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

H. Shafiei, *Member, IEEE,* , A. Khonsari, and P. Mousavi

**Abstract**—The topic of serverless computing has proved to be a controversial subject both within academic and industrial communities. Many have praised the approach to be a platform for a new era of computing and some have argued that it is in fact a step backward. Though, both sides agree that there exist challenges that must be addressed in order to better utilize its potentials. This paper surveys existing challenges toward vast adoption of serverless services and also explores some of the challenges that have not been thoroughly discussed in the previous studies. Each challenge is discussed thoroughly and a number of possible directions for future studies is proposed. Moreover, the paper reviews some of the unique opportunities and potentials that the serverless computing presents.

**Index Terms**—Cloud Services, Serverless Computing, Function-as-a-Service (FaaS)

## 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 but 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 more affordable computation services.

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 transparent to 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.

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 [5]. In contrast, others believe that serverless computing will become the face of cloud computing and the performance issues will be addressed eventually [6]. This paper falls into the latter category.

In this paper, we review the opportunities presented by serverless computing. We emphasize that serverless services are indeed more affordable approaches for many networked services, more customer friendly as they relieve customers from the intricacies of deployment, and also they are more environmentally friendly for the reasons we will discuss later in this paper. Moreover, we argue that new market places will form around these services which implies new business opportunities for computer engineers. Various serverless applications, which have been proposed in the course of the past few years, are surveyed in this paper.

The are a number of challenges that serverless services are currently facing. These challenges are thoroughly surveyed in this paper. One of the main concerns in these services, that has serious drawbacks on the overall performance of serverless applications, is the cost of preserving the state. In fact, the serverless applications are often compared with serverful applications (running on a single VM), where objects and methods (functions) of the applications are executed on a single machine and have access to a shared memory and cache [5], [7]. It is natural to see that the latter has significantly better performance than the former since the state of the system is preserved in the memory and various parts of an application can access that. This shared memory space on a single machine is not always available in serverless applications due to its auto-scaling nature that does not guarantee the execution of related functions on a single machine. Several related work to address this issue have been surveyed in this paper and future research directions are also proposed.

Another important challenge is the security and privacy of serverless services. Various categories of attacks and privacy breaches and their possible countermeasures are discussed in this paper. Furthermore, the issues of scheduling, pricing, caching, provider management


- *H. Shafiei is with the School of Computer Science, IPM, Tehran, Iran. E-mail: shafiei@ipm.ir*
- *A. Khonsari is with the Department of Electrical and Computer Engineering, University of Tehran, Tehran, Iran. He is also with the School of Computer Science, IPM, Tehran, Iran. E-mail: ak@ipm.ir*
- *P. Mousavi is graduated from Department of Electrical and Computer Engineering, University of Tehran, Iran E-mail: pa.mousavi@ut.ac.ir*

and function invocation in serverless computing are all surveyed in this paper, and future directions are also proposed.

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 and also presents its application. Section 4 surveys the challenges toward vast adoption of the concept. Section 5 concludes the paper.

# 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 common 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 is a combination of FaaS and BaaS that incorporates the following characteristics.

## 2.2 Characteristics

There are a number of properties that a service should have in order to call it a serverless service:

1) The execution environment must be transparent to the customer. The processing node, the virtual machine, the container, its operating system and etc. are all hidden to the customer.
2) The provider[1] 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 amount of resources the customer actually used 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 transparent to the provider. The provider knows their data dependencies, dependencies to external libraries, run-time environments and state during and after execution.

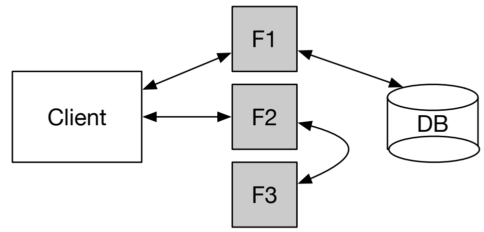


Fig. 1: An example of a serverless application.

A serverless application is usually comprised of two parts:

- A rich client. 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 per user's request or based on a predefined event.

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 saw a sequence of function executions. Later in this paper, we argue that these sequences are important for the purpose of performance optimization.

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

# 3 Opportunities

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

## 3.1 Affordable Scalability

The very first and foremost opportunity that serverless computing offers is the ability for customers to deploy the functionalities of their applications without worrying about the scalability and the execution infrastructure or

1. Throughout this paper by service provider we mean the organization (or company) who serves customers in a serverless fashion.

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 of 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 are 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 (this is mainly due to the transparency of execution environment).

### 3.2 No Deployment and Maintenance Complexity

One of the promises of the cloud computing listed in [8] 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 [9] 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.

### 3.3 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 [10] and Apple's App Store [11]. Such a scenario is predictable for serverless computing i.e., with the growth in the popularity of serverless computing, new market places for functions will arise. 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 geo-spatial function that checks whether a point resides inside a geo-spatial polygon. He/she could buy such function from those markets.

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 possibly thousands of options for a specific task.

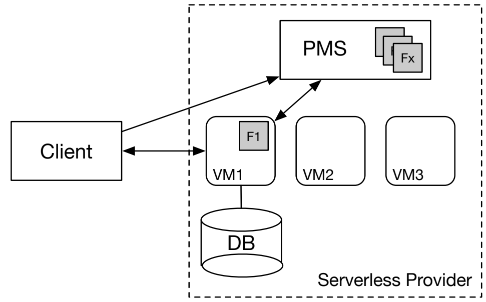


Fig. 2: An example of a serverless provider.

### 3.4 A New Programming Paradigm

In a serverless programming paradigm, functions play a key role in designing software services. In fact, serverless applications can be viewed as a composition of functions rather than a construction of code blocks. This can be called "composition over construction" paradigm. As discussed, function markets will offer programmers with various functions per customers' requirements. Thus, by providing tested, efficient and widely used functions to programmers, the complexity of software development is reduced drastically. However, this bears new challenges which programmers could face i.e., they should move from widely adopted and accepted object-oriented programming to functional programming paradigm. Although, functions can still be used in an object-oriented environment, with the so-called mutability of data in such environments, the complexity of code will increase significantly [12]. Functional programming is an old concept, however, it is gaining new traction during the past few years among software developers [13].

### 3.5 Green Computing

Serverless computing promotes green computing from two different perspectives:

#### *3.5.1 Resource Sharing*

By on-demand execution of functions and releasing the resources after execution, the energy consumption is reduced significantly. In other alternative services such as IaaS, even for a single small request the whole virtual machine with its resources must be kept alive. By sharing the resources of the infrastructure with other functions, serverless computing improves resource utilization.

#### *3.5.2 Quality of Code*

By billing per execution time, serverless computing forces the economical incentives upon programmers to improve the resource usage of their implemented code. This is not a great concern in other services. For example, billing of IaaS is per allocated resource such as the number of CPU cores. How much a CPU core is utilized does not usually affect the bill. Indeed, automated testing tools should be developed that not only verify the soundness of code but also evaluate the quality of code from the perspective of its resource usage.

# 4 APPLICATIONS

Although the concept is relatively new, many real-world serverless applications have been proposed in the literature. In what follows, we categorize and review these applications:

## 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., [14] proposed an architecture for chatbot on OpenWhisk [4]. An XMPP-based serverless approach for instant messaging is also introduced in [15].

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., [16], [17] proposed two real-time GPS tracking methods on low-power processors.

Serverless services are also utilized for data analytics applications [18]. 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.

## 4.2 Urban and Industrial Management Systems

The pay-as-you-go model of serverless services paved the way for introduction and implementation of various budget-restricted urban and industrial management systems. Al-Masri et.al., [19] presented an urban smart waste management system. A serverless monitoring and detection approach for smart grid is presented in [20]. Hussain et.al., [21] proposed a serverless service for oil and gas field management system.

Serverless services have been also utilized for urban disaster recovery applications. Franz et.al., [22] proposed a community formation method after disasters using serverless services. Another similar approach is also proposed in [23].

## 4.3 Scientific Computing

It has been debated in [5] that serverless computing is not an attractive alternative for scientific computing applications, albeit, many studies have focused their attentions toward serverless services for those application. 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 field for which the utilization of serverless services is feasible.

Spillner et.al., [24] 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 [25] where high-performance Function-as-a-Service is proposed for scientific applications. A serverless tool for linear algebra problems is proposed in [26] and a case for matrix multiplication is presented in [27]. Serverless is harnessed for large-scale optimization in [28]. A serverless case study for scientific workflows is discussed in [29].

Serverless approaches have been also used in DNA and RNA computing [30], [31]. On-demand high performance serverless infrastructures and approaches for biomedical computing is proposed in [32].

## 4.4 Machine Learning

Machine learning in general and specifically approaches based on neural networks, are currently one of the most attractive research trends. A case of serverless machine learning is discussed in [33]. Ishakian et.al., [34] discussed various deep learning models for serverless platforms. Neural network training of serverless services is explored in [35]. Also a pay-per-request deployment of neural network models using serverless services is discussed in [36].

## 4.5 Video Processing

Serverless approaches have been proposed for video processing. Sprocket [37] 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.

Zhang et al. [38] 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.

## 4.6 Security

The power of serverless computing has been leveraged for providing security for various systems and infrastructures. A mechanism for securing Linux containers has been proposed in [39]. Serverless services have also been utilized for intrusion detection. StreamAlert [40] is a serverless, real-time intrusion detection engine built upon Amazon Lambda. Serverless approaches have been also used for ensuring data security. A method for automatically securing sensitive data in public cloud using serverless architectures has been introduced in [41].

In a completely different approach, serverless approaches have been used to develop a botnet [42]. 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 (IoTs)

Serverless computing model has been exploited for various IoT domains. 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 [44]. A serverless body area network for e-health IoT applications is presented in [45].

In another research direction, Presson et al. [46] introduced Kappa which is a flexible and intuitive serverless platform for IoTs.

## 5 Challenges

This section presents challenges toward serverless computing. Instead of referring to various papers individually, we summarize the challenges presented in [5], [6], [47]–[50] as follows.

As the topic of serverless computing is relatively new, there are a number of challenges that need to be addressed. Some of these challenges are not unique to the serverless computing model and can arise in any computing environment such as security and privacy, caching or pricing. There are also some types of challenges that are specifically unique to serverless services such as function invocation or functions' intra-communication. In this section, we review both types.

Discussing the challenges toward vast adoption of serverless services is particularly important since there has been a controversy between various pundits about the benefits of migration from serverful approaches to serverless ones. The drawbacks listed in [5] and poor emulation results presented in [51] are all rooted in existing challenges that are unique to serverless services and need to be addressed in future studies. For example, placement of functions in physical infrastructure (processing nodes) is one of those challenges. In fact, with poor placement of functions on nodes without considering their data dependencies, dependency to other functions (i.e., putting the ones that share state, physically far from each other) or lack of any knowledge about the function itself, we will end up with systems that perform lower than their serverful counterparts.

### 5.1 Programming, Modeling, Testing and Debugging

As the topic of serverless computing is relatively new, its development tools are not rich enough. This poses a great challenge for software developers. Also, the lack of proper modeling paradigms leads to non-unified development approaches which will reduce quality of the code and also complicate collaborations of developers in long term. To remedy this shortcoming, Perez et al. [52] 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 [53]. Also, a model-based analysis of serverless application is discussed in [54].

Debugging and testing tools are integral parts of any software development approach. Serverless computing is not an exception. Few papers have elaborated on this; [55] proposes a combined monitoring and debugging approach for serverless functions. Integration testing of serverless functions is discussed in [56].

Another important tool to test the applicability and performance of any new idea is benchmark suites. A rich set of benchmark suites for serverless infrastructures is presented in [57].

### 5.2 Pricing

Many big technology companies now offer serverless computing services with different specifications and prices. As the popularity of serverless services increase, the number of companies and their options for pricing will grow. Many factors affect the price offered by each company. These factors range from 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 is received by a server at 2 a.m. in Winter incurs less energy bills compared to that of an exactly same request with the same resource consumption at 12 p.m. workday in Summer. Another factor is the load level that is imposed to the provider at that moment. The price offered by the competitors is also a key decision factor. Various pricing models have been proposed for cloud computing in general [58] 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 quality of service. Note that, one of the important factors in determining the quality of service is the response time.

Finding the optimal pricing strategy with multiple providers and customer constraints is a challenging issue that must be addressed in research studies. A similar notion have extensively discussed in the area of smart grids where supply and demand are considered in extracting online pricing models [59].

It is noteworthy to mention that, the nature of serverless services makes the online pricing more feasible compared to that of other cloud services. In those services such as IaaS, the cost of shipping and maintaining multiple virtual machines or containers in multiple service providers is higher compared to that of serverless settings.

## 5.3 Scheduling

In order to save energy, serverless service providers put the execution infrastructure in a sleep mode, after a period of inactivity. This period for Amazon's Lambda is 15 minutes [1]. Usually, the state of the function is saved in a persistent storage so that after the wake-up it can be restored. Upon receiving a request for a function that is in the sleep mode, the provider wakes up the infrastructure and restores the state of the system. This, naturally takes some time and leads to some latency in the service of the serverless software which in turn reduces the users' so-called quality of experience. For example, consider a simple login functionality that uses a serverless back-end. In an extreme scenario, a user has to wait up to an order of minutes to be able to login to an application after a brief period of inactivity. This may render the entire system useless. Thus, sleep scheduling is an important issue that must be addressed for serverless service providers.

Currently service providers consider constant duration for the inactivity period. It is unclear why Amazon sets 15 minutes for inactivity duration. But it is obvious that setting the same period for all functions seems naive. There are some approaches that must be considered for the scheduling:

### *5.3.1 Prediction and Forecasting*

The pattern on which the requests receive to the serverless service provider should be extracted. Using modeling and advanced techniques such as machine learning, the pattern of incoming requests can be predicted. There are various parameters that may contribute to the pattern such as the activity of other related functions, time of day or external events (e.g. a sport match).

### *5.3.2 Data Probing*

There are cases in which two functions are not directly related, however, moderate or extreme changes in data caused by one function leads to execution of other functions. For example, in a location-based recommendation application, when a user moves far than a couple of kilometers (they change their vicinity), a function that finds and suggests nearest attractions is called. Note that the function that reports the geo-location is not directly related to that of suggestion extraction function.

### *5.3.3 Warm Services*

The service providers could preserve a warm state for each function to handle requests after a period of inactivity. In this approach, some of the containers are kept ready for function invocation to handle a request whilst the main containers are being started. Then, new requests are forwarded to the main containers afterwards. One may argue that this approach leads to waste in resources available to the provider. However, the rationale behind this approach is the fact that by multiplexing the available resources on the warm containers between various inactive functions, the energy actually can be saved while improving the response time of the service. The provider should prepare warm containers beforehand and scale them in a smart fashion.

### *5.3.4 Service-level Agreement (SLA)*

The provider may agree upon preparing an always-on service for a set of functions or prepare a warm container based on the agreement with the customer.

A serverless platform designed for latency-sensitive DAG-structured serverless applications is presented in [60]. The key idea in the paper is to utilize a deadline-aware job scheduling for serverless environments. A cluster-level centralized and core-granular scheduler for serverless functions is introduced in [61].

## 5.4 Sharing and Intra-communications

Serverless software is 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 the state. In other cloud services, such data sharing infrastructure 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.

Function-level network addressing is not available in today's serverless infrastructures, thus, two functions communicate through intermediary storage systems. This burdens a huge amount of load on relatively slow and expensive storage systems compared to that of direct network connections.

Here, there are two separate challenges that must be addressed: (1) function addressing and (2) intra-communication for functions.

Functions in serverless services have some characteristics that must be considered in order to be able to introduce an 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 computing 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 changing 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 growth drastically. Thus, the proposed

addressing space should be scalable enough to be able to handle such number of functions.

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

1) Intermediate functions that serve as brokers between functions. Current serverless services use a variation of this basic idea. However, it has been argued in [6] that this burdens extreme overhead on the infrastructure. A performance evaluation of this approach that reveals the bottlenecks and further investigation and optimization of contributing factors, is a good direction for related research studies.
2) Communicate through stateless communication schemes (or protocols) such as REST or SOAP. These protocols are vastly used over the Internet and seem to be good candidates. However, the existing protocols may not be directly applicable in serverless scenarios. For example, REST is too coupled with the HTTP protocol and also its vocabulary is not rich enough for handling the situations which arise between functions. Introducing new protocols that are specifically designed for function communication needs researchers' attention.
3) Distributed message queues are also good candidates for serverless computing. This type of communication medium has already been introduced and investigated for cloud systems [62]. Available messages queues such as Apache Kafka [63] are not directly applicable for serverless environments due to their performance issues. Amazon developed Kinesis Data Streams [64] that can be used for this purpose. The topic has potential for further research.

### 5.5 Modes of Execution

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 single functions, instead, usually there are many interdependent functions, processing and passing data to each other and send back the result to the application. For example, a real-world implementation of an online social network (with functionalities similar to Facebook [65]) on Amazon's Lambda infrastructure has around 170 functions [7].

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*. Also, there are many other possibilities for the sequences of functions. As we will discuss later in this paper, knowledge of these sequences 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.

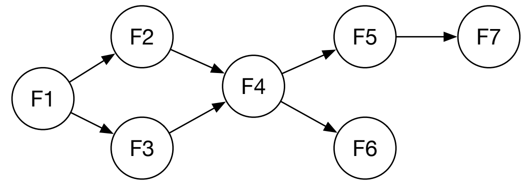


Fig. 3: An example of a dependency graph for a serverless application. This application has 7 functions where the start function is F1.

From each serverless application, a directed dependency graph can be exploited where the nodes are functions and edges show precedence of the execution of one function to another. Figure 3 shows an example of such a graph. The application depicted in the figure has 7 functions (F1 to F7). A directed edge from F1 to F2 means that each of the executions of F2 must be preceded by F1. For example, users must *sign up* to be able to *check out*.

A unique path in the graph is simply the list of unique nodes (not containing cycles) connected to each other by directed edges. In each application, there are a number of unique paths that are more common than the others, let this number be $n$. We call the set of $n$ most frequent unique paths in the dependency graph as the "$n$ modes of execution" of the regarding application. Assuming that $n$ is a system-wide variable we simply refer to it as "modes of execution" of applications.

There are two approaches to extract the modes of execution for a serverless application:

#### 5.5.1 Predefined Modes

In this approach, the application developer submits a file that contains the modes of execution, after registering their functions. The provider receives and processes the file for further actions. Extraction of modes of execution for an application is not a trivial task. Automated tools must be developed such that by giving the application as an input, the tool generates the modes of execution.

#### 5.5.2 Detection during Execution

In this approach, the dependency graph is formed inside the service provider based on the activities of the application. This approach is more user-friendly since it does not need any further dependency graph generation tools. It is also more flexible in the sense that it extracts the modes of execution based on actual behavior of the application. Recall that by definition, we are seeking $n$ most frequent paths for the application. In large applications with hundreds of functions, there are many paths and sometimes it is not trivial to predict the exact behavior of the users to extract the most frequent ones.

Exploiting them during execution leads to more accurate estimations. This approach is also more robust than the others since the predefined modes presented by the users could be faulty or suffer from less elaborate processing.

## 5.6 Function Invocation

Every serverless provider must enable two basic methods for the application developers; registration of functions and invocation of those registered functions. In a function invocation, a request is sent to the provider that contains the input data of the function (or the address to the data) and the ID of the function. The function invocation has various types:

### 5.6.1 Application-based vs Function-based

In many practical scenarios, the customer's application sends a request for function invocation and the result is sent back to it. Then, based on the result, the application may send another invocation request to the provider. There are also a number of scenarios in which the function itself invokes another function during or at the end of its execution.

### 5.6.2 Request-based vs Event-based

In a request-based invocation, an application (or a function) requests the invocation of a function, whereas, in an event-based approach a probe is defined by the function owner such that an event is triggered based on changes detected by the probe. For example, a probe can be defined over a database that if the number of purchases in a day goes beyond a threshold, a certain function from accounting service must be invoked.

### 5.6.3 Synchronous vs Asynchronous

The invocation of the function can either be synchronous to the caller or asynchronous. In a synchronous invocation the caller (which could either be an application or a function) is dependent on the called function i.e., it waits until the called function ends. In an asynchronous invocation, the caller invokes the function and proceeds to its other tasks. In the latter, the owner must provide callback mechanisms to notify the caller after the execution. For example, in an online social network application, the *login* function is synchronous since the application cannot proceed without user authentication, whereas, avatar preparation function (a function which generates multiple scales of the avatar i.e, thumbnail, etc.) can be invoked asynchronously.

Asynchronous function invocation is challenging in some of the serverless's real-world application domains. For example, there are two challenges regarding this type of function invocation in low-power IoTs. Those devices put the hardware in a sleep mode after a period of inactivity. If during one of these sleep cycles the callback of an asynchronous function reaches to the device, it will be missed. This scenario is not unique to serverless environments, however, as being short-lived is one of the defining features of functions in these environments, the function might not "live" enough to be able to resend the callback. Also, if the request is function-based, the requester function must outlive the called function. The provider must consider these cases and extend the lifespan of the functions. However, the extension of function's lifespan increases the resource consumption inside providers. The trade-off between these two must be further studied by future research studies.

Serverless providers usually enable variety of approaches to invoke registered function. This often ranges from well-known REST-based approaches to custom command-line interfaces (CLIs) [66]. Although using REST is very common for today's serverless function invocation, one can be skeptical about the feasibility of these approaches in low-power devices. Making HTTP requests does not bear heavy computation loads on the requester, however, for the requester to be able to handle asynchronous callbacks, an HTTP server must be available inside the device which consumes vast amount of the device's scarce resource and sometimes goes beyond its hardware capabilities. To overcome this issue, either the devices should use some sort of intermediary broker between them and the serverless provider, or they should use other protocols that are designed to handle these cases.

## 5.7 Packing

The main purpose of serverless's existence lies in its ability to auto-scale itself by invoking copies of customers' functions and assign each request to those copies. There are no predefined approaches to where to put the copies of functions physically. Instead, load balancing systems inside the service provider decide based on current utilization of computing nodes and their available resources. As reported in [6], this approach leads to serious performance degradation. It has also been reported that a serverless application has performed 2.5 orders of magnitude slower in this setting compared to that of an IaaS-based approach [5]. This lower performance is mainly due to the fact that the data needs to be shipped across the network (i.e., nodes and racks) in order to be received by the function. This leads to higher traffic inside the provider which in turn exacerbates 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.

There are some considerations for packing of functions that must be taken into account:

### 5.7.1 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 possible due to various reasons: (1) one or more of the data sources are already near another 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. Thus, finding an optimal point 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.

#### 5.7.2 Multiple Functions and A Single Data Source

This case is actually simpler. In this case, multiple functions are packed together with the data source. However, there are scenarios in which the number of functions and their processing requirements are not readily available for packing.

#### 5.7.3 Chunks of Data vs Bulk of 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 table of a company's database, that specific table is important and can be packed with the function instead of the whole database.

#### 5.7.4 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 is 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 all together.

#### 5.7.5 Source Location

The relative geographical location of the request to the function also may play a role. In fact, 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 prior to the execution of function, careful manifestation of data dependency is needed in order to find the optimum placement of the function. In the evolutionary mode which was described above, the packing is done during the execution. The packing can also be done after the first execution. In this case, the monitoring data is revised after the first execution and by using optimization techniques and machine learning approaches the optimal packing strategy is extracted in order to improve the performance of the service and to minimize costs.

### 5.8 Data Caching

To avoid the bottlenecks and latencies of persistent storage, software applications utilize multiple levels of caches. This is a common practice among cloud-based applications as well [67]. Utilizing caches in serverless environments is a challenging issue since functions are executed independent 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 computing node, which renders the caching totally useless. This is also true when multiple functions consecutively work on a chunk of data i.e., if the infrastructure changes, the cached data becomes expired. Without caching, the costs, overheads and latency grow dramatically which make the serverless services infeasible. Thus, this is one of the important challenges toward the successful implementation of any serverless service.

In designing and implementing caches, the following must be considered:

#### 5.8.1 Effect of packing

In packing of function and 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 computing 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 consumes the data toward the vicinity of the data, and thus with a proper local caching, the chance of cache hits is improved.

#### 5.8.2 Effect of the modes of execution

The mode 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 mode is considered in the local caching strategy.

#### 5.8.3 Local caching vs distributed caching

As discussed above, in some of the real-world scenarios of serverless computing an efficient local caching can be feasible[2]. However, there are cases in which the functions cannot be shipped to the vicinity of the data. In these cases a distributed caching can be utilized.

Distributed in-memory caches usually utilize distributed hash functions (DHT) to extract the location

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

of cached data [68]. 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.

In the case of serverless services, this may actually 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.

### 5.9 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 provider management system (PMS) 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 category and required resources. The provider then sends back credentials for invocation. Other tracking and commercial aspects are handled by the provider during this step.
- *Invocation*: A provider receives invocation requests from applications or other functions, checks user credentials and then finds a feasible computing node and assigns the function to the node for execution. To find a feasible node for functions, it classifies functions based on their resources and data requirements and also provisions the available resources of nodes in an online manner. The aforementioned tasks of packing, scheduling and caching are also the responsibility of the provider's management system which are done in collaboration with the computing nodes.
- *Execution*: Although the execution takes place inside the nodes, PMS closely monitors the execution of functions to detect errors and malfunctions. It gathers the execution log to analyze the footprints and thus improve future invocations.

There are two approaches to attain a management 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.

In order for the PMS to be able to handle its responsibilities, manage resources and optimize the services, it should have an online view of the entire system. There are various pieces of information that contribute to the formation of this view, such as:

1) Information about the functions: their data dependency, the modes of execution, 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 PMS.

### 5.10 Security and Privacy

Security is an indispensable concern in any computation service, be it serverless or not. There are various security challenges that are common between serverless and other cloud services. Those challenges are not the concern of this paper. Instead we focus on the security issues that specifically threaten the normal operation of serverless service. Moreover, we consider the privacy of users in such environments.

#### *5.10.1 Authentication and Authorization Schemes*

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 [69]. Amazon's Lambda currently implemented such a scheme that uses a bearer token authentication strategy to determine the caller's identity [70]. 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 attacks could be encountered. 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. Secure energy-efficient protocols for authentication in such devices should be introduced.

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 attack is to replay a logout request indefinitely to prevent users from accessing their desired service. To remedy these types of attacks, a challenge-response protocol must be adopted for serverless services.

There is also the issue of authorization for functions. That is, which functions can invoke which functions. This is basically different from application-level authentication that we mentioned earlier. Here we authorize a function to call another function i.e., function-level authorization. Lacking a proper authorization scheme can actually pose severe threats to the applications security. Recall that one of the advantages of serverless services, that 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. A proper mechanism should be introduced to handle function-level authorization.

#### 5.10.2 Common Execution Environments

In the wake of Meltdown [71] and Spectre [72] 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. On one hand, research studies should focus on providing isolation strategies to overcome vulnerabilities, and on the other hand, methods for detection of malicious activities of functions must be investigated.

To counter these type of attacks, a lightweight and high performance JavaScript engine is presented in [73] that utilizes Intel's SGX enclaves to secure the execution environments.

#### 5.10.3 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 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 heavy financial load to the application owner which is in the interest of its competitors. Monitoring approaches must be introduced for serverless providers to detect and mitigate these types of attacks

Resource exhaustion attack 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 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 a heavy traffic inside the network of the provider.

#### 5.10.4 Privacy Issues

There are many privacy-intensive applications for serverless services specially in the area of IoTs. 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, an attacker may be interested in answering the question whether a user has a heart condition or not.

There are many 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 revoked. 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 revocation, e.g., in a health-care monitoring system, a sequence of functions that are being revoked could reveal some kind of health condition in the patient.
- At which time/location a function is revoked. For example, in an online retail store, it could reveal the vicinity in which a product is popular, which is interesting for the store's competitors.
- The rate of function revocation. 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 various new opportunities and potentials that serverless computing offers were discussed. We surveyed some of the new opportunities that the vast adoption of this computing model will present. Then, we surveyed the challenges that prevent utilizing the great potentials of serverless services. We discussed possible solutions to remedy those challenges. We also believe that by elaborating on the challenges, in a near future, serverless computing will indeed become the face of cloud computing as predicted in [6].

## REFERENCES


[1] "Amazon lambda," https://aws.amazon.com/lambda/, accessed: 2019-10-7.

[2] "Cloud functions," https://cloud.google.com/functions/, accessed: 2019-10-7.

[3] "Azure functions," https://azure.microsoft.com/en-us/services/functions/, accessed: 2019-10-7.

[4] "Openwhisk," https://openwhisk.apache.org/, accessed: 2019-10-7.

[5] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," *arXiv preprint arXiv:1812.03651*, 2018.

[6] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar *et al.*, "Cloud programming simplified: a berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[7] G. Adzic and R. Chatley, "Serverless computing: economic and architectural impact," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 884–889.

[8] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, no. 13, p. 2009, 2009.

[9] "Heroku," https://heroku.com/, accessed: 2019-10-7.

[10] "Google play store," https://play.google.com, accessed: 2019-11-7.

[11] "Apple app store," https://www.apple.com/ios/app-store/, accessed: 2019-11-7.

[12] M. Coblenz, J. Sunshine, J. Aldrich, B. Myers, S. Weber, and F. Shull, "Exploring language support for immutability," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 736–747.

[13] I. Perez, M. Bärenz, and H. Nilsson, "Functional reactive programming, refactored," *ACM SIGPLAN Notices*, vol. 51, no. 12, pp. 33–44, 2018.

[14] M. Yan, P. Castro, P. Cheng, and V. Ishakian, "Building a chatbot with serverless computing," in *Proceedings of the 1st International Workshop on Mashups of Things and APIs*. ACM, 2016, p. 5.

[15] P. Saint-Andre, "Serverless messaging," 2018.

[16] S. Anand, A. Johnson, P. Mathikshara, and R. Karthik, "Real-time gps tracking using serverless architecture and arm processor," in *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*. IEEE, 2019, pp. 541–543.

[17] ——, "Low power real time gps tracking enabled with rtos and serverless architecture," in *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*. IEEE, 2019, pp. 618–623.

[18] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Ristov, and R. Prodan, "A serverless real-time data analytics platform for edge computing," *IEEE Internet Computing*, vol. 21, no. 4, pp. 64–71, 2017.

[19] E. Al-Masri, I. Diabate, R. Jain, M. H. L. Lam, and S. R. Nathala, "A serverless iot architecture for smart waste management systems," in *2018 IEEE International Conference on Industrial Internet (ICII)*. IEEE, 2018, pp. 179–180.

[20] X. Geng, O. Ma, Y. Pei, Z. Xu, W. Zeng, and J. Zou, "Research on early warning system of power network overloading under serverless architecture," in *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)*. IEEE, 2018, pp. 1–6.

[21] R. F. Hussain, M. A. Salehi, and O. Semiari, "Serverless edge computing for green oil and gas industry," *arXiv preprint arXiv:1905.04460*, 2019.

[22] J. Franz, T. Nagasuri, A. Wartman, A. V. Ventrella, and F. Esposito, "Reunifying families after a disaster via serverless computing and raspberry pis," in *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 2018, pp. 131–132.

[23] K. Coleman, F. Esposito, and R. Charney, "Speeding up children reunification in disaster scenarios via serverless computing," in *Proceedings of the 2nd International Workshop on Serverless Computing*. ACM, 2017, pp. 5–5.

[24] J. Spillner, C. Mateos, and D. A. Monge, "Faaster, better, cheaper: The prospect of serverless scientific computing and hpc," in *Latin American High Performance Computing Conference*. Springer, 2017, pp. 154–168.

[25] R. Chard, T. J. Skluzacek, Z. Li, Y. Babuji, A. Woodard, B. Blaiszik, S. Tuecke, I. Foster, and K. Chard, "Serverless supercomputing: High performance function as a service for science," *arXiv preprint arXiv:1908.04907*, 2019.

[26] V. Shankar, K. Krauth, Q. Pu, E. Jonas, S. Venkataraman, I. Stoica, B. Recht, and J. Ragan-Kelley, "numpywren: serverless linear algebra," *arXiv preprint arXiv:1810.09679*, 2018.

[27] S. Werner, J. Kuhlenkamp, M. Klems, J. Müller, and S. Tai, "Serverless big data processing using matrix multiplication as example," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 358–365.

[28] A. Aytekin and M. Johansson, "Harnessing the power of serverless runtimes for large-scale optimization," *arXiv preprint arXiv:1901.03161*, 2019.

[29] M. Malawski, "Towards serverless execution of scientific workflows-hyperflow case study." in *Works@ Sc*, 2016, pp. 25–33.

[30] B. D. Lee, M. A. Timony, and P. Ruiz, "A serverless web tool for dna sequence visualization," *Nucleic acids research*, 2019.

[31] L.-H. Hung, D. Kumanov, X. Niu, W. Lloyd, and K. Y. Yeung, "Rapid rna sequencing data analysis using serverless computing," *bioRxiv*, p. 576199, 2019.

[32] D. Kumanov, L.-H. Hung, W. Lloyd, and K. Y. Yeung, "Serverless computing provides on-demand high performance computing for biomedical research," *arXiv preprint arXiv:1807.11659*, 2018.

[33] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz, "A case for serverless machine learning," in *Workshop on Systems for ML and Open Source Software at NeurIPS*, vol. 2018, 2018.

[34] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 257–262.

[35] L. Feng, P. Kudva, D. Da Silva, and J. Hu, "Exploring serverless computing for neural network training," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 334–341.

[36] Z. Tu, M. Li, and J. Lin, "Pay-per-request deployment of neural network models using serverless architectures," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 2018, pp. 6–10.

[37] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter, "Sprocket: A serverless video processing framework," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2018, pp. 263–274.

[38] M. Zhang, Y. Zhu, C. Zhang, and J. Liu, "Video processing with serverless computing: a measurement study," in *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2019, pp. 61–66.

[39] N. Bila, P. Dettori, A. Kanso, Y. Watanabe, and A. Youssef, "Leveraging the serverless architecture for securing linux containers," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2017, pp. 401–404.

[40] J. Naglieri, "Streamalert: A serverless, real-time intrusion detection engine," 2017.

[41] N. L. Fonseka and A. Pansari, "System and method for automatically securing sensitive data in public cloud using a serverless architecture," Mar. 26 2019, uS Patent 10,242,221.

[42] D. Wu, B. Fang, J. Yin, F. Zhang, and X. Cui, "Slbot: A serverless botnet based on service flux," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018, pp. 181–188.

[43] B. Cheng, J. Fuerst, G. Solmaz, and T. Sanada, "Fog function: Serverless fog computing for data intensive iot services," in *2019 IEEE International Conference on Services Computing (SCC)*. IEEE, 2019, pp. 28–35.

[44] L. F. Herrera-Quintero, J. C. Vega-Alfonso, K. B. A. Banse, and E. C. Zambrano, "Smart its sensor for the transportation planning based on iot approaches using serverless and microservices architecture," *IEEE Intelligent Transportation Systems Magazine*, vol. 10, no. 2, pp. 17–27, 2018.

[45] G. Regan, "A serverless architecture for wireless body area network applications," in *Model-Based Safety and Assessment: 6th International Symposium, IMBSA 2019, Thessaloniki, Greece, October 16–18, 2019, Proceedings*. Springer Nature, 2019, p. 239.

[46] P. Persson and O. Angelsmark, "Kappa: serverless iot deployment," in *Proceedings of the 2nd International Workshop on Serverless Computing*. ACM, 2017, pp. 16–21.

[47] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, "Status of serverless computing and function-as-a-service (faas) in industry and research," *arXiv preprint arXiv:1708.08028*, 2017.

[48] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski *et al.*, "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*. Springer, 2017, pp. 1–20.

[49] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2017, pp. 405–410.

[50] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 159–169.

[51] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural implications of function-as-a-service computing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2019, pp. 1063–1075.

[52] A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, "A programming model and middleware for high throughput serverless computing applications," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM, 2019, pp. 106–113.

[53] M. Gabbrielli, S. Giallorenzo, I. Lanese, F. Montesi, M. Peressotti, and S. P. Zingaro, "No more, no less-a formal model for serverless computing," *arXiv preprint arXiv:1903.07962*, 2019.

[54] S. Winzinger and G. Wirtz, "Model-based analysis of serverless applications," in *Proceedings of the 11th International Workshop on Modelling in Software Engineerings*. IEEE Press, 2019, pp. 82–88.

[55] J. Manner, S. Kolb, and G. Wirtz, "Troubleshooting serverless functions: a combined monitoring and debugging approach," *SICS Software-Intensive Cyber-Physical Systems*, vol. 34, no. 2-3, pp. 99–104, 2019.

[56] S. Winzinger and G. Wirtz, "Coverage criteria for integration testing of serverless applications."

[57] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, Y. He, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems," in *Proceedings of the Twenty Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, April 2019.

[58] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, and I. Ahmad, "Cloud computing pricing models: a survey," *International Journal of Grid and Distributed Computing*, vol. 6, no. 5, pp. 93–106, 2013.

[59] P. Li, H. Wang, and B. Zhang, "A distributed online pricing strategy for demand response programs," *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 350–360, 2017.

[60] A. Singhvi, K. Houck, A. Balasubramanian, M. D. Shaikh, S. Venkataraman, and A. Akella, "Archipelago: A scalable low-latency serverless platform," *arXiv preprint arXiv:1911.09849*, 2019.

[61] K. Kaffes, N. J. Yadwadkar, and C. Kozyrakis, "Centralized core-granular scheduling for serverless functions," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2019, pp. 158–164.

[62] D. Patel, F. Khasib, I. Sadooghi, and I. Raicu, "Towards in-order and exactly-once delivery using hierarchical distributed message queues," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2014, pp. 883–892.

[63] N. Garg, *Apache Kafka*. Packt Publishing Ltd, 2013.

[64] "Amazon kinesis data streams," https://aws.amazon.com/kinesis/data-streams/, accessed: 2019-10-7.

[65] "Facebook," https://facebook.com, accessed: 2019-10-7.

[66] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with openlambda," in *8th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[67] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao, "Cloudcache: On-demand flash cache management for cloud computing," in *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, 2016, pp. 355–369.

[68] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 45–52, 2011.

[69] M. Jones, P. Tarjan, Y. Goland, N. Sakimura, J. Bradley, J. Panzer, and D. Balfanz, "Json web token (jwt)," 2012.

[70] "Lambda authorizer," https://docs.aws.amazon.com/apigateway, accessed: 2019-10-7.

[71] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[72] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *arXiv preprint arXiv:1801.01203*, 2018.

[73] S. Brenner and R. Kapitza, "Trust more, serverless," in *Proceedings of the 12th ACM International Conference on Systems and Storage*. ACM, 2019, pp. 33–43.