokes a copy of the function according to the user's request or based on a predefined event.

[5]Throughout this paper by a service provider we mean the organization (or company) that serves customers in a serverless fashion.
[6]Note that, the client part is not essential in every serverless scenario i.e. often functions are directly called by other functions or they are triggered by an external event

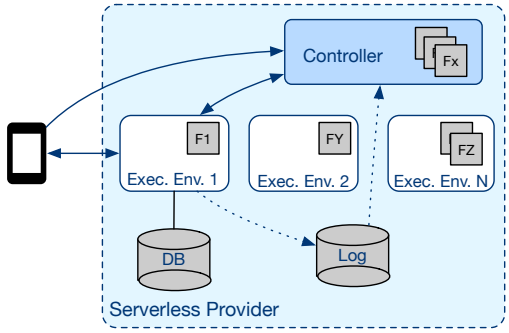

Fig. 2. An example of a invocation scenario in a serverless provider. Note that, this figure only depicts the concept and the actual implementation may differ

Figure 1 depicts an example of a serverless application. In this example, the client invokes function F1 which retrieves data from the database, performs some operations on it, and sends the result back to the client. In another scenario, the client invokes F2, F2 invokes another function called F3, the result is sent back to F2 and then it sends its result to the client. In this scenario, we can see a sequence of function executions. Later in this paper, we argue that these sequences are important for performance optimization.

Providers accept customers' functions and store them. Upon receiving an invocation request, the provider assigns the function to one of its computation nodes for execution. The parameters upon which the provider selects the execution node have a great impact on the performance of the system which is thoroughly surveyed in Section 5 of this paper. The node (which can be a virtual machine, a container, or any sandbox execution environment) executes the function, sends back the result to the client, and sends the execution log to the provider. The provider can use these logs to improve further execution of the function. Figure 2 shows an example of such a scenario.

3 OPPORTUNITIES

In this section, we discuss the opportunities that serverless computing offers.

3.1 No deployment and maintenance complexity

The very first and foremost opportunity that serverless computing offers is to relieve users from managing the infrastructure, which is now already accomplished in Infrastructure-as-a-Service (IaaS), however, users still have to manage their virtual resources i.e, installing and configuring related packages and libraries. Certainly, Platform-as-a-Service (PaaS) providers such as Heroku [13] have made the management slightly easier, although, users still have to configure the application to match the PaaS requirements which is not a trivial task. Serverless computing takes a big step in this manner. Users only have to register their functions and then receive the credentials to invoke the functions.

3.2 Affordable scalability

Another promise of the cloud computing is the ability for customers to deploy the functionalities of their applications without worrying about the scalability of the execution infrastructure or platform. The scalability is a direct result of the auto-scaling nature of these services i.e. per request the service invokes a copy of the requested function and there is virtually no bound for the number

Table 1. Serverless application domains

Application	Main reason	Assessment
Real-time Collaboration and Analytics	Auto-scaling feature	Promising
Urban and Industrial Management Systems	Pricing model	Promising
Scientific Computing	Lower deployment overhead	Fair
Artificial Intelligence and Machine Learning	Pricing model	Fair
Video Processing and Streaming	Lower deployment overhead	Fair
System and Software Security	Auto-scaling feature	Promising
Internet of Things (IoT)	Auto-scaling feature	Promising
E-commerce, Banking and Blockchains	Auto-scaling feature	Fair

of concurrent requests. Each invocation is assigned to the most feasible and available resource for execution.

The affordability of serverless services is mainly due to the reduced costs of the providers. There is two main reasons for the more reduced costs, (1) resource multiplexing and (2) infrastructure heterogeneity. Resource multiplexing leads to higher utilization of available resources. For example, consider the case in which an application has one request every minute and it takes milliseconds to complete each request. In this case, the mean CPU usage is very low. If the application is deployed to a dedicated machine then this is highly inefficient. Many similar applications could all share that one machine. Infrastructure heterogeneity means that the providers also can use their older machines that are less attractive for other direct services to reduce their costs (mainly because the execution environment is a black-box for the customer).

It is important to note that the affordability of serverless services depends on the usage scenario i.e. in some cases renting a virtual machine is cheaper than using serverless's pay-as-you-go model. Lin et al. [93] developed an economic model to establish a tradeoff between pricing of virtual machines and serverless services. Their study characterizes scenarios that the customers can benefit from switching to serverless services.

3.3 New market places

With the advent of modern operating systems for mobile devices such as Android or iOS, various market places for applications, that are specifically designed for those operating systems, have emerged such as Google Play Store [12] and Apple's App Store [6]. Such a scenario is already appearing for serverless paradigm i.e. with the growth in the popularity of serverless computing, new market places for functions has emerged. In these types of markets, developers can sell their developed functions to others. Every generalized or domain-specific functionality can be bought or offered in those markets. For example, a software developer may need a geospatial function that checks whether a point resides inside a geospatial polygon. They could buy such functions from those markets. AWS Serverless Application Repository [4] is an example of such a capability. [124] presents a quantitative analysis of functions available inside AWS Application Repository.

The competition forced by the economics of these markets will lead to high-quality functions i.e. both from the perspective of code efficiency, cleanness, documentation, and resource usage. The function markets may present buyers with a catalog for every function which shows the resource usage of the function and prices it incurs per request. Thus, the buyer can choose from many options for a specific task.

4 APPLICATIONS

Many real-world serverless applications have been proposed in the literature during the past few years. We categorize these applications into eight domains. Table 1 lists the application domains, their related papers, the main reason for migration stated in the papers, and assessments based on the arguments made by the authors, the obtained results, and the challenges they reported. In what follows, we survey these application domains, in detail.

4.1 Real-time collaboration and analytics

The stateless nature of serverless services makes them an attractive platform for real-time collaboration tools such as instant messaging and chatbots. Yan et.al., [153] proposed an architecture for chatbot on OpenWhisk [15]. An XMPP-based serverless approach for instant messaging is also introduced in [117]. Real-time tracking is another example of collaboration tools that are very suitable for serverless services as these applications are not heavily dependant on the system's state. Anand et.al., [23, 24] proposed two real-time GPS tracking methods on low-power processors.

Serverless services are also utilized for data analytics applications [104]. In these applications, various sources stream real-time data to a serverless service. The service gathers, analyses, and then represents the data analytics. The auto-scaling feature of serverless computing makes the handling of concurrent massive data streams, possible. Müller et. al [102] proposed Lambada which is a serverless data analytics approach that is one order of magnitude faster and two orders of magnitude cheaper compared to commercial Query-as-a-Service systems.

4.2 Urban and industrial management systems

The pay-as-you-go model of serverless services paved the way for the introduction and implementation of various budget-restricted urban and industrial management systems. Al-Masri et.al., [20] presented an urban smart waste management system. Hussain et.al., [76] proposed a serverless service for oil and gas field management system. An implementation of a serverless GIS platform for land valuation is presented in [101].

The distributed nature and auto-scaling feature of serverless services make it an apt choice for smart grids. Zhang et. al, [157] proposed event-driven serverless services to handle SCADA/EMS failure events. A distributed data aggregation and analytics approach for smart grids is proposed in [73]. Serverless services have been also utilized for urban disaster recovery applications. Franz et.al., [60] proposed a community formation method after disasters using serverless services. Another similar approach is also proposed in [46].

The migration toward serverless paradigm seems a reasonable choice for this domain of applications, especially, for public sector services or for developing countries due to its lower deployment overheads and also its pay-as-you-go pricing model.

4.3 Scientific computing

It has been debated in [66] that serverless computing is not an attractive alternative for scientific computing applications, albeit, many studies have focused their attention toward serverless services for those applications. We believe disagreement lies in the fact that the range of scientific computing and its applications are vast and there are certainly some areas in this domain for which the utilization of serverless services is feasible.

Spillner et.al., [125] argue that serverless approaches provide a more efficient platform for scientific and high-performance computing by presenting various prototypes and their respective measurements. This idea is also echoed in [42] where high-performance Function-as-a-Service is proposed for scientific applications. A serverless tool for linear algebra problems is proposed in

[119] and a case for matrix multiplication is presented in [143]. Serverless paradigm is harnessed for large-scale optimization in [28]. A serverless case study for scientific workflows is discussed in [98].

Serverless approaches have been also used in DNA and RNA computing [74, 90]. Niu et. al, [106] utilized the potentials of serverless paradigm in all-against-all pairwise comparison among all unique human proteins. On-demand high-performance serverless infrastructures and approaches for biomedical computing are proposed in [88].

Scientific applications that require extensive fine-grained communication are difficult to support with a serverless approach, whereas those that have limited or coarse-grained communication are good candidates. Also, note that scientific computations with time-varying resource demands will benefit from migrating to a serverless paradigm.

4.4 Artificial intelligence and machine learning

Machine learning in general and neural network-based learning, in particular, are currently one of the most attractive research trends. The suitability of the serverless paradigm for this domain has received mixed reactions both from research and industrial communities. For example, it has been argued that deep learning functions are tightly coupled (they require extensive communication between functions), and also these functions are usually compute and memory intensive, as such, the paradigm is not promising for these applications [57]. Nevertheless, it has been discussed that deep neural networks can benefit from serverless paradigms as they allow users to decompose complex model training into several functions without managing virtual machines or servers [150]. As such, various such approaches have been proposed in the literature. A case of serverless machine learning is discussed in [40]. Ishakian et al., [78] discussed various deep learning models for serverless platforms. Neural network training of serverless services is explored in [57]. Also, a pay-per-request deployment of neural network models using serverless services is discussed in [135]. A prototype serverless implementation for the estimation of double machine learning models is presented in [89]. A distributed machine learning using serverless architecture also discussed in [138].

Christidis et al., [44] introduce a set of optimization techniques for transforming a generic artificial intelligence codebase to serverless environments. Using realistic workloads of the UK rail network, they showed that by wielding their techniques the response time remained constant, even as the database scales up to 250 million entries. A serverless framework for the life-cycle management of machine learning-based data analytics tasks is also introduced in [34].

The viability of serverless as a mainstream model serving platform for data science applications is studied in [149]. The authors presented several practical recommendations for data scientists on how to use the serverless paradigm more efficiently on various existing serverless platforms.

4.5 Video processing and streaming

Serverless approaches have been proposed for video processing. Sprocket [25] is a serverless video processing framework that exploits intra-video parallelism to achieve low latency and low cost. The authors claim that a video with 1,000-way concurrency using Amazon Lambda on a full-length HD movie costs about $3 per hour of processed video. In another interesting work, a serverless framework for auto-tuning video pipelines discussed in [116]. It achieves 7.9 times lower latency and 17.2 times cost reduction on average compared to that of serverful alternatives. In [114] GPU processing power is harnessed in a serverless setting for video processing. Zhang et al., [156] present a measurement study to extract contributing factors such as the execution duration and monetary cost of serverless video processing approaches. They reported that the performance of video processing applications could be affected by the underlying infrastructure.

Serverless video processing and broadcasting applications have gained much traction both from industrial and research communities during the COVID-19 pandemic. A live media streaming in a serverless setting is presented in [87]. A serverless face-mask detection approach is discussed in [140]. Serverless paradigm also has been utilized in video surveillance applications. Elordi et al. [55] proposed an on-demand serverless video surveillance using deep neural networks.

4.6 System and software security

The power of serverless computing has been leveraged for providing security for various software systems and infrastructures. A mechanism for securing Linux containers has been proposed in [35]. Serverless services have also been utilized for intrusion detection. StreamAlert [103] is a serverless, real-time intrusion detection engine built upon Amazon Lambda. Birman et al. [36] presented a serverless malware detection approach using deep learning.

Serverless approaches have been also used for ensuring data security. A method for automatically securing sensitive data in the public cloud using serverless architectures has been introduced in [58]. Hong et al. [69] presented six different serverless design patterns to build security services in the cloud.

We believe that the serverless approach has great potentials to improve the security of systems and services. This is due to various reasons:

(1) Security threats are often ad-hoc in nature. In these cases, the pay-as-you-go pricing leads to reduced costs.
(2) Some of the attacks exhibit sudden traffic bursts. The auto-scaling feature of the serverless services facilitates the handling of such a scenario.
(3) Attackers may conduct widespread attacks interrupting various components and infrastructures of the victim. Serverless functions are standalone in the sense that the functions can be executed in various execution environments.

In the opposite direction, serverless approaches have been used to develop a botnet [147]. We think that preventing attackers from using serverless infrastructures to conduct these types of attacks is an important issue that needs to be addressed.

4.7 Internet of Things (IoT)

The serverless computing paradigm has been exploited for various IoT domains. Benedetti et al. [31] conducted various experiments on IoT services to analyze the aptness of different serverless settings. Using a real-world dataset, authors of [139] showed that a serverless approach to manage IoT traffic is feasible and utilizes fewer resources than a typical serverful approach. Cheng et al. [43] propose a serverless fog computing approach to support data-centric IoT services. A smart Internet of Things (IoT) approach using the serverless and microservice architecture is proposed in [68]. A serverless body area network for e-health IoT applications is presented in [113]. A serverless IoT platform for smart farming is introduced in [134]. Serverless paradigms also have been utilized for coordination control platforms for UAV swarms [72].

In another research direction, Presson et al. [109] introduced a flexible and intuitive serverless platform for IoT. A decentralized framework for serverless edge computing in the Internet of Things is presented in [45]. The objective of the paper is to form a decentralized FaaS-like execution environment (using in-network executors) and to efficiently dispatch tasks to minimize the response times. In another interesting work, George et al. introduced Nanolambda [62] which is a framework that brings FaaS to microcontroller-based IoT devices using a Python runtime system. Amazon's Greengrass [7] also provides a serverless edge runtime for IoT applications.

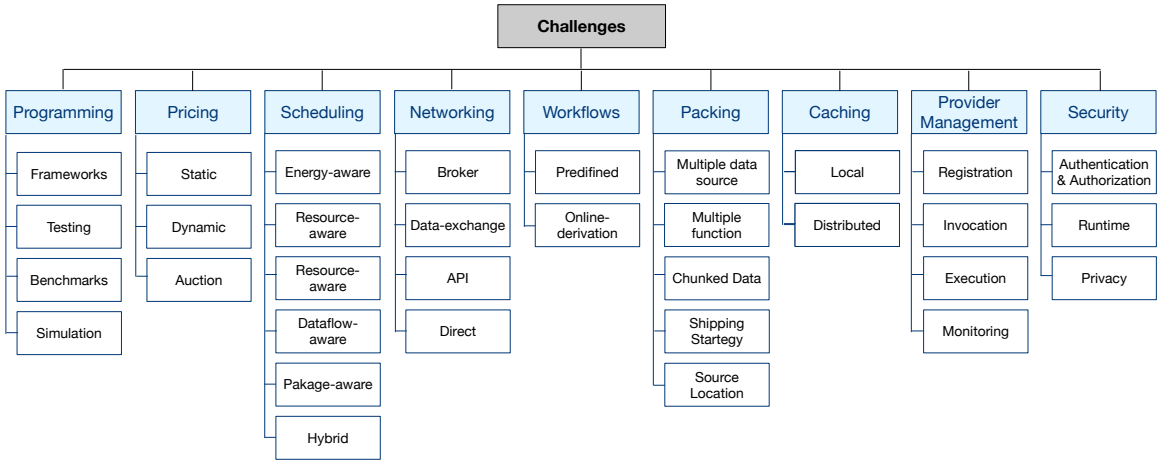

Fig. 3. An overview of the challenges discussed in this paper

It is reasonable to confer that serverless services can act as feasible back-ends for IoT applications that have infrequent and sporadic requests. For the scenarios where rapid unpredictable surges of requests emerge, serverless services can conveniently handle requests as they can auto-scale rapidly.

4.8 E-commerce, banking and crypto-currencies

The inherent scalability of serverless services has enticed few e-commerce and banking applications. A serverless implementation of a core banking system is presented in [86]. Goli et al. [63] present a case study of migrating to serverless in the FinTech industry. Huy et al. [77] implemented a crypto-currency tracking system based on OpenWhisk.

5 CHALLENGES

In this section, we summarize and discuss the challenges faced by the application domains surveyed in Sec 4. We categorize those challenges into nine topics and survey the existing solutions for each of them. Figure 3 depicts an overview of these topics. We also present the areas that need further attention from the research community and identify open problems.

5.1 Programming, modeling, testing, and debugging

As the topic of serverless computing is relatively new, its development tools, concepts, and models are not rich enough. This poses a great challenge for software developers. The lack of proper modeling paradigms leads to non-unified development approaches which will reduce the quality of the code and also complicate collaborations of developers in long term. To remedy this shortcoming, Perez et al. [107] propose a programming model and middleware for serverless computing applications. The focus of the paper is limited to file processing applications. A formal model for serverless computing using lambda calculus is presented in [61]. Also, model-based analysis of the serverless application is discussed in [144].

To enhance the adoption of a novel paradigm either new programming frameworks should be developed or existing programs on legacy frameworks should be ported to the new paradigm. Zhang et al. [159] introduced Kappa which is a programming framework for serverless computing. It facilitates and simplifies parallel programming on serverless platforms i.e. the programmer can

write ordinary Python code and Kappa transforms the code and executes it on a serverless platform using parallel lambda functions. Pywren [81] enables programmers to directly run their existing Python codes on Amazon's Lambda platform.

Debugging and testing tools are integral parts of any software development approach. Serverless computing is not an exception. Few papers have elaborated on this. Lenarduzzi et al. [91] reflect expert views on the issues and challenges that need to be investigated. [99] proposes a combined monitoring and debugging approach for serverless functions. Integration testing of serverless functions is discussed in [145].

Another important tool to test the applicability and performance of any new idea is benchmark suites. Several benchmark tools have been developed for serverless applications and service providers during the past few years. A rich set of benchmark suites for serverless infrastructures is presented in [56]. Yu et al. [155] introduced *ServerlessBench*, an open-source benchmark suite that includes many test cases to explore important metrics such as communication efficiency, startup latency, and overall performance. FaaSdom [97] is another benchmark suite for serverless computing platforms that also integrates a model to estimate budget costs for deployments across the supported providers. Another such benchmarking tool is PanOpticon [123] which also provides an array of configurable parameters and gives out performance measurements for each selected configuration and platform.

Simulation tools are also important for rapid modeling of real-world situations or testing new ideas and concepts. Mahmoudi et al. [96] introduced SimFaaS which is a performance simulator for serverless platforms. It simplifies the prediction of various serverless computing key performance metrics.

5.2 Pricing and cost prediction

Many big technology companies now offer serverless computing services with different specifications and prices. As the popularity of serverless services increases, the number of companies and their options for pricing will grow. Many factors affect the price offered by each company. These factors range from the company's revenue strategy to the platform it uses and the energy prices (based on the region or the time of day during which the function execution occurs). For example, a request that gets to a server at 2 a.m. in winter typically costs lower compared to that of the same request with the same resource consumption at 2 p.m. workday in the summer. Another factor is the load level that is imposed on the provider at that moment i.e. whether the provider nearing its peak power demand or not. Peak demand prices are reported to be 200-400 times that of the nominal rate [105]. Consequently, the contribution of peak charge in the electricity bill for a service provider can be considerable, e.g., from 20% to 80% for several Google data centers [151]. The price offered by the competitors is also a key decision factor. Various pricing models have been proposed for cloud computing in general [21] that are not directly applicable to serverless service. Extracting a pricing model for service providers is a challenging issue that should further be studied by the research community.

The pricing problem is also important for customers. As discussed above, the diversity of the prices will lead to a competitive environment between service providers. Thus, the customer can choose between various price options in an online manner to reduce the costs. In this way, customers put their functions on multiple serverless services (or ship it instantly) and then based on the online available prices, the software decides to place the request to the most affordable service provider. The ultimate goal of customers is to reduce their costs while maintaining the quality of service. Note that, one of the important factors in determining the quality of service is the response time.

Finding an optimal or a sub-optimal pricing strategy with multiple providers and customer constraints is a challenging issue that must be addressed in research studies. A similar notion has been extensively discussed for cloud computing in general, where supply and demand are considered in extracting dynamic pricing models [152].

It is noteworthy to mention that, the nature of serverless services makes online pricing more feasible compared to that of other cloud services. In those services such as IaaS, the cost of moving and maintaining several virtual machines in various service providers is higher compared to that of function placement in several serverless providers. This enables the scenario in which function placement can be done using auctions as discussed in [32].

Currently, major serverless providers only offer static pricing. This makes the task of predicting the customer costs straightforward as the costs only depend on the resource usage rather than other somewhat harder to predict parameters (such as time of use, etc.). Thus, several research studies have focused on either predicting or modeling resource usage of serverless functions. Using Monte Carlo simulation, Eismann et al. [54] proposed a methodology for the cost prediction of serverless workflows. They showed that the proposed approach can predict the response time and output parameters of a function based on its input parameters with an accuracy up to 96.1%. Cordingly et al. [47] introduced a tool for accurate performance predictions with error percentage below 4%. An analytical model using a heuristic algorithm is presented in [92]. The results show accuracy up to 98%. The latter approach is easier to implement and faster to attain as it only executes a greedy algorithm. Overall, it seems that the problem of predicting resource usage of serverless functions is tractable with high accuracy.

5.3 Scheduling

Invocation requests are sent to the provider either by customers' applications or other functions. These requests often have predefined deadlines. This is especially of great importance for real-time, latency-sensitive, or safety-critical applications. The provider must schedule where (i.e. which computation node) and when to execute the functions such that it conforms with the deadlines while considering other system-related criteria such as energy consumption, or resource utilization. There are various strategies that the scheduler could adopt. In what follows, we summarize those strategies. Note that, the real-world schedulers may adopt one or more of these strategies.

Energy-aware scheduling: The main idea in this type of scheduling is to put inactive containers or the execution environment in a hibernate mode (or cold-state mode) to reduce energy consumption. The transition from cold-state to active mode incurs delays in the execution of invoked functions which may go beyond the deadlines defined by the customer[7]. Thus, in these approaches, the executions are scheduled so that the number of such transitions minimizes. For example, Suresh et al. [128] introduced ENSURE which is a scheduler for serverless applications. To prevent cold starts, it proactively reserves a few additional containers in a warm state which can smoothly handle workload variations. Using a theoretical model they tried to minimize the amount of additional capacity while ensuring an upper bound for the request latency. Fifer [64] also uses a similar approach i.e. it proactively spawns containers to avoid cold-starts.

An energy-aware scheduler also can take advantage of delaying non-latency-sensitive tasks (such as background or maintenance tasks) to reduce overall energy consumption. We think introducing execution scheduling for a mixture of latency-sensitive and non-latency-sensitive functions can be

[7]By our estimates on a laboratory installation of OpenWhisk, the cold start latency can range from 2 to 6 seconds; by contrast, real-world open-source function executions usually take millisecond scale

a good direction for future research studies.

Resource-aware scheduling: Serverless applications are diverse, so are their resource consumption patterns. For example, a scientific computing function is usually CPU-intensive while an analytic application is often memory-intensive. Co-locating many CPU-intensive functions in a physical node leads to resource contention and may incur delays to the execution of those functions. This is also true for other types of computing resources such as memory, disk, and network. Resource-aware schedulers place functions to the computation nodes so that they can provide resource requirements of the functions in a timely manner. FnSched [127] falls into this category of serverless schedulers. it first categorizes functions based on their CPU usage. It then reduces CPU contention between co-located functions by dynamically regulating their CPU shares at runtime. In another approach, Kim et. al. [84] proposed a dynamic CPU capping method along with a group-aware scheduling algorithm for serverless computing which can effectively reduce CPU contention between functions inside computation nodes. HoseinyFarahabady et. al. [71] proposed a prediction tool to estimate the rate of event streams inside providers with the end goal of reducing CPU contention and improving QoS. A QoS-aware resource allocation scheme that dynamically scales by predicting the future rate of incoming events for serverless environments is also introduced in [70]. In another direction, Fifer [64] conducts offline profiling to calculate the expected execution time of functions and balances the load adaptively. A framework that uses Bayesian Optimization to find the optimal configuration for serverless functions is presented in [19]. It uses statistical learning techniques to gather samples and predict the cost and execution time of a serverless function across unseen configuration values. The framework uses the predicted cost and execution time, to select the best configuration parameters for running a single or a chain of functions while satisfying customer objectives.

Workflow-aware scheduling: Serverless applications usually require the execution of several functions to handle their tasks. These stateless small discrete functions are chained together and orchestrated as serverless workflows. We describe this in Sec. 5.5 comprehensively. By knowing the chain of function invocations, schedulers can speculate the next functions in the chain to forecast, schedule, and provision in advance e.g., prepare a warm container for the execution. Xanadu [50] uses such speculation to reduce the overheads and delays up to 10 times compared with OpenWhisk. Sequoia [132] and Archipelago [122] also adopt variations of this strategy. [122] uses DAG-structure[8] to predict the size of worker pools and thus achieve low scheduling overheads for request execution. Sequoia allows prioritizing, scheduling, and queuing of function chains, or functions within workflow chains in order to improve Quality-of-Service (QoS).

We think that this area has the potential for further studies. For example, probabilistic serverless DAGs discussed in [92] can be leveraged to further improve workflow-aware scheduling methods.

Dataflow-aware scheduling: Although in an ideal serverless setting the functions are stateless and do not depend on any external data sources, in practice, this is usually not the case. For example, in many machine learning applications, the dependency on external sources is high. The scheduler thus should take the availability of the data or the data serving delays into account. Hunhoff et al. [75] introduced *freshen* which allows developers or providers to proactively fetch data along with other runtime reuse features to reduce overheads when executing serverless functions. The scheduling policy of Cloudburst [126] also prioritizes data locality. Rausch et al. [112] presented a domain-specific dataflow-aware serverless scheduler for edge computing.

[8]Directed Cyclic Graphs (DAG) is usually used to show dependency between tasks in scheduling algorithms

We further investigate the data dependency in Sec. 5.6. Data caching is also of great importance to reduce the delay imposed by data dependency, we discuss this in Sec. 5.7.

Package-aware scheduling: In some of the current serverless technologies e.g., OpenLambda, the computation node should fetch and install application libraries and dependent packages declared by function upon receiving an invocation request. This obviously takes some time and delays the execution. Amumala et al. [27] proposed a package-aware scheduling scheme that addresses this issue. Other serverless platforms are usually designed such that the customers themselves incorporate those package during the registration phase.

This area of research also has potentials for further investigations. For example, machine learning approaches could be utilized to predict future needed libraries based on currently installed libraries similar to the next basket recommendation problem [29].

Hybrid scheduling: Hybrid virtual-machine/serverless scheduling also has gained attention recently. The idea is to have the best of all worlds scenario; a central scheduler decides to place requests to a private virtual machine or to a serverless provider. Spock [65] is a hybrid scheduling mechanism that flattens request peaks using VM-based auto-scaling. Skedulix [48] is also a hybrid scheduler with the objective of minimizing the cost of using public cloud infrastructures while conforming with user-specified deadlines.

5.4 Networking, sharing and intra-communications

A serverless software is typically a composition of many functions that work together to provide the desired functionality. To attain this, the functions need to somehow communicate with each other and share their data or their state. In other cloud services, this is attained through network addressing. For example in IaaS, each virtual machine can send and receive data through point-to-point networking using network addresses.

Functions intra-communication and network-level function addressing in serverless platforms are challenging. Functions in serverless services have characteristics that must be considered to be able to introduce some kind of addressing scheme for them:

- Due to the auto-scaling nature of serverless computing, at any given time there may be several running invocations of the same function inside various computation nodes around the world. This rules out the possibility of addressing based on function name or location.
- The functions are often short-lived. The short life span of the functions means that any addressing scheme should be fast enough to track the rapid changes of the system's entire state.
- With the growth in the usage of serverless services, the number of copies of functions that are being deployed will grow drastically. Thus, the proposed addressing space should be scalable enough to be able to handle that volume of functions.

Even with a proper addressing scheme, intra-communication between functions is still challenging. There are several possible approaches:

(1) Intermediate functions or external coordinators that serve as brokers between functions that are suggested in [59]. The same idea is also utilized in [137] where proxies act as coordinators. However, it has been argued in [82] that this burdens extreme overhead on the infrastructure. Performance evaluations of this approach that reveal the bottlenecks and further investigation and optimization of contributing factors, is a good direction for related research studies.
(2) Communication through data exchange is another approach. This is achievable either through data exchange mechanisms that rely on cloud storage to pass the data such as that of [102]

and [108], or using distributed message queues such as Amazon's Kinesis Data Streams [3]. There is also the possibility of tailored VM-based resources exchange mechanisms [111, 146]. This type of communication imposes an order of magnitude higher latency than point-to-point communications [141]. Introducing new fast protocols and methods for data exchange that is specifically designed for function communication in serverless environments needs researchers' attention.

(3) Communicate through APIs using stateless communication schemes (or protocols) such as REST or SOAP. These protocols are vastly used over the Internet and seem to be good candidates. Such an approach is introduced by [16] using specialized APIs and network protocols.

(4) Direct network sockets over TCP/IP is also another approach that is introduced in [141] and [133]. The idea is to enable generalized networking capabilities for functions rather than facilitating only function-to-function communications. Wawrzoniak et al. [141] proposed such a method over TCP/IP. Their benchmark shows a sustained throughput of 621 Mbit/s and a round-trip latency of less than 1 ms. In another related approach, Thomas et al. introduced Particle [133] which is an optimized network stack for serverless settings. It improves application runtime by up to 3 times over existing approaches.

5.5 Serverless workflows

As discussed earlier in this paper, a serverless application is a composition of various functions working in coordination with each other to accomplish the desired tasks. Rarely, we have applications that are composed of a single function, instead, usually, there are many interdependent functions, processing and passing data to each other and send back the result to the application. For example, real-world implementation of an online social network (with functionalities similar to Facebook [10]) on Amazon's Lambda infrastructure has around 170 functions [17]. These small functions are orchestrated into high-level workflows to form serverless applications.

In each application, functions are executed in various sequences. For example, users *sign up* in the application, view latest *products*, click on the *add-to-cart* button and *check out*. These 4 functions are executed in a sequence. Obviously, this is not the only execution sequence in the application since the user may have already signed up and just needs to *sign in*. There are many other possibilities for the sequences of functions. As we discuss in this paper, knowledge of these sequences (or chains) plays a key role in improving the performance of serverless services. Providers can use this knowledge to pre-fetch, prepare and optimize functions to reduce costs and serve customers with better performance.

From each serverless application, a directed task graph can be derived where the nodes are functions and edges show the dependency or precedence of the execution of one function to another. Figure 4 shows an example of such a graph. The application depicted in the figure has 8 functions (F1 to F8). A directed edge from F1 to F2 means that the execution of F2 depends on the execution of F1. For example, users must *sign up* to be able to *check out*. The figure shows different types of sub-structures in the task graph i.e. parallel execution, cycle, self-loop, and branch. Each of these structures must be considered and investigated inside providers to improve the system's overall performance as studied in [92].

There are two main approaches to attain workflows for a serverless application:

5.5.1 Predefined workflows. In this approach, the application developer submits a file that contains the workflow or defines it through an interface. The provider receives and processes the workflow for further actions. Many of the existing serverless providers offer this service. AWS Step Functions

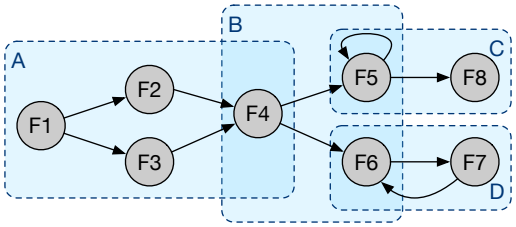

Fig. 4. An example of a task graph for a serverless application. This application has 8 functions where the start function is F1. Four different types of structures in task graphs are shown i.e. (A) a parallel execution, (B) a branch, (C) a self-loop, and (D) a cycle.

[8], Azure Durable Functions [9], Alibaba Serverless Workflow [1], and Google Cloud Composer [11] are examples of such a service.

Various approaches exist to define and specify workflows for serverless platforms. Amazon uses States Language [5] which is a language that enables users to define state machine and determine states transitions. Microsoft Azure uses standard programming language of choice (such as JavaScript, Python, C#, or PowerShell) code to represent workflows [38]. Google's composer service enables users to use Python to define workflows. To enable a platform-independent language for various existing platforms, Ristov et al. [115] introduced AFCL which is a generic language for serverless workflow specification that can be translated to multiple platforms e.g., AWS Lambda or IBM Cloud Functions. Triggerflow [95] is another interesting approach to compose event-based workflows for serverless systems.

Wen et al. [142] conducted an empirical study of these serverless workflow services. They thoroughly investigated the characteristics and performance of these services based on real-world workloads. In another related study, the performance of sequential workflows under various platforms is discussed in [38]. Domain-specific serverless workflows also have been proposed in the literature to accelerate serverless application development in that domain. SWEEP [80] which is a utility to define, execute and evaluate workflows of scientific domains, falls into this category.

5.5.2 Detection and extraction during execution. In this approach, a workflow graph is formed inside the service provider based on the activities of the application. This approach is simpler for the developer since it does not need any further workflow definition tools or services. It is also more flexible in the sense that it extracts the chains of execution based on the actual behavior of the application (these chains are described as *implicit* chains in [50]). In large applications with hundreds or even thousands of functions, there are many paths and sometimes it is not trivial to predict the exact behavior of the users to define them. Exploiting them during execution leads to more accurate estimations. This approach is also more robust than the other since the workflows defined by the users could be faulty or less accurate. We think this topic is a good direction for future research studies.

5.6 Packing

The main incentive for the migration toward a serverless service is its ability to auto-scale itself by executing copies of customers' functions and assign each request to those copies. Schedulers select the physical node to execute the functions based on various strategies that we discussed in Sec 5.3. In many real-world scenarios, functions have certain dependencies on a single remote data source or even several data sources. The data must be shipped from those sources to the computation nodes. This leads to orders of magnitude slower function execution [82] mainly because of higher

traffic inside the provider which increases the latency and reduces the overall performance of the system. Thus, it seems wise to ship the function as near as possible to the data i.e. "pack" functions with data.

To attain a robust approach for packing of functions with data, Zhang et al. [158] proposed Shredder which is a multi-tenant cloud store that allows serverless functions to be performed directly within storage nodes (i.e. packing functions to storage). However, this approach leads to complexity in provisioning and utilization as the storage and compute resources are co-located. To overcome this issue [33] propose an approach that enables users to write storage functions that are logically decoupled from storage. The storage servers then choose where to execute these functions physically using a cost model.

It is also possible to pack data and functions together. Shashidhara [120] proposed Lambda-KV, which is a framework that aggregates and transforms compute and storage resources into a single entity to enable data locality (i.e. packing functions and storage together). This approach is not suitable for the scenarios in which the data changes with high frequency e.g., IoT sensor data readings in certain scenarios. However, the method is applicable to many transaction-based applications such as that of e-commerce and banking applications or that of background applications such as video processing tasks.

Choosing between the two approaches i.e. packing data to functions or the other way is highly dependant on the application domain. As such, Kayak [154] is proposed that adaptively chooses between shipping data to functions (i.e. packing storage to functions) or vice versa. It maximizes throughput while meeting application latency requirements. There are further considerations for packing of functions that must be taken into account:

Multiple data sources and a single function: There are some scenarios in which a function consumes various data sources. The basic idea would be to ship the data sources together and then pack the function with those data sources. However, this may not be feasible due to various reasons: (1) one or more of the data sources are already near other functions that consume the data, (2) the movement is not physically possible due to the lack of sufficient storage, (3) the movement is not feasible since the number of times that functions access the data is considerably low. Finding an optimal position for the function based on the distance between the physical location of the function and its data sources on the network topology is an interesting problem that needs to be addressed.

Multiple functions and a single data source: This case is actually simpler. In this case, multiple functions are packed together with the data source in one machine. Fasslets [121] is well-suited for this scenario. It provides isolation abstraction for functions based on software-fault isolation (SFI) while enabling memory regions to be shared between functions in the same machine. Photon [52] tries to co-locate multiple instances of the same function within the same runtime to benefit from the application state and data.

Chunked data: The packing of function and data can be done with chunks of data instead of the whole data. For example, for a function that queries customers' tables of a company's database, that specific table is important and can be packed with the function instead of the whole database.

Evolutionary vs revolutionary movement of data: As mentioned above, there are scenarios in which data must be moved toward the function. This can be done in evolutionary or revolutionary modes. In the former mode, the chunks of data are moved based on requests from the function, the movement is done incrementally. This may lead to inconsistency in the data which must be taken

care of by the provider. In the latter, the data is moved altogether.

Source location: The relative geographical location of the request to the function also may play a role. Packing data and functions together and then shipping them to the nearest possible location to the requester would reduce the delays that the service faces due to network traffic.

The packing can be done before, during, or after the first execution of a function. In the case in which the packing is done before the execution of the function, a careful manifestation of data dependency is needed to find the optimum placement of the function. To this end, Tang et al. [131] proposed LambData that enables developers to declare function's data intents. In the evolutionary model, the packing is done during the execution. It can also be done after the first execution. In this case, the execution logs are inspected after the first execution and by using optimization techniques and machine learning approaches the optimal packing strategy is extracted to improve the performance of the service and to minimize costs. Note that, packing may undermine the benefits of statistical multiplexing, leading to queuing delays and inefficient resource utilization which is a good direction for future research studies.

5.7 Data caching

To avoid the bottlenecks and latencies of persistent storage, software systems utilize multiple levels of caches. This is a common practice among cloud-based applications [26]. Utilizing caches in serverless environments is a challenging issue since functions are executed independently of the infrastructure. For example, consider a serverless customer management application. When a function requests the data of a user from the database, the platform usually caches the data to handle any further requests and to reduce the number of costly database accesses. This works perfectly in serverful services. However, in a serverless service, the next execution of the function may be assigned to another available computation node, which renders the caching useless. This is also true when multiple functions consecutively work on a chunk of data i.e. if the computation node changes, the cached data becomes expired. Without caching, the costs, overheads, and latency grow dramatically which makes serverless services infeasible. Thus, this is one of the important challenges toward the successful implementation of any serverless service. Few caching mechanisms have been proposed for serverless systems. Amazon offers ElastiCache [2] which is an in-memory cache and data store. It is at least 700x more expensive than Amazon's storage service (called S3). Pu et al. [111] proposed a method to attain a cost-efficient combination of slow storage and costly in-memory caches. InfiniCache [137] is another in-memory object caching system based on stateless cloud functions. CloudBurst [126] also proposes a caching mechanism in its architecture.

We think this subject has the potentials for many further research studies. In designing and implementing caches, the following must be considered:

Effect of packing: In one of the packing schemes i.e. packing function with data, the functions are shipped as near as possible to the data. This may lead to a scenario in which multiple invocations of the same function are executed in a computation node near the data. This actually reduces the complexity of caching. Instead of focusing on a system-wide cache solution, one can focus on efficient local caching mechanisms. The action of packing also tends to ship other functions that consume the data toward the vicinity of the data, and thus with proper local caching, the chance of cache hits is improved.

Effect of workflows: The sequence upon which a batch of functions is executed also has a great impact on designing caches. In fact, in a sequential execution, the likelihood of data dependency between two or more consecutive functions is high. Thus, caching will be effective if the execution

chains are considered in the local caching strategy.

Local caching vs distributed caching In some of the real-world scenarios of serverless computing, an efficient local caching can be feasible[9]. However, there are cases in which the functions cannot be shipped to the vicinity of the data. In these cases, distributed caching can be utilized.

Distributed in-memory caches often utilize distributed hash functions (DHT) to extract the location of cached data [136]. Then, the data is routed to the requester. If the data does not reside in the distributed cache, the requester extracts the data and caches it. This works well when the cost of extracting data from its source is higher than that of getting it from the remote cache. CloudBurst [126] proposes a variation of this idea. It utilizes both DHT-like distributed storage along with local caching to improve the performance of serverless applications.

Note that, using distributed caching may incur more costs. We are facing a scenario in which the function cannot be shipped to the vicinity of the data. In this scenario, caching the data in the server that executes the function may accelerate the future invocations of the same function. However, as most other functions are shipped near the data, the cost of routing the cached data to the functions compared to that of extracting it from the source directly may actually be higher. This must be considered in any distributed cache design for serverless services.

5.8 Provider Management

The management operation inside the serverless providers is a complex and resource-demanding task. It involves many monitoring and provisioning operations for each of the infrastructures inside the provider. The controller should handle and keep track of functions' registration, invocation, and execution. Below, we discuss each operation in detail:

Registration: Every user should be able to upload their function, select its required resources. The provider then sends back credentials for invocation. Other tracking and business aspects are handled by the provider during this step.

Invocation: A provider receives invocation requests from applications or other functions, checks the requester's credentials, and then finds a feasible computation node and assigns the function to the node for execution. The tasks of placement, scheduling, packing, and caching are part of the responsibility of the controller which is done in collaboration with the computation nodes.

Initialization and Execution: To execute functions, providers often use sandbox execution environments to provide strong isolation between function instances. OpenWhisk uses containers for the execution [15], however, containers usually have isolation problems. As such, many recent studies have focused on providing serverless sandboxes with reliable isolation. Catalyzer [51] is an example of such efforts. It provides strong isolation with sub-millisecond startup time. FireCracker [18] introduces Virtual Machine Monitor (VMM) device model and API for managing and configuring MicroVM. It provides strong isolation with minimal overhead (less than 5MB of memory) and millisecond scale boot time. Unikernels in which the function is linked with a bare minimum library operating system is another approach to attain a sandbox execution environment for serverless computing [130].

[9]Here, by local we mean a caching mechanism shared between multiple servers in a rack or possibly a cluster of racks near each other.

Monitoring: Although the execution takes place inside the computation nodes, the controller should closely monitor the execution of functions to detect errors and malfunctions. It gathers the execution logs to analyze the footprints and thus improve future invocations. Various commercial monitoring approaches exist for serverless and cloud systems such as Epsagon[10], Datadog[11], and Dynatrace[12]. These solutions usually are restricted to basic metrics such as CPU utilization. Eismann et al. [53] proposed a resource consumption monitoring module specifically tailored for serverless platforms.

There are two approaches to attain a controller system for serverless providers; centralized or distributed. While the centralized approach is more trivial and more efficient, it may experience extreme loads and it could become the single point of failure. Distributed monitoring, on the other hand, is complex and hard to implement.

For the controller to be able to handle its responsibilities, manage resources and optimize the services, it should have an online view of the entire system. Various pieces of information contribute to the formation of this view, such as:

(1) Information about the functions: their data dependency, the workflow, their owner, the origin of the requests, rate of invocations, etc.
(2) The state of the infrastructure: the location of nodes, the communication infrastructure, their online available resources, which functions are assigned to them, the execution logs, etc.
(3) The data sources: the format of data, the location of data sources, their infrastructure, etc.
(4) The state of local caches: what they have in the caches, what policy for cache they use, what is the size of their cache, etc.

Having all of the above information in an online manner incurs heavy overhead on the provider. On the other hand, having partial information may lead to imprecise decisions by the controller. This challenge deserves attention from both research and industrial communities.

5.9 Security and Privacy

Security is an indispensable concern in any computation service, be it serverless or not. Various security challenges are common between serverless and other cloud services. Precht et al. [110] reviewed some of those challenges. Here, we survey the security issues that specifically threaten the normal operation of serverless service. We also consider the privacy of users in such environments.

5.9.1 Authentication and authorization. The foremost security challenge in any serverless scheme is how to authenticate applications so that only legitimate ones can use the available functions. Without authentication, a freeloader can use the available resources of the victims. A common approach to counter these attacks is the usage of authentication tokens in the header of requests. JWT is an example of such tokens [83]. Amazon's Lambda currently implemented such a scheme that uses a bearer token authentication strategy to determine the caller's identity [14]. However, if the request is sent through an unsecured channel, the attacker could simply extract the token and reuse it in another application. Using SSL/TLS protocols, this type of threat could be handled. However, there are cases in which these sophisticated public-key-based protocols are beyond the capabilities of the application's hardware. Very low-power IoT devices are an example of such hardware. Request signing is already proposed for such scenarios which incur lower resource consumption [129]. Albeit, we think the design and implementation of security protocols with minimal energy footprint for authentication in IoT and embedded devices is a promising direction for future research.

[10] https://epsagon.com/
[11] https://www.datadoghq.com/
[12] https://www.dynatrace.com/

Serverless services are also susceptible to replay attacks. In these types of attacks, the attacker doesn't know about the content of the message being transmitted, however, they are interested in the effect of transmitting the message. So, the attacker captures the secured function execution request and replays it to sabotage the normal operation of the system. An example of such an attack is to replay a logout request indefinitely to prevent users from accessing their desired service. Detection and prevention approaches should be introduced for these types of attacks.

There is also the issue of authorization i.e. to specify which users or functions can invoke a certain function and to restrict others' access. This is different from the application-level authentication that we mentioned earlier. Here we authorize a function or a user to call another function. Lacking a proper authorization scheme can pose severe threats to the application's security. Recall that one of the envisioned advantages of serverless services, which we mentioned, is the ability to purchase off-the-shelf functions. Without authorization schemes, functions can be used without the consent of the application owner. To remedy this important issue Amazon Lambda uses a role-based access controls approach i.e. customers statically assign functions to roles that are associated with a set of permissions. The shortcoming of this approach is that it is restricted to a single function and the workflow-level access control is not considered. Sankaran et al. [118] proposed WillIam that enables identity and access control for serverless workflows.

5.9.2 Runtime Security. In the wake of Meltdown [94], and Spectre [85] attacks, the vulnerability of applications against common execution environments has become one of the main security concerns. This issue is particularly severe in serverless environments since many functions from various owners are being executed in a shared execution environment. To counter these types of attacks, a lightweight and high-performance JavaScript engine is presented in [37] that utilizes secure enclaves for the execution environments. The limitation of this work is that it suffers from high memory overhead and it only supports JavaScript. We think this is an area of research that deserves special attention from the research community.

Runtime environments also can be customized to surveil the functions during execution to detect and prevent malicious activities. SecLambda [79] introduces a modified container runtime environment called runsec. It intercepts HTTP requests and I/O operations to check whether or not the function conforms with a set of predefined security policies. Trapeze [22] also adopts the same strategy i.e. it puts each serverless function in a sandbox that intercepts all interactions between the function and the rest of the world based on policy enforcement rules defined by their dynamic information flow control model. Valve [49] proposes a serverless platform that performs runtime tracing and enforcement of information flow control. The downside of these approaches is that they have a fairly heavy impact on the performance of serverless services. More lightweight approaches need to be investigated for delay-sensitive functions.

5.9.3 Resource Exhaustion Attacks. The main focus of the attacker in these types of attacks is to over-utilize the resources of the victim to either disrupt the service or impose excessive financial/monetary loads. The victim in this type of attack can be both the service provider or the customer. An attacker may tamper with the application to send fraudulent requests to the provider. Although, the auto-scaling nature of serverless services can handle these situations, however, the load may go beyond the SLA with the provider and thus the provider may deny further requests, or at least it can impose a heavy financial load on the application owner. Monitoring approaches must be introduced for serverless providers to detect and mitigate these types of attacks.

Resource exhaustion attacks can also be established against the provider itself. These types of attacks would be particularly destructive for small to medium-sized providers. In this scenario, the attacker is familiar with the internal mechanisms of the provider or they can exploit it by studying the system's behavior. Using the knowledge, the attacker could launch a series of attacks that

disrupt the normal operation of the system by intentionally preventing any optimization effort. For example, by knowing the packing strategy used by the provider, the attacker may inject fake dependencies to other data sources to prevent the function from being shipped near the data source which imposes heavy traffic inside the network of the provider.

5.9.4 Privacy Issues. There are many privacy-sensitive applications for serverless services especially in the area of IoT. For example, in a health-care application that gathers patients' data and then processes the data to infer certain conclusions, the privacy of the users is essential. The intent of the attacker here is not to alter the normal operation of the system as opposed to security attacks. Rather, they attempt to deduce knowledge about a user or a group of users, using a minimal set of gathered information. For example, in the domain of IoT healthcare system's an attacker may be interested in answering the question of whether a user has a heart condition or not.

There is much contextual data that an attacker could gather to infer knowledge about the victim. These are especially attainable when the network only uses application-layer security protocols. Here, we list some of these contextual data that can reveal sensitive information about the victim, along with their respective examples:

- Which function is invoked. For example in a serverless surveillance system, if the function *gate-opened* is called. The attacker can deduce that someone has entered.
- The sequence of function invocation, e.g., in a health-care monitoring system, a sequence of functions that are being invoked could reveal some kind of health condition in the patient.
- At which time/location a function is invoked. For example, an online retail store could reveal the vicinity in which a product is popular, which is interesting for the store's competitors.
- The rate of function invocation. In our previous example i.e. surveillance system, this could reveal sensitive data about the traffic at the gates.

Comprehensive anonymity and disguise methods must be introduced for serverless services to prevent the breach of users' privacy.

6 CONCLUSION

In this paper, we surveyed some of the new opportunities that the vast adoption of serverless computing model will present. Then, we surveyed and categorized various serverless application domains. For each domain, we summarized the objectives for migrating to serverless services and assessed the aptness of the paradigm. We listed challenges that those applications faced and discussed existing solutions for them. We presented the areas that need further research investigations and identified open problems.

REFERENCES

[1] [n.d.]. Alibaba Serverless Workflow. https://www.alibabacloud.com/product/serverless-workflow. Accessed: 2021-3-24.
[2] [n.d.]. Amazon ElastiCache. https://aws.amazon.com/elasticache/. Accessed: 2020-3-24.
[3] [n.d.]. Amazon Kinesis Data Streams. https://aws.amazon.com/kinesis/data-streams/. Accessed: 2021-4-7.
[4] [n.d.]. Amazon Serverless Repo. https://aws.amazon.com/serverless/serverlessrepo/. Accessed: 2021-3-24.
[5] [n.d.]. Amazon State Language. https://docs.aws.amazon.com/step-functions/latest/dg/concepts-amazon-states-language.html. Accessed: 2020-3-24.
[6] [n.d.]. Apple App store. https://www.apple.com/ios/app-store/. Accessed: 2019-11-7.
[7] [n.d.]. AWS IoT Greengrass. https://aws.amazon.com/greengrass/. Accessed: 2020-3-24.
[8] [n.d.]. AWS Step Functions. https://aws.amazon.com/step-functions. Accessed: 2021-3-24.
[9] [n.d.]. Azure Durable Functions. https://docs.microsoft.com/en-us/azure/azure-functions/durable/. Accessed: 2021-3-24.
[10] [n.d.]. Facebook. https://facebook.com. Accessed: 2021-3-24.
[11] [n.d.]. Google Cloud Composer. https://cloud.google.com/composer. Accessed: 2021-3-24.

[12] [n.d.]. Google Play store. https://play.google.com. Accessed: 2019-11-7.
[13] [n.d.]. Heroku. https://heroku.com/. Accessed: 2021-4-7.
[14] [n.d.]. Lambda Authorizer. https://docs.aws.amazon.com/apigateway. Accessed: 2021-4-7.
[15] [n.d.]. Openwhisk. https://openwhisk.apache.org/. Accessed: 2021-4-7.
[16] [n.d.]. Serverless Networking SDK. http://networkingclients.serverlesstech.net/. Accessed: 2021-3-24.
[17] Gojko Adzic and Robert Chatley. 2017. Serverless computing: economic and architectural impact. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*. ACM, 884–889.
[18] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX symposium on networked systems design and implementation (NSDI 20)*. 419–434.
[19] Nabeel Akhtar, Ali Raza, Vatche Ishakian, and Ibrahim Matta. 2020. COSE: Configuring Serverless Functions using Statistical Learning. In *IEEE Conference on Computer Communications (INFOCOM 2020)*. IEEE, 129–138.
[20] Eyhab Al-Masri, Ibrahim Diabate, Richa Jain, Ming Hoi Lam Lam, and Swetha Reddy Nathala. 2018. A Serverless IoT architecture for smart waste management systems. In *IEEE International Conference on Industrial Internet (ICII)*. IEEE, 179–180.
[21] May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. 2013. Cloud computing pricing models: a survey. *International Journal of Grid and Distributed Computing* 6, 5 (2013), 93–106.
[22] Kalev Alpernas, Cormac Flanagan, Sadjad Fouladi, Leonid Ryzhyk, Mooly Sagiv, Thomas Schmitz, and Keith Winstein. 2018. Secure serverless computing using dynamic information flow control. *arXiv preprint arXiv:1802.08984* (2018).
[23] Sundar Anand, Annie Johnson, Priyanka Mathikshara, and R Karthik. 2019. Low Power Real Time GPS Tracking Enabled with RTOS and Serverless Architecture. In *4th IEEE International Conference on Computer and Communication Systems (ICCCS)*. IEEE, 618–623.
[24] Sundar Anand, Annie Johnson, Priyanka Mathikshara, and R Karthik. 2019. Real-time GPS tracking using serverless architecture and ARM processor. In *11th International Conference on Communication Systems & Networks (COMSNETS)*. IEEE, 541–543.
[25] Lixiang Ao, Liz Izhikevich, Geoffrey M Voelker, and George Porter. 2018. Sprocket: A serverless video processing framework. In *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 263–274.
[26] Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. 2016. CloudCache: On-demand flash cache management for cloud computing. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*. 355–369.
[27] Gabriel Aumala, Edwin Boza, Luis Ortiz-Avilés, Gustavo Totoy, and Cristina Abad. 2019. Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 282–291.
[28] Arda Aytekin and Mikael Johansson. 2019. Harnessing the Power of Serverless Runtimes for Large-Scale Optimization. *arXiv preprint arXiv:1901.03161* (2019).
[29] Ting Bai, Jian-Yun Nie, Wayne Xin Zhao, Yutao Zhu, Pan Du, and Ji-Rong Wen. 2018. An attribute-aware neural attentive model for next basket recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 1201–1204.
[30] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. 2017. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*. Springer, 1–20.
[31] Priscilla Benedetti, Mauro Femminella, Gianluca Reali, and Kris Steenhaut. 2021. Experimental Analysis of the Application of Serverless Computing to IoT Platforms. *Sensors* 21, 3 (2021), 928.
[32] David Bermbach, Setareh Maghsudi, Jonathan Hasenburg, and Tobias Pfandzelter. 2020. Towards auction-based function placement in serverless fog platforms. In *IEEE International Conference on Fog Computing (ICFC)*. IEEE, 25–31.
[33] Ankit Bhardwaj, Chinmay Kulkarni, and Ryan Stutsman. 2020. Adaptive Placement for In-memory Storage Functions. In *USENIX Annual Technical Conference (ATC 20)*. 127–141.
[34] Anirban Bhattacharjee, Yogesh Barve, Shweta Khare, Shunxing Bao, Aniruddha Gokhale, and Thomas Damiano. 2019. Stratum: A serverless framework for the lifecycle management of machine learning-based data analytics tasks. In *USENIX Conference on Operational Machine Learning (OpML 19)*. 59–61.
[35] Nilton Bila, Paolo Dettori, Ali Kanso, Yuji Watanabe, and Alaa Youssef. 2017. Leveraging the serverless architecture for securing linux containers. In *37th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 401–404.
[36] Yoni Birman, Shaked Hindi, Gilad Katz, and Asaf Shabtai. 2020. Cost-Effective Malware Detection as a Service Over Serverless Cloud Using Deep Reinforcement Learning. In *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 420–429.

[37] Stefan Brenner and Rüdiger Kapitza. 2019. Trust more, serverless. In *12th ACM International Conference on Systems and Storage*. ACM, 33–43.
[38] Sebastian Burckhardt, Chris Gillum, David Justo, Konstantinos Kallas, Connor McMahon, and Christopher S Meiklejohn. 2021. Serverless Workflows with Durable Functions and Netherite. *arXiv preprint arXiv:2103.00033* (2021).
[39] Rajkumar et al. Buyya. 2018. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–38.
[40] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2018. A Case for Serverless Machine Learning. In *Workshop on Systems for ML and Open Source Software at NeurIPS*, Vol. 2018.
[41] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. The server is dead, long live the server: Rise of Serverless Computing, Overview of Current State and Future Trends in Research and Industry. *arXiv preprint arXiv:1906.02888* (2019).
[42] Ryan et al. Chard. 2019. Serverless Supercomputing: High Performance Function as a Service for Science. *arXiv preprint arXiv:1908.04907* (2019).
[43] Bin Cheng, Jonathan Fuerst, Gurkan Solmaz, and Takuya Sanada. 2019. Fog Function: Serverless Fog Computing for Data Intensive IoT Services. In *IEEE International Conference on Services Computing (SCC)*. IEEE, 28–35.
[44] Angelos Christidis, Sotiris Moschoyiannis, Ching-Hsien Hsu, and Roy Davies. 2020. Enabling serverless deployment of large-scale ai workloads. *IEEE Access* 8 (2020), 70150–70161.
[45] Claudio Cicconetti, Marco Conti, and Andrea Passarella. 2020. A Decentralized Framework for Serverless Edge Computing in the Internet of Things. *IEEE Transactions on Network and Service Management* (2020).
[46] Kyle Coleman, Flavio Esposito, and Rachel Charney. 2017. Speeding up children reunification in disaster scenarios via serverless computing. In *2nd International Workshop on Serverless Computing*. ACM, 5–5.
[47] Robert Cordingly, Wen Shu, and Wes J Lloyd. 2020. Predicting Performance and Cost of Serverless Computing Functions with SAAF. In *IEEE Intl. Conf. on Dependable, Autonomic and Secure Computing*. IEEE, 640–649.
[48] Anirban Das, Andrew Leaf, Carlos A Varela, and Stacy Patterson. 2020. Skedulix: Hybrid cloud scheduling for cost-efficient execution of serverless applications. In *13th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 609–618.
[49] Pubali Datta, Prabuddha Kumar, Tristan Morris, Michael Grace, Amir Rahmati, and Adam Bates. 2020. Valve: Securing Function Workflows on Serverless Computing Platforms. In *Proceedings of The Web Conference 2020*. 939–950.
[50] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. 2020. Xanadu: Mitigating cascading cold starts in serverless function chain deployments. In *21st International Middleware Conference*. 356–370.
[51] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 2020. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. In *25th International Conference on Architectural Support for Programming Languages and Operating Systems*. 467–481.
[52] Vojislav Dukic, Rodrigo Bruno, Ankit Singla, and Gustavo Alonso. 2020. Photons: Lambdas on a diet. In *11th ACM Symposium on Cloud Computing*. 45–59.
[53] Simon Eismann, Long Bui, Johannes Grohmann, Cristina L Abad, Nikolas Herbst, and Samuel Kounev. 2020. Sizeless: Predicting the optimal size of serverless functions. *arXiv preprint arXiv:2010.15162* (2020).
[54] Simon Eismann, Johannes Grohmann, Erwin Van Eyk, Nikolas Herbst, and Samuel Kounev. 2020. Predicting the Costs of Serverless Workflows. In *ACM/SPEC International Conference on Performance Engineering*. 265–276.
[55] Unai Elordi, Luis Unzueta, Jon Goenetxea, Estíbaliz Loyo, Ignacio Arganda-Carreras, and Oihana Otaegui. 2021. On-demand Serverless Video Surveillance with Optimal Deployment of Deep Neural Networks. (2021).
[56] Yu Gan et al. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.
[57] Lang Feng, Prabhakar Kudva, Dilma Da Silva, and Jiang Hu. 2018. Exploring serverless computing for neural network training. In *IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 334–341.
[58] Nathal L Fonseka and Ankit Pansari. 2019. System and method for automatically securing sensitive data in public cloud using a serverless architecture. US Patent 10,242,221.
[59] Sadjad Fouladi, Francisco Romero, Dan Iter, Qian Li, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, and Keith Winstein. 2019. From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers. In *USENIX Annual Technical Conference*. 475–488.
[60] Justin Franz, Tanmayi Nagasuri, Andrew Wartman, Agnese V Ventrella, and Flavio Esposito. 2018. Reunifying Families after a Disaster via Serverless Computing and Raspberry Pis. In *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 131–132.
[61] Maurizio Gabbrielli, Saverio Giallorenzo, Ivan Lanese, Fabrizio Montesi, Marco Peressotti, and Stefano Pio Zingaro. 2019. No more, no less-A formal model for serverless computing. *arXiv preprint arXiv:1903.07962* (2019).

[62] Gareth George, Fatih Bakir, Rich Wolski, and Chandra Krintz. 2020. NanoLambda: Implementing Functions as a Service at All Resource Scales for the Internet of Things.. In *IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 220–231.

[63] Alireza Goli, Omid Hajihassani, Hamzeh Khazaei, Omid Ardakanian, Moe Rashidi, and Tyler Dauphinee. 2020. Migrating from monolithic to serverless: A Fintech case study. In *Companion of the ACM/SPEC International Conference on Performance Engineering*. 20–25.

[64] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Nachiappan Chidambaram, Mahmut T Kandemir, and Chita R Das. 2020. Fifer: Tackling underutilization in the serverless era. *arXiv preprint arXiv:2008.12819* (2020).

[65] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Mahmut Taylan Kandemir, Bhuvan Urgaonkar, George Kesidis, and Chita Das. 2019. Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud. In *12th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 199–208.

[66] 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).

[67] Scott et al. Hendrickson. 2016. Serverless computation with openlambda. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*.

[68] 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.

[69] Sanghyun Hong, Abhinav Srivastava, William Shambrook, and Tudor Dumitraș. 2018. Go serverless: Securing cloud via serverless design patterns. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*.

[70] MohammadReza HoseinyFarahabady, Young Choon Lee, Albert Y Zomaya, and Zahir Tari. 2017. A qos-aware resource allocation controller for function as a service (faas) platform. In *International Conference on Service-Oriented Computing*. Springer, 241–255.

[71] M Reza HoseinyFarahabady, Albert Y Zomaya, and Zahir Tari. 2017. A model predictive controller for managing QoS enforcements and microarchitecture-level interferences in a lambda platform. *IEEE Transactions on Parallel and Distributed Systems* 29, 7 (2017), 1442–1455.

[72] Justin Hu, Ariana Bruno, Brian Ritchken, Brendon Jackson, Mateo Espinosa, Aditya Shah, and Christina Delimitrou. 2020. HiveMind: A scalable and serverless coordination control platform for UAV swarms. *arXiv preprint arXiv:2002.01419* (2020).

[73] Gang Huang, Chao Wu, Yifan Hu, and Chuangxin Guo. 2021. Serverless Distributed Learning for Smart Grid Analytics. *Chinese Physics B* (2021).

[74] Ling-Hong Hung, Dimitar Kumanov, Xingzhi Niu, Wes Lloyd, and Ka Yee Yeung. 2019. Rapid RNA sequencing data analysis using serverless computing. *bioRxiv* (2019), 576199.

[75] Erika Hunhoff, Shazal Irshad, Vijay Thurimella, Ali Tariq, and Eric Rozner. 2020. Proactive serverless function resource management. In *Sixth International Workshop on Serverless Computing*. 61–66.

[76] Razin Farhan Hussain, Mohsen Amini Salehi, and Omid Semiari. 2019. Serverless edge computing for green oil and gas industry. *arXiv preprint arXiv:1905.04460* (2019).

[77] Lam Phuoc Huy, Saifullah Saifullah, Marcel Sahillioglu, and Christian Baun. 2021. Crypto Currencies Prices Tracking Microservices Using Apache OpenWhisk. (2021).

[78] Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2018. Serving deep learning models in a serverless platform. In *IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 257–262.

[79] Deepak Sirone Jegan, Liang Wang, Siddhant Bhagat, Thomas Ristenpart, and Michael Swift. 2020. Guarding Serverless Applications with SecLambda. *arXiv preprint arXiv:2011.05322* (2020).

[80] Aji John, Kristiina Ausmees, Kathleen Muenzen, Catherine Kuhn, and Amanda Tan. 2019. SWEEP: accelerating scientific research through scalable serverless workflows. In *12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*. 43–50.

[81] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the cloud: Distributed computing for the 99%. In *Symposium on Cloud Computing*. 445–451.

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

[83] Michael Jones, Paul Tarjan, Yaron Goland, Nat Sakimura, John Bradley, John Panzer, and Dirk Balfanz. 2012. Json web token (jwt). (2012).

[84] 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.

[85] Paul et al Kocher. 2018. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203* (2018).
[86] Jens Kohler. 2021. A Serverless FaaS-Architecture: Experiences from an Implementation in the Core Banking Domain. (2021).
[87] Konstantinos et al. Konstantoudakis. 2021. Serverless Streaming for Emerging Media: Towards 5G Network-Driven Cost Optimization. *arXiv preprint arXiv:2102.04910* (2021).
[88] Dimitar Kumanov, Ling-Hong Hung, Wes Lloyd, and Ka Yee Yeung. 2018. Serverless computing provides on-demand high performance computing for biomedical research. *arXiv preprint arXiv:1807.11659* (2018).
[89] Malte S Kurz. 2021. Distributed Double Machine Learning with a Serverless Architecture. *arXiv preprint arXiv:2101.04025* (2021).
[90] Benjamin D Lee, Michael A Timony, and Pablo Ruiz. 2019. A serverless web tool for DNA sequence visualization. *Nucleic acids research* (2019).
[91] Valentina Lenarduzzi and Annibale Panichella. 2020. Serverless Testing: Tool Vendors' and Experts' Points of View. *IEEE Software* 38, 1 (2020), 54–60.
[92] Changyuan Lin and Hamzeh Khazaei. 2020. Modeling and Optimization of Performance and Cost of Serverless Applications. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (2020), 615–632.
[93] Xiayue Charles Lin, Joseph E Gonzalez, and Joseph M Hellerstein. 2020. Serverless Boom or Bust? An Analysis of Economic Incentives. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.
[94] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *arXiv preprint arXiv:1801.01207* (2018).
[95] Pedro García López, Aitor Arjona, Josep Sampé, Aleksander Slominski, and Lionel Villard. 2020. Triggerflow: trigger-based orchestration of serverless workflows. In *14th ACM International Conference on Distributed and Event-based Systems*. 3–14.
[96] Nima Mahmoudi and Hamzeh Khazaei. 2021. SimFaaS: A Performance Simulator for Serverless Computing Platforms. *arXiv preprint arXiv:2102.08904* (2021).
[97] Pascal Maissen, Pascal Felber, Peter Kropf, and Valerio Schiavoni. 2020. FaaSdom: A benchmark suite for serverless computing. In *14th ACM International Conference on Distributed and Event-based Systems*. 73–84.
[98] Maciej Malawski. 2016. Towards Serverless Execution of Scientific Workflows-HyperFlow Case Study.. In *Works@ Sc*. 25–33.
[99] Johannes Manner, Stefan Kolb, and Guido Wirtz. 2019. Troubleshooting Serverless functions: a combined monitoring and debugging approach. *SICS Software-Intensive Cyber-Physical Systems* 34, 2-3 (2019), 99–104.
[100] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.
[101] Muhammed Oguzhan Mete and Tahsin Yomralioglu. 2021. Implementation of serverless cloud GIS platform for land valuation. *International Journal of Digital Earth* (2021), 1–15.
[102] Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. In *ACM SIGMOD International Conference on Management of Data*. 115–130.
[103] Jack Naglieri. 2017. StreamAlert: A Serverless, Real-time Intrusion Detection Engine. (2017).
[104] Stefan et al. Nastic. 2017. A serverless real-time data analytics platform for edge computing. *IEEE Internet Computing* 21, 4 (2017), 64–71.
[105] Truong X Nghiem, Madhur Behl, Rahul Mangharam, and George J Pappas. 2011. Green scheduling of control systems for peak demand reduction. In *50th IEEE Conference on Decision and Control and European Control Conference*. IEEE, 5131–5136.
[106] Xingzhi Niu, Dimitar Kumanov, Ling-Hong Hung, Wes Lloyd, and Ka Yee Yeung. 2019. Leveraging serverless computing to improve performance for sequence comparison. In *10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*. 683–687.
[107] Alfonso Pérez, Germán Moltó, Miguel Caballer, and Amanda Calatrava. 2019. A programming model and middleware for high throughput serverless computing applications. In *34th ACM/SIGAPP Symposium on Applied Computing*. ACM, 106–113.
[108] Matthew Perron, Raul Castro Fernandez, David DeWitt, and Samuel Madden. 2020. Starling: A scalable query engine on cloud functions. In *ACM SIGMOD International Conference on Management of Data*. 131–141.
[109] Per Persson and Ola Angelsmark. 2017. Kappa: serverless IoT deployment. In *2nd International Workshop on Serverless Computing*. ACM, 16–21.
[110] Mike Prechtl, Robin Lichtenthäler, and Guido Wirtz. 2020. Investigating possibilities for protecting and hardening installable FaaS platforms. In *Symposium and Summer School on Service-Oriented Computing*. Springer, 107–126.
[111] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. 2019. Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 193–206.

[112] Thomas Rausch, Alexander Rashed, and Schahram Dustdar. 2021. Optimized container scheduling for data-intensive serverless edge computing. *Future Generation Computer Systems* 114 (2021), 259–271.
[113] Gilbert Regan. 2019. A Serverless Architecture for Wireless Body Area Network Applications. In *6th International Symposium on Model-Based Safety and Assessment (IMBSA 19)*. Springer Nature, 239.
[114] Sebastián Risco and Germán Moltó. 2021. GPU-Enabled Serverless Workflows for Efficient Multimedia Processing. *Applied Sciences* 11, 4 (2021), 1438.
[115] Sasko Ristov, Stefan Pedratscher, and Thomas Fahringer. 2021. AFCL: An Abstract Function Choreography Language for serverless workflow specification. *Future Generation Computer Systems* 114 (2021), 368–382.
[116] Francisco Romero, Mark Zhao, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. Llama: A Heterogeneous & Serverless Framework for Auto-Tuning Video Analytics Pipelines. *arXiv preprint arXiv:2102.01887* (2021).
[117] Peter Saint-Andre. 2018. Serverless Messaging. (2018).
[118] Arnav Sankaran, Pubali Datta, and Adam Bates. 2020. Workflow Integration Alleviates Identity and Access Management in Serverless Computing. In *Annual Computer Security Applications Conference*. 496–509.
[119] Vaishaal et al. Shankar. 2018. numpywren: serverless linear algebra. *arXiv preprint arXiv:1810.09679* (2018).
[120] Rajath Shashidhara. [n.d.]. Lambda-KV: Distributed Key-Value store with co-located Serverless compute. ([n. d.]).
[121] Simon Shillaker and Peter Pietzuch. 2020. Faasm: lightweight isolation for efficient stateful serverless computing. In *USENIX Annual Technical Conference (ATC 20)*. 419–433.
[122] Arjun Singhvi, Kevin Houck, Arjun Balasubramanian, Mohammed Danish Shaikh, Shivaram Venkataraman, and Aditya Akella. 2019. Archipelago: A Scalable Low-Latency Serverless Platform. *arXiv preprint arXiv:1911.09849* (2019).
[123] Nikhila Somu, Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. 2020. Panopticon: A comprehensive benchmarking tool for serverless applications. In *nternational Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 144–151.
[124] Josef Spillner. 2019. Quantitative analysis of cloud function evolution in the AWS serverless application repository. *arXiv preprint arXiv:1905.04800* (2019).
[125] Josef Spillner, Cristian Mateos, and David A Monge. 2017. Faaster, better, cheaper: The prospect of serverless scientific computing and hpc. In *Latin American High Performance Computing Conference*. Springer, 154–168.
[126] Vikram Sreekanti, Chenggang Wu, Xiayue Charles Lin, Johann Schleier-Smith, Jose M Faleiro, Joseph E Gonzalez, Joseph M Hellerstein, and Alexey Tumanov. 2020. Cloudburst: Stateful functions-as-a-service. *arXiv preprint arXiv:2001.04592* (2020).
[127] Amoghvarsha Suresh and Anshul Gandhi. 2019. Fnsched: An efficient scheduler for serverless functions. In *5th International Workshop on Serverless Computing*. 19–24.
[128] Amoghavarsha Suresh, Gagan Somashekar, Anandh Varadarajan, Veerendra Ramesh Kakarla, Hima Upadhyay, and Anshul Gandhi. 2020. ENSURE: Efficient Scheduling and Autonomous Resource Management in Serverless Environments. In *IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 1–10.
[129] K Swedha and Tanuja Dubey. 2018. Analysis of web authentication methods using Amazon web services. In *9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 1–6.
[130] Bo Tan, Haikun Liu, Jia Rao, Xiaofei Liao, Hai Jin, and Yu Zhang. 2020. Towards Lightweight Serverless Computing via Unikernel as a Function. In *IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.
[131] Yang Tang and Junfeng Yang. 2020. Lambdata: Optimizing Serverless Computing by Making Data Intents Explicit. In *IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 294–303.
[132] Ali Tariq, Austin Pahl, Sharat Nimmagadda, Eric Rozner, and Siddharth Lanka. 2020. Sequoia: Enabling quality-of-service in serverless computing. In *11th ACM Symposium on Cloud Computing*. 311–327.
[133] Shelby Thomas, Lixiang Ao, Geoffrey M Voelker, and George Porter. 2020. Particle: ephemeral endpoints for serverless networking. In *11th ACM Symposium on Cloud Computing*. 16–29.
[134] Sergio Trilles, Alberto González-Pérez, and Joaquín Huerta. 2020. An IoT platform based on microservices and serverless paradigms for smart farming purposes. *Sensors* 20, 8 (2020), 2418.
[135] Zhucheng Tu, Mengping Li, and Jimmy Lin. 2018. Pay-per-request deployment of neural network models using serverless architectures. In *Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. 6–10.
[136] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. 2011. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review* 41, 1 (2011), 45–52.
[137] Ao Wang, Jingyuan Zhang, Xiaolong Ma, Ali Anwar, Lukas Rupprecht, Dimitrios Skourtis, Vasily Tarasov, Feng Yan, and Yue Cheng. 2020. Infinicache: Exploiting ephemeral serverless functions to build a cost-effective memory cache. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*. 267–281.

[138] Hao Wang, Di Niu, and Baochun Li. 2019. Distributed machine learning with a serverless architecture. In *IEEE Conference on Computer Communications (INFOCOM 19)*. IEEE, 1288–1296.
[139] I Wang, E Liri, and KK Ramakrishnan. 2020. Supporting IoT Applications with Serverless Edge Clouds. In *9th IEEE International Conference on Cloud Networking (CloudNet)*. IEEE, 1–4.
[140] Zekun Wang, Pengwei Wang, Peter C Louis, Lee E Wheless, and Yuankai Huo. 2021. WearMask: Fast In-browser Face Mask Detection with Serverless Edge Computing for COVID-19. *arXiv preprint arXiv:2101.00784* (2021).
[141] Mike Wawrzoniak, Ingo Müller, Rodrigo Fraga Barcelos Paulus Bruno, and Gustavo Alonso. 2021. Boxer: Data Analytics on Network-enabled Serverless Platforms. In *11th Annual Conference on Innovative Data Systems Research (CIDR'21)*.
[142] Jinfeng Wen and Yi Liu. 2021. An Empirical Study on Serverless Workflow Service. *arXiv preprint arXiv:2101.03513* (2021).
[143] Sebastian Werner, Jörn Kuhlenkamp, Markus Klems, Johannes Müller, and Stefan Tai. 2018. Serverless Big Data Processing using Matrix Multiplication as Example. In *IEEE International Conference on Big Data (Big Data)*. IEEE, 358–365.
[144] Stefan Winzinger and Guido Wirtz. 2019. Model-based analysis of serverless applications. In *11th International Workshop on Modelling in Software Engineerings*. IEEE Press, 82–88.
[145] Stefan Winzinger and Guido Wirtz. 2020. Applicability of Coverage Criteria for Serverless Applications. In *IEEE International Conference on Service Oriented Systems Engineering (SOSE)*. IEEE, 49–56.
[146] Chenggang Wu, Vikram Sreekanti, and Joseph M Hellerstein. 2020. Autoscaling tiered cloud storage in anna. *The VLDB Journal* (2020), 1–19.
[147] Di Wu, Binxing Fang, Jie Yin, Fangjiao Zhang, and Xiang Cui. 2018. Slbot: A serverless botnet based on service flux. In *IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 181–188.
[148] Mingyu Wu, Zeyu Mi, and Yubin Xia. 2020. A Survey on Serverless Computing and Its Implications for JointCloud Computing. In *IEEE International Conference on Joint Cloud Computing*. IEEE, 94–101.
[149] Yuncheng Wu, Tien Tuan Anh Dinh, Guoyu Hu, Meihui Zhang, Yeow Meng Chee, and Beng Chin Ooi. 2021. Serverless Model Serving for Data Science. *arXiv preprint arXiv:2103.02958* (2021).
[150] Fei Xu, Yiling Qin, Li Chen, Zhi Zhou, and Fangming Liu. 2021. $lambdaDNN$: Achieving Predictable Distributed DNN Training with Serverless Architectures. *IEEE Trans. Comput.* (2021).
[151] Hong Xu and Baochun Li. [n.d.]. Reducing electricity demand charge for data centers with partial execution. In *5*.
[152] Hong Xu and Baochun Li. 2013. Dynamic cloud pricing for revenue maximization. *IEEE Transactions on Cloud Computing* 1, 2 (2013), 158–171.
[153] Mengting Yan, Paul Castro, Perry Cheng, and Vatche Ishakian. 2016. Building a chatbot with serverless computing. In *Proceedings of the 1st International Workshop on Mashups of Things and APIs*. ACM, 5.
[154] Jie You, Jingfeng Wu, Xin Jin, and Mosharaf Chowdhury. 2021. Ship Compute or Ship Data? Why Not Both? (2021).
[155] Tianyi et al. Yu. 2020. Characterizing serverless platforms with serverlessbench. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 30–44.
[156] Miao Zhang, Yifei Zhu, Cong Zhang, and Jiangchuan Liu. 2019. Video processing with serverless computing: a measurement study. In *29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 61–66.
[157] Song Zhang, Xiaochuan Luo, and Eugene Litvinov. 2021. Serverless computing for cloud-based power grid emergency generation dispatch. *International Journal of Electrical Power & Energy Systems* 124 (2021), 106366.
[158] Tian Zhang, Dong Xie, Feifei Li, and Ryan Stutsman. 2019. Narrowing the gap between serverless and its state with storage functions. In *ACM Symposium on Cloud Computing*. 1–12.
[159] Wen Zhang, Vivian Fang, Aurojit Panda, and Scott Shenker. 2020. Kappa: a programming framework for serverless computing. In *11th ACM Symposium on Cloud Computing*. 328–343.