

# A Holistic View on Resource Management in Serverless Computing Environments: Taxonomy and Future Directions

ANUPAMA MAMPAGE, SHANIKA KARUNASEKERA, and RAJKUMAR BUYYA, The University of Melbourne, Australia

Serverless computing has emerged as an attractive deployment option for cloud applications in recent times. The unique features of this computing model include rapid auto-scaling, strong isolation, fine-grained billing options and access to a massive service ecosystem which autonomously handles resource management decisions. This model is increasingly being explored for deployments in geographically distributed edge and fog computing networks as well, due to these characteristics. Effective management of computing resources has always gained a lot of attention among researchers. The need to automate the entire process of resource provisioning, allocation, scheduling, monitoring and scaling, has resulted in the need for specialized focus on resource management under the serverless model. In this article, we identify the major aspects covering the broader concept of resource management in serverless environments and propose a taxonomy of elements which influence these aspects, encompassing characteristics of system design, workload attributes and stakeholder expectations. We take a holistic view on serverless environments deployed across edge, fog and cloud computing networks. We also analyse existing works discussing aspects of serverless resource management using this taxonomy. This article further identifies gaps in literature and highlights future research directions for improving capabilities of this computing model.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → **Distributed architectures**.

Additional Key Words and Phrases: Serverless Computing, Resource Management, Workload Characterization, Performance Prediction, Application Modelling, Resource Scheduling, Resource Scaling

## ACM Reference Format:

Anupama Mampage, Shanika Karunasekera, and Rajkumar Buyya. 2021. A Holistic View on Resource Management in Serverless Computing Environments: Taxonomy and Future Directions. *ACM Comput. Surv.* 1, 1 (November 2021), 38 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 INTRODUCTION

In computing, resource management refers to the process of provisioning and allocating the right amount of resources for an application, and scheduling the constituent components of the application on the selected resources to meet certain Quality of Service (QoS) goals set by the involved parties. The serverless computing model is built on the principle that the provider takes full responsibility of the entire aspect of resource management for applications, unburdening the consumers of the complexities of infrastructure management. As such, once an application is deployed on a serverless platform, the serverless provider is accountable for the whole chain of

---

Authors' address: Anupama Mampage, [mampage@student.unimelb.edu.au](mailto:mampage@student.unimelb.edu.au); Shanika Karunasekera, [karus@unimelb.edu.au](mailto:karus@unimelb.edu.au); Rajkumar Buyya, [rbuyya@unimelb.edu.au](mailto:rbuyya@unimelb.edu.au), Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2021 Association for Computing Machinery.

0360-0300/2021/11-ART \$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

activities starting with the initial resource allocation and scheduling responsibilities as well as any subsequent monitoring and resource scaling requirements of the application.

An application deployed on a serverless platform is formed of granular code segments containing the application logic, called "functions", which are generally stateless. A single application may be composed of a single function or multiple functions with dependencies on each other. The user specifies the workflow dependency graph and the abstract resource requirements [68] and the provider handles the runtime orchestration decisions for the application. A serverless platform could be deployed on distributed computing resources available on multiple infrastructure domains. This could be the public cloud, on-premise private cloud resources or the emerging edge and fog computing architectures. At the edge computing networks, the compute resources will include sensors, smart phones and smart appliances with spare compute and storage capabilities. Fog resources, which form the network between the edge and the cloud, would have compute, storage and memory capabilities at a level in between the edge and cloud layers. The resource management decisions in a serverless platform deployed commercially, need to be directed towards satisfying both the consumer and the provider. QoS goals for the consumer is associated with application performance, the incurred cost for application execution and concerns of application security. The provider aims to use the underlying resources efficiently, targeting a high throughput system. Increased resource utilization at one point sacrifices the QoS guarantees to the user and thus these are identified as conflicting objectives.

Management of resources in serverless computing environments deployed in cloud, fog and edge resources have been studied by many. The basic scheduling challenge of mapping function execution requests to available compute resources, is influenced by various other challenges that are identified within the broader concept of resource management. These include application workload modelling techniques, mechanisms for application isolation and resource scaling strategies.

A literature survey can provide a foundation to examine and understand an evolving research area, in the context of existing work. Preliminary surveys on serverless computing identify opportunities of this new computing model, inherent challenges with the serverless concept and ideas for overcoming the said drawbacks [115], [68] [48], [107], [101], [27]. Hellerstein et al. pinpoint specific challenges of the serverless model with regard to applicability for data-driven distributed computing [62]. Garcia et al. analyse and discuss trade-offs of today's serverless platforms required for effective processing of big data analytics applications [52]. Many studies focus on exploring performance of existing commercial and open-source serverless platforms [131], [78], [75], [106], [136]. Eismann et al. present a characterization of serverless use cases [45]. A comprehensive study on the characterization of applications which have successfully adapted to the serverless domain is done in [44]. None of these works are focused specifically on the overall aspect of resource management under the serverless computing model and the emerging body of related literature.

In this article, we propose a broad survey on the aspects of resource management in serverless computing environments covering the cloud, fog and edge computing environments. We first identify three major elements of resource management under this computing model, namely workload characterization and performance prediction (workload modelling), resource scheduling and scaling. We then propose a taxonomy of factors which affect these three aspects, considering the unique characteristics of this novel computing model. This classification covers effects of serverless system designs, application characteristics, and QoS goals, on the application workload modelling, scheduling and scaling decisions. In conducting this study, we refer to existing serverless platforms as well as research literature over the past few years. Specifically, we do a thorough analysis on the existing research works and incorporate those presenting sound techniques, with a high potential to drive innovation in this field of study. It is to be noted that our focus is primarily on studies referring to aspects of resource management in serverless environments. Next, we analyse key

techniques for resource management in existing literature by applying this taxonomy. This survey presents serverless system designers with a broader overview of the primary characteristics to be considered for their target infrastructures and a clear idea of the influencing factors. A summary of existing approaches is provided for researchers studying resource management techniques for serverless environments. This study also assists serverless application developers to determine ways to maximize benefit from their deployments.

The rest of the paper is organized as follows: Section 2 provides a brief background on the serverless model, its unique features, an overview on the existing serverless platforms and frameworks, a highlight on the primary use cases for this computing model and a discussion on the concept of serverless resource management which leads to the motivation for this work. Section 3 presents the proposed taxonomy of resource management. Section 4 summarizes existing works on serverless resource management based on the taxonomy. Section 5 concludes the paper by summarizing our detailed review and highlighting ideas for future directions in this area, stemming from the gaps identified in existing approaches.

## 2 BACKGROUND

The scope of resource management in any computing environment needs to be aligned with the specific challenges unique to that particular ecosystem. In this survey we take a holistic view on the different aspects of the serverless computing model, that influence designing resource management techniques that enable stakeholders to meet their respective goals and objectives.

In order to better understand the scope and breadth of the resource management problem, in this section, we present a brief background on the concept of serverless computing, the basic execution flow, and the key characteristics of a serverless environment. Further, we provide a summary on the features of existing key commercial and open-source serverless platforms, along with a short introduction to the main application domains for serverless use cases. Then we introduce the scope of resource management under this computing model, its challenges and the motivation for our work.

### 2.1 Serverless Computing

Serverless computing is an application service model in which, the provider manages the server, and handles all the responsibilities related to the execution of the application during its lifetime, with minimum involvement of the user.

Starting with bare-metal servers maintained on-premise at redundancy, in order to enable high availability, the way server infrastructure is managed for use by consumers has evolved largely over the years. The biggest transformation in this regard was with the advent of the concept of cloud computing around early 2000. Along with it came the three cloud service models, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) (Figure 1). Under the IaaS model (e.g., AWS EC2, Azure Virtual Machines (VMs), Google Cloud Platform), the provider manages hardware resources at their data centers. The developer rents out required virtualized cloud resources from a vendor and thus possesses the responsibilities of resource provisioning, runtime configurations and the management of application code and data. While this pay-as-you-go model in acquiring cloud computing resources allows customers to pay for only the resources leased from the cloud providers, studies show a significant gap between the resources acquired and paid for by cloud users and the actual resource utilization (CPU, memory etc.) [28]. Thus an inherent challenge with the IaaS model is the resultant under utilization of resources in general, when resources are acquired to match peak demand of a system, and the resultant over utilization and performance degradation when resources are acquired to cater to average demand levels. The PaaS model (e.g., AWS Elastic Beanstalk, Azure app services, Google app engine) provides a platform

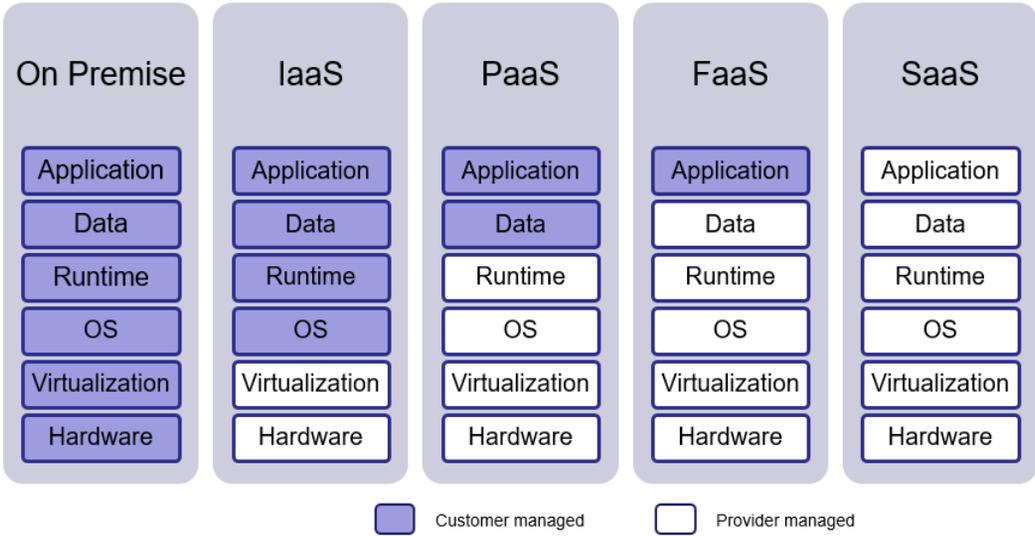


Fig. 1. Evolution of Cloud Service Models

for users to develop, run and manage customized applications while the execution environment is managed by the cloud provider. The serverless computing model is a similar paradigm where, while the developers have control over the code they deploy, they need to follow certain standards to suit the provider platforms. In addition, this model offers far more granularity with regard to application scaling and billing schemes which differentiate it from other execution models and create new opportunities. For example, under the serverless model, an application has the option to scale to zero, with no instance of the application consuming any resources when there is no traffic. This is in contrast to a PaaS model, where at least one instance of the application will always be up and running, consuming some amount of resources, irrespective of the traffic levels.

**2.1.1 Serverless Computing Architecture.** Serverless architecture is an event-driven architecture where users would initially deploy code with the application logic, in the form of stateless functions. A serverless platform defines a set of event sources which could trigger the invocation of these pre-deployed functions as per the user requirement. A user defines rules, binding deployed functions with corresponding event sources. The supported event sources depend on the platform and these could be HTTP requests from a user interface, a change to a database or an object storage, a notification from an Internet of Things (IoT) device etc. Figure 2 illustrates the basic execution flow of the serverless architecture.

Upon the occurrence of an event, a request(s) is sent to the API gateway in order to invoke the relevant function(s) as per the defined set of rules. The scheduler then determines which worker node is best suited to accommodate the function execution and dispatches the execution request to the relevant worker. A suitable isolated environment with the required resource configurations (e.g., a container) is created on the worker to accommodate the request. Function execution commences once the required runtime and the associated application code from the application repository, are loaded on to the created environment. Upon completion of execution, the response is sent to the user and the environment created is usually destroyed, releasing the allocated resources. Intermediary data and state management is done via external storage services. The function scaling decisions to

meet the requirements of subsequent function execution requests, are taken considering the data from monitoring metrics and the implemented function scaling logic.

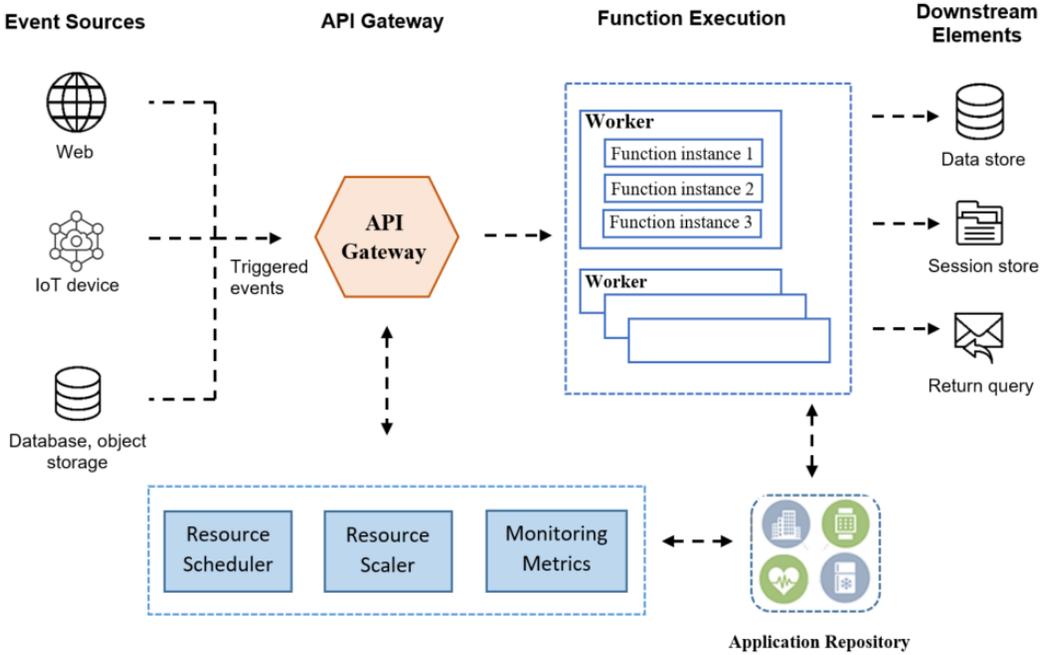


Fig. 2. Serverless Architecture

## 2.2 Key Characteristics of Serverless Platforms

As per [15], below are some unique key features of existing serverless platforms.

- **Auto-scaling** – The platform is expected to be able to scale resources automatically and instantly as per the demand. Serverless platforms are equipped with container technologies which have minimal start-up delays, thus enabling the provision of thousands of instances within a few seconds. Similarly, when there is no traffic to an application, the function instances scale to zero maintaining minimum idle resources.
- **Billing** – The usage is metered and users are only charged for the resources used when serverless functions are in execution. This means that when a function scales to zero and no node is running the user's code, there is no cost to the user.
- **Performance and limits** – A variety of limits are set on the run time resource requirements of a serverless code, including the number of concurrent requests, maximum memory and CPU resources available to a function invocation and an upper bound in execution time before a function instance times out.
- **Programming languages** – Most platforms support function code written in multiple programming languages which include Javascript, Java, Python, Go and Swift.
- **Security and accounting** – Serverless platforms are multi-tenant and thus providers need to be considerate of isolating the execution of functions of different users. Linux kernel features like namespaces and cgroups offered by container technologies provide some level of resource isolation for individual function executions.

### 2.3 Serverless Computing Platforms and Frameworks

Currently, all the major cloud service providers have launched commercial serverless platforms, namely Amazon Web Services (AWS) Lambda, Google Cloud Functions, Azure Functions and IBM Cloud Functions. While they provide additional complementary services as well for serverless application executions, they require the function code to be composed in certain ways resulting in vendor lock-in in the long term. To overcome these limitations, open source serverless frameworks have emerged over the years. These frameworks could be deployed on private cloud resources as well as on devices at the edge/fog, thus bringing the serverless computing capabilities on-premise with better flexibility. Figure 3 presents the existing commercial serverless platforms and some of the open source frameworks.

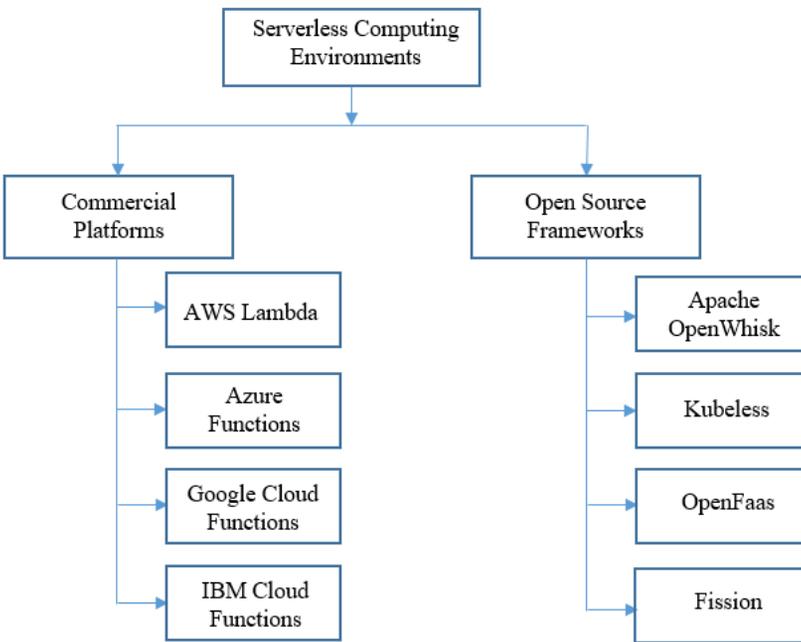


Fig. 3. Existing Serverless Platforms and Frameworks

**2.3.1 AWS Lambda.** Function as a service (FaaS) offering from AWS is done via the AWS Lambda serverless platform launched in 2014. AWS Lambda currently holds the leading position in the market for serverless computing with a wide range of services available. Lambda supports code written in Node.js, Python, Java, C# and Golang languages [78]. The platform identifies changes to data in an Amazon Simple Storage Service (Amazon S3) bucket or an Amazon DynamoDB table, HTTP requests using Amazon API Gateway and API calls made using AWS SDKs as event triggers. At the time of writing this article, AWS offers a free-tier of 1 million requests and 400,000 GB-Seconds (GB-s) of computing time per month. Beyond this limit, function executions are charged per GB-s of memory allocated and the number of execution requests. Lambda allows the developer to specify a maximum memory limit that the function will have access to, at the deployment stage, and allocates CPU power proportionate to the allowed memory limit [109]. AWS enables applications to be executed as composed of a single function or multiple functions. AWS Step Functions [109]

facilitates developers to define workflows with a sequence of Lambda functions and create a state machine that orchestrates the set of functions in the application. AWS uses a new virtualization technology called Firecracker, to execute function requests [1]. These are lightweight micro-virtual machines (microVMs). AWS Lambda treats instance placement in VMs as a bin packing problem to maximize VM memory utilization. AWS has set a concurrency limit for scaling a single function and experiments show that idle instance recycling is done based on a pre-defined inactive period [131].

**2.3.2 Azure Functions.** Microsoft launched its serverless services in 2016 as Azure functions. Azure supports a number of runtime languages such as C#, Node.js, PHP, Bash, Power Shell. The billing scheme is similar to that of AWS except that billing is done per GB-s of average memory consumed during an execution instead of the memory allocated. Azure uses the concept of function app as the unit of deployment and management of a serverless application. A function app is comprised of one or more functions which are deployed, managed and scaled together. All the functions in a function app share the same pricing plan and runtime configurations [9]. Azure Functions seems to try not to co-locate concurrent instances of the same function on the same VM, which indicates a spread placement approach [131]. Azure currently supports the execution of stateful functions as well in a serverless computing environment with Azure Durable Functions, which is an extension of Azure Functions [22].

**2.3.3 Google Cloud Functions (GCF).** Google released its serverless solution in 2017. GCF supports code written in Node.js, Python, Go, Java, .NET, Ruby and PHP. Google has a free tier of 2 million requests with 400,000 GB-s of computing time per month, as of now. Their pricing scheme is different to AWS Lambda and Azure Functions pricing mechanism since the user is charged for both GB-s of memory provisioned and GHz-s of CPU provisioned [49]. GCF supports a set of primary triggers and additional triggers and the user application is allowed to be integrated with any Google service, supporting cloud Pub/Sub or HTTP callbacks.

**2.3.4 Apache OpenWhisk.** OpenWhisk is an open source serverless platform developed by IBM and later incorporated as an Apache incubator project. It represents the underlying technology used in IBM Cloud Functions. OpenWhisk combines technologies such as Nginx, Kafka, Docker and CouchDB in forming its serverless platform. OpenWhisk supports many deployment options and could be deployed both locally and within a cloud environment. The OpenWhisk programming model is based on the three primary concepts, actions, triggers and rules. Actions are functions that execute deployed code. Triggers are a set of events created from different sources. Rules bind actions with triggers. OpenWhisk supports a number of runtimes including, .Net, Go, Java, JavaScript, PHP and Python [95]. In selecting a VM instance for a function execution, OpenWhisk follows a hash-based first-fit heuristic where the function name's hash value is used to identify a preferred host in order to improve reusing warm containers [125]. Multiple functions, which may even be implemented in different languages, could be composed together to create a function pipeline called a sequence. A sequence could be considered a single action in terms of its creation and invocation. In executing a function sequence, the output from one action becomes the input to the next action in the sequence [95]. In addition, function sequences could be extended to create custom workflows with conditional branching, with support for error handling, retries and loops by means of the OpenWhisk Composer [34]. OpenWhisk makes use of pre-warmed containers and warm containers (container re-use) to manage unwanted container startup delays.

**2.3.5 Kubeless.** Kubeless is a Kubernetes-native open-source serverless framework developed by Bitnami [76]. Kubeless creates functions as a custom Kubernetes resource using a Custom Resource Definition (CRD). An in-cluster controller is used to watch these custom resources and

execute functions on-demand by dynamically launching runtimes as required. Kubeless supports runtimes of Golang, Python, NodeJS, Ruby, PHP, .NET and Ballerina and allows to use HTTP or event triggers to invoke functions. The Kubeless framework uses core Kubernetes functionalities such as Deployments, ConfigMaps and Services as it is. It also leverages the native Kubernetes components for function scheduling, auto-scaling and monitoring as well.

**2.3.6 OpenFaas.** OpenFaas started as an independent project by Alex Ellis in 2016 [94]. Initially, it was developed in collaboration with VMWare and now it involves a large community of users and developers. OpenFaas framework is built on Docker and Kubernetes, and could be deployed on a private or public cloud environment, and even on a resource constrained edge device such as a Raspberry Pi, due its lightweight nature. OpenFaas provides an API gateway for invoking functions, which can be accessed via its REST API, Command Line Interface (CLI) or the User Interface (UI). The gateway acts as an external route to the functions, collects cloud native metrics through Prometheus [99], and also takes function scaling decisions with the help of Prometheus and an AlertManager component. AlertManager works by reading the requests per second metric from Prometheus and alerting the gateway to scale functions based on the min/max replica count set at function deployment. Alternatively users could use the built-in Horizontal Pod Autoscaler (HPA) of Kubernetes. OpenFaas offers runtimes of Node.js, Python, and Go for function deployment. OpenFaas also supports the orchestration of multi-function workflow applications with synchronous and asynchronous function chains and parallel branching.

## 2.4 Application Domains

In general, bursty and compute-intensive workloads which are stateless and ephemeral could benefit more from a serverless architecture. From a cost perspective, the auto-scaling feature of the platforms proves useful, when traffic arrives in bursts (inconsistent traffic levels), since the system can also scale to zero when there is no traffic [15]. During recent times, serverless computing has increasingly being explored for use in many applications domains such as web and mobile, big data, internet of things, machine learning model training and large scale mathematical computations. The nature of the serverless workload will vary depending on the different requirements of these domains and their inherent characteristics. The focus of the resource management techniques needs to be adapted accordingly. As discussed in the following sections, the core design features of the existing platforms make the serverless model easily adaptable for some domains while in others, an extra effort is needed to reap the full benefits of this novel model.

**2.4.1 Web Services.** Serverless model is said to have been initially developed for lightweight use cases like serving APIs or small backend services [77]. Even today, studies on serverless use cases reveal that web services is the most common application domain utilizing this new computing paradigm due to ease of adoption [45].

**2.4.2 Big Data Analytics.** Due to high data volumes and computational requirements, big data processing is traditionally undertaken using clusters of machines with jobs consisting of multiple tasks being executed across a set of machines. Advancement of technology in the field has addressed these performance and scalability needs through distributed computing frameworks specialized for big data analytics [137], [23]. The costs and the knowledge required for configuring, deploying and maintaining these systems is still a challenge. The lower startup times, automated resource management, function level auto-scaling and granular billing features present an interesting opportunity in the serverless model for big data processing [77]. The potential of serverless for big data analytics is being recognized by the major cloud providers as well. AWS provides guidance on reaping benefits of Lambda for streaming data analytics [109]. IBM introduces IBM-PyWren, a data

analytics platform using IBM Cloud Functions [105]. A number of research efforts are seen trying to adopt the serverless model for obtaining favorable results for big data applications. A prototype serverless Spark execution engine using AWS Lambda as the Spark cluster is presented [73]. FaaS model is explored for various MapReduce jobs [56], [47]. However, many fundamental challenges still exist that impede the performance of distributed data-driven applications on current serverless platforms. The data-shipping architecture where function execution is completely isolated from data, and the short-lived, stateless nature of functions makes it a requirement to routinely ship data to code, for each function invocation. Functions are also usually not network-addressable and thus two functions could only communicate through slow and expensive storage. Current FaaS platforms lack access to any specialized hardware and functions are only provisioned with the processing power of a timeslice of a CPU hyperthread [62]. A serverless framework for data processing is introduced in [90] focusing on a few of these challenges. The authors propose an exchange operator to support inter-function communication, a batch-start approach to reduce start up time of a large number of parallel workers and a low cost scan operator for reading query input efficiently. A task scheduler for serverless data analytics applications, to minimize job completion time and cost, is proposed in [140].

**2.4.3 Internet of Things.** FaaS model could be beneficial for adoption in IoT applications in edge/fog computing networks owing to a number of factors. Deploying applications as a number of light-weight functions go in line with resource limitations at the edge devices. Statelessness of serverless functions add portability for parts of applications to be moved across the edge/cloud computing network with lesser complications. As a result, today, serverless computing has been exploited in many IoT domains including home automation and other custom-built IoT solutions. AWS offers AWS IoT Greengrass [113] which is included as a service in the serverless ecosystem as well and this allows processing data at the edge. Azure IoT edge [10], which could be used in conjunction with their FaaS service Azure Functions, moves cloud analytics to the edge devices. A number of researches focus on introducing frameworks and scheduling techniques for serverless applications deployed in edge-cloud computing networks [46], [19], [97], [37], [16], [29]. To date, challenges do exist in adopting the serverless model in IoT scenarios. Retrieval of large data sets to the edge dynamically, each time a function is called is not practical specially with latency sensitive applications. Thus arises the need for a caching service at the edge. Geographic dispersion, heterogeneity and affinity of data sources at the edge suggest the need for more decentralized resource pooling and scheduling techniques devised specially for IoT application scenarios.

**2.4.4 Machine Learning.** Machine learning (ML) applications typically consist of three phases: model design, model training and model inference (model serving). VM clusters were traditionally used for the diverse tasks in ML model training. The distinct stages of a ML workflow pipeline consist of varying computational requirements. As such the traditional approaches face several challenges such as the need for developers to provision, configure and manage these resources and the over and under provisioning of VM resources during different stages of the execution of a training workflow [68]. On the face, serverless computing seems to be a promising approach for resource provisioning challenges for ML users, in terms of its simplified deployment opportunities, fine-grained resource provisioning and billing models and the ability to auto-scale both computation and storage resources. However, the small local memory and storage footprints of serverless functions, lack of function to function communication or fast shared storage, the short-lived function runtimes and unpredictable launch times make serverless platforms less conducive for ML processing as it is [24]. Research is being undertaken in this area to propose serverless frameworks with potential developments that could overcome these specific limitations. Carreira et al. [25] introduce a framework for ML training pipelines on serverless computing. Results outperform a VM-based execution in terms of

training completion time, but at a relatively higher cost. Out of the three phases of ML applications, Ali et al. [5] identify model serving as having the greatest potential to benefit from serverless, since prediction request arrival is usually dynamic and the requests have strict latency requirements. They show that by devising a strategy to enable batching (i.e., bundling of several ML inference requests and serving them together), ML serving inference process could benefit significantly from serverless.

**2.4.5 Mathematical Computation.** Large scale mathematical computations are traditionally deployed on supercomputers or high-performance computing clusters connected by high-speed, low-latency networks [68]. Considering this, serverless seems a poor fit for such applications. However, the ability to unburden non-computer scientists of having to manage infrastructure and scalability to support varying needs of resource parallelism during a computation, highlights benefits of a FaaS model over managing a cluster with a fixed size. Shankar et al. [117] present Numpywren, a serverless system for linear algebra computations. Werner et al. [132] present a prototype for matrix multiplication using FaaS. These experiments show that serverless computing could be a good fit for large scale linear algebra (e.g., matrix multiplication, singular value decomposition) when computation time dominates communication delays. But the high latency of external storage causes limitations for smaller problem sizes.

## 2.5 Resource Management in Serverless Computing Environments

The core differentiator of the serverless model from other computing models is the complete shift of server management responsibilities to the vendor, thus rendering the model 'serverless', from the perspective of the developer. A major, if not the primary aspect of server management, lies in the process of proper management of server resources for the execution of applications. Resource management in a serverless environment refers to the overall aspect of managing the resource requirements of an application workload and the available system resources efficiently, with minimal involvement of the user. Due to the autonomic nature of the expected resource management process in these environments, special focus is required on each step of this process for better performance of the applications and the system. We identify three major aspects of resource management, which need to be dealt with in a manner suitable for this new serverless computing model.

- **Workload characterization and performance prediction:** Developers prefer minimal work in using a serverless deployment model. Having to specify a resource configuration and other characteristics when deploying an application could be cumbersome. Hence it is ideal for a serverless platform to be able to infer application as well as workload characteristics using modelling and bench-marking techniques, which could then be used for performance prediction. An effective scheme for developing such an understanding, leads to better resource scheduling and scaling decisions as well, which help in satisfying user QoS requirements.
- **Resource scheduling:** Mapping workloads to suitable host nodes based on their resource requirements, while efficiently utilizing the available resources is an important challenge to both the developer and the cloud providers or system owners. Scheduling also involves determining the order of execution of applications when the resource demand exceeds the available resource capacity. While the developer would expect certain QoS guarantees, it is essential for the provider to manage resources efficiently, which is a primary goal of resource management.
- **Resource scaling:** Under the serverless model, environment creation and resource allocations to applications happen in real time, as and when workloads arrive. This ensures greater flexibility in resource allocations and higher resource efficiency. In order to maintain required

application performance while maintaining scaling at such a granular level requires smart and dynamic resource scaling techniques.

*2.5.1 Challenges in Resource Management.* Below we have identified challenges associated with resource management strategies which are specifically significant in a serverless environment. We have analyzed the stated challenges from the perspective of end users and the service providers.

- **Cold start delay:** Due to the auto-scaling nature of these environments, resource creation needs to happen on the go and setting up new resources for a function execution results in a considerable start up time. Applications often face performance degradations due to this initial delay, which becomes specially significant for functions with very short execution times. On the other hand, maintaining idle resource pools to alleviate this issue often results in wasted resources on the provider side.
- **Co-located application interference:** Applications deployed on serverless platforms run in multi-tenant environments, inside specially created isolated environments such as containers. When several applications run on the same host node and compete for the same set of resources, it is difficult to fully avoid resource interference effects on the applications. In extreme cases, if resource interference is not handled properly, dominant applications could consume all the resources. This would lead to very poor performance for other user applications. Developing sandboxing techniques which are light-weight enough to avoid large setup times and secure enough to provide the required level of resource isolation is an associated challenge for cloud vendors.
- **Resource efficiency:** In contrast to a general cloud computing billing model, serverless systems usually charge only for the resources allocated/consumed during the application execution, with a millisecond accuracy. The provider on the other hand may be maintaining the underlying infrastructure for longer periods. This creates the need for special attention to have strategies for high resource efficiency on the host nodes. On the user side, in order to avoid poor application performance, there is a tendency to overbook resources for function executions. Frequent under utilization of these resources could lead to poor value for money for the user and lack of confidence on these services in the long run.
- **Diverse workload management:** With serverless systems, there is minimal user involvement in the resource management process. Thus, these systems need to develop an understanding on the application and workload characteristics on their own, in order to deliver a favorable outcome. The diverse nature of applications that are being deployed on serverless platforms makes this a challenging task. The lack of an understanding on the application requirements and characteristics could result in delays in resource setup time, heightened resource interference effects, etc. that lead to customer dissatisfaction.
- **Runtime limitations:** To alleviate the risk of losing flexibility of the infrastructure, serverless providers impose various runtime limitations on applications. These include limitations on the maximum allocated CPU, memory, disk, I/O resources as well as maximum allowed execution time for a function instance. Platforms also configure an upper limit on function concurrency levels which limits the number of parallel executions of the same function by a user. Although successful in preserving flexibility of the system, these limitations often deem the serverless platforms unsuitable for long-running applications with highly compute-intensive workloads.
- **Lack of QoS guarantees:** The blackbox nature of the serverless systems means that the operational aspects visible to end users is minimal. Although this is generally convenient to most users, for certain high-precision, latency-sensitive applications, these environments could be unusable without precise performance guarantees. For the provider, guaranteeing

performance to each and every user in a shared environment is a very complex task requiring a lot of attention.

- **State management:** The fundamental building blocks of a serverless architecture naturally leans more towards short-lived and stateless task executions. Although currently some service providers offer solutions for orchestrating stateful workflows, generally developers are expected to work with external storage services for intermediary state management. Having to frequently retrieve data to and from these storage devices with added delays in the network poses many challenges for the execution of latency critical data-intensive applications. On the other hand, since flexibility in resource management is a distinguishing quality for a serverless environment, addressing the issue of state management to the satisfaction of all users is a highly challenging task for a provider.

**2.5.2 Motivation.** The three aspects of resource management identified above need to be addressed considering the aforementioned associated challenges. Various approaches have been experimented by researchers to overcome these challenges and devise better workload characterization, resource scheduling and scaling techniques. In this paper, we identify a classification of the factors which influence these decisions, by reviewing the existing related literature. In the classification, we discuss challenges inherent to this new computing model, as well as the general concerns to resource management in the context of serverless computing.

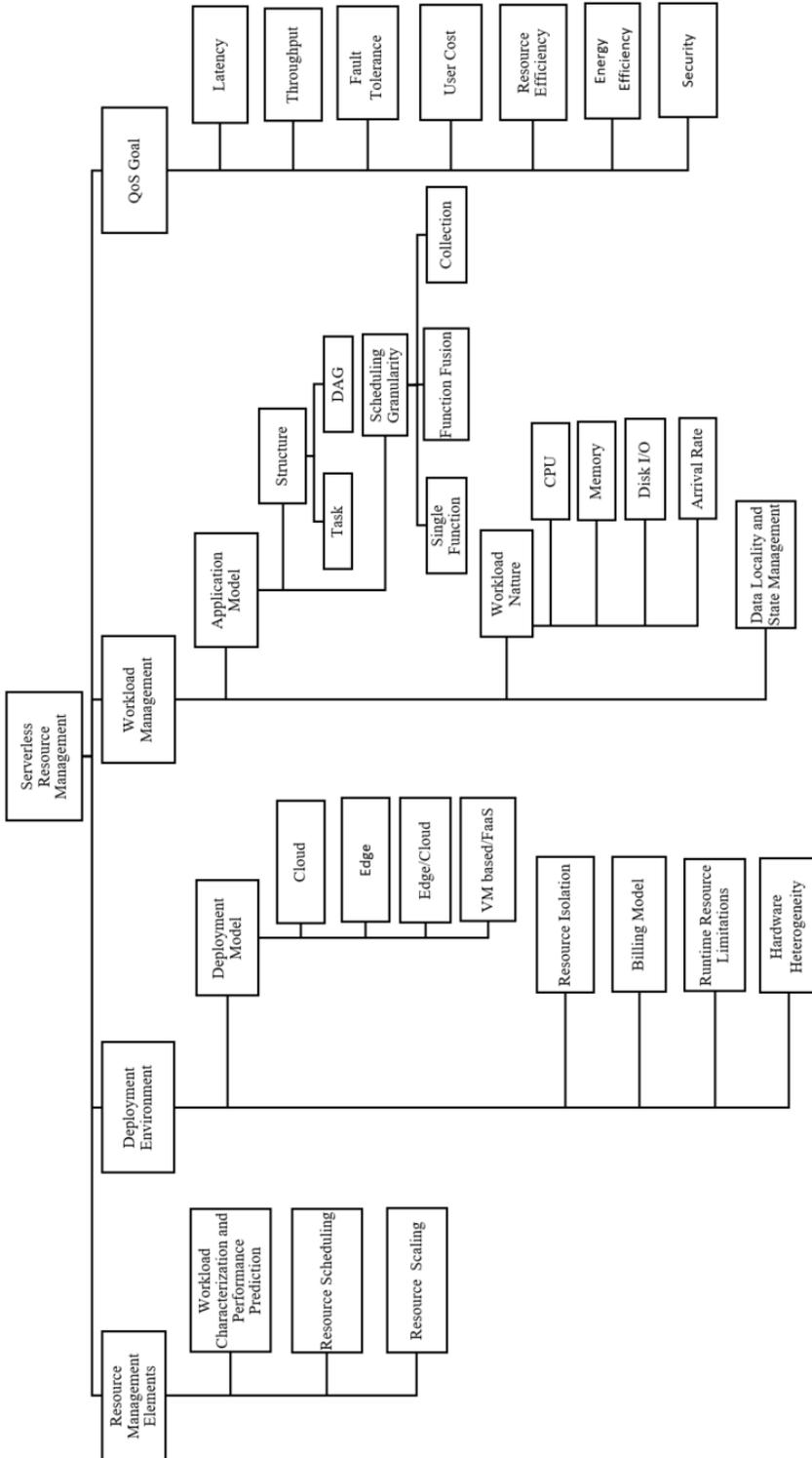
Successfully managing resources in any system is determined by the underlying system design features as well as the understanding on the characteristics of the incoming workloads. As such, our classification comprises of a reference to the key design aspects related to resource management in a serverless system, characteristics of the workloads submitted to these systems, and finally, the primary goals of resource management in these environments. Further, we summarize the existing research work related to workload modelling and prediction, resource scheduling and scaling techniques in the serverless domain, using our proposed taxonomy. We also propose ideas for future work in developing enhanced techniques. Serverless system designers would benefit from our classification by gaining an understanding on the key focus areas of a system design under this computing model and also the existing approaches that have already been evaluated. Researchers studying resource management techniques would be able to refer to existing approaches which could subsequently form the basis for the design of novel and rigorous techniques in the future. This classification would also assist application developers in the design of their serverless applications, choosing the right infrastructure and in budgeting their deployments. The main contributions of this work are as follows:

- We identify three major elements of resource management in serverless environments.
- We propose a taxonomy of the factors which influence resource management under this computing model, by reviewing existing literature on the above identified elements.
- We summarize the existing works on serverless resource management in the context of the proposed taxonomy.
- We identify research gaps in different aspects of serverless resource management and propose ideas for future research directions.

### 3 THE TAXONOMY

The top level of our taxonomy is comprised of: the key elements of resource management, deployment environment, workload management and QoS goal. Figure 4 illustrates the proposed taxonomy. We discuss each category in detail in the following sections.

Fig. 4. The Taxonomy of Resource Management in Serverless Computing Environments



### 3.1 Elements of Resource Management

In this section we briefly discuss the techniques explored in literature so far, under the three primary elements of resource management that we have identified. All the factors discussed in detail in the rest of the sections in the taxonomy drive the development of techniques for these key areas.

*3.1.1 Workload Characterization and Performance Prediction.* Workload characterization refers to using empirical and theoretical approaches to understand and model, the varying resource requirements of different applications, workload arrival patterns and run time behavioral patterns. The knowledge gained in this manner is subsequently used to derive latency and cost models for the system and the applications, which is known as performance prediction. The whole process of developing performance models using the understanding on workload characteristics could be referred to as workload modelling. In the existing literature, mathematical modelling techniques are used in abundance in developing workload characterization techniques and prediction models in serverless environments.

Singvi et al. use a method of exponentially weighted moving average over the measured request arrival rate to get a new estimate of the workload arrival rates [120]. Gunasekaran et al. use a moving window Linear Regression (LR) model to predict the average request rate in their VM/FaaS hybrid model, in order to provision the required VMs in advance [58]. In [116], the authors characterize the entire production workload of Azure Functions. They identify each function in terms of its trigger type, execution time, resource usage and invocation frequency. This information is subsequently used to develop a hybrid histogram and time-series model to reduce cold start invocations. In addition to these research efforts, a number of works have also focused on composing benchmark suites for serverless workloads. Kim et al. present a suite of function workloads which consume CPU, memory, disk I/O and network resources in varying levels [72]. Their collection of applications consists of micro-benchmarks as well as data-oriented realistic applications which would assist in realizing a multitude of application scenarios for research works in the field. Grambow et al. design an application-centric benchmarking framework for serverless use cases [57]. It is built in a flexible manner to accommodate a variety of applications and also to suit federated cloud and edge scenarios.

A mixture density network based model is used to predict function response time in [43]. Based on the resultant model, a Monte-carlo simulation is used to do cost prediction for serverless workflow executions. This approach takes into consideration, the effects of input parameters to each function, on the execution times. Similarly, application latencies parameterized by the function input data sizes and framework overheads, are modeled using Regularized Ridge Regression in their work by Das et al. [38]. They use training data obtained by running a substantial number of jobs in AWS Lambda, and the OpenFaaS platform deployed in the private cloud. Application specific performance models are developed in [37] which are able to predict application latency and costs for various container configurations under an edge/cloud environment. They consider network transfer times, container start up times, function execution times and storage latencies in building their models and use various techniques of regression over the training data sets. For estimating application latency in the cloud, they identify Gradient Boosted Regression Trees to be the most robust. These performance and cost models enable application developers to plan and budget their deployments in the most cost effective manner. Lin et al. propose an analytical model to predict response time and cost for workflow execution in serverless platforms [81]. They consider variations in response time and cost with the allocated memory in building these models. Performance and cost predictions under different resource configurations in serverless settings are explored in [2] and [36]. Mahmoudi et al. also propose an analytical model to help developers to extract performance metrics for their applications before the actual deployment [84]. In particular, their model enables the calculation

of the cold start probability, average response time and the required average number of function instances, under stable conditions. Linear Regression is used in [59] for function execution time prediction which is then used in a Least Slack First (LSF) algorithm to select a request for execution from the queue. Zhang et al. use regression techniques to determine the total latency of executing a batch of serverless tasks over the edge and cloud runtime environments, in order to determine the runtime with the least latency [141]. In their work, median sliding window time series modelling technique is used to predict runtime deployment time while Bayesian Ridge regression technique is used for processing time estimation.

Application performance is also largely affected by the characteristics and behavior of the underlying serverless platform. Platform characterization and evaluation is thus an important factor in application performance prediction process. Yu et al. present an open-source benchmark suite for exploring a number of commercial and open-source serverless platforms in terms of their function startup latency, performance isolation, communication efficiency, CPU contention etc [135]. Benchmarking frameworks have also been developed accommodating experiments on multiple existing serverless platforms, for ease of comparing their runtime behaviors and performance levels [87], [35]. Mahmoudi et al. present a simulation framework for serverless application developers to develop, test and optimize the performance and cost of their applications [85].

**3.1.2 Resource Scheduling.** Application scheduling in a serverless environment addresses the challenges of decision making with regard to resource provisioning, resource allocation and scheduling of function invocation requests. A comprehensive resource scheduling scheme would make use of performance prediction tools as identified above, for determining the optimal level of resource allocations and the scheduling policies to meet consumer and provider expectations.

Constraint programming-based approaches (e.g., Integer Linear Programming (ILP), Mixed Integer Linear Programming (MILP)) try to minimize or maximize an objective by satisfying the set of constraints for the function execution and the restrictions of available resources. Scheduling serverless functions over the underlying infrastructure of a serverless platform based on user Service Level Agreement (SLA) is a NP-hard problem. Therefore algorithms and approaches for obtaining optimal scheduling decisions may not be feasible for these platforms considering the magnitude of the problem caused by the scale of resources. In contrast, using heuristic approaches is faster and they are scalable to large clusters. These approaches are able to provide acceptable performance and near-optimal solutions. Schedulers in commercial and open-source serverless platforms also seem to be using simple load balancing mechanisms such as round robin and bin-packing approaches [131], [125]. A considerable set of works exist in literature which study the applicability of various heuristic based approaches for function scheduling in serverless environments [88], [127], [82], [120]. In their work, Pinto et al. explore the applicability of Bayesian Upper Confidence Bound (UCB) algorithms for deciding the optimum location for the execution of functions in an edge/cloud network [97]. Elgamal et al. focus on the problem of finding the most cost effective way of fusing multiple functions and placing them, either on the cloud or the edge [46]. The problem is formulated into a Constrained Shortest Path (CSP) problem and they find the Lagrange Relaxation-based Aggregated Cost (LARAC) algorithm to be the most efficient for solving the problem. Palma et al. present a novel idea of a declarative language which the developers can use to state a scheduling policy and performance goals for their functions [40]. Later, the scheduler uses this policy to choose the best fitting worker for the execution. Although at very initial stages, learning based approaches such as machine learning, and Deep Reinforcement Learning (DRL) methods are also being increasingly explored for resource allocation and scheduling decision making in these environments [108], [86].

**3.1.3 Resource Scaling.** The ability to scale resources automatically to meet time varying application demand levels is a unique feature of paramount importance, under the serverless deployment

model. In order to achieve high resource efficiency, the underlying resources are expected to scale-out as required, when there is a rise of demand for an applications and scale-in with diminishing demand. As such, an application would scale to zero with no resource consumption, at times when there is no demand for application execution. This ensures that the user is only billed for the exact resource consumption during application execution. In turn, this relieves them of incurring costs for over provisioned idling resources during demand drops, which is often a critical issue in traditional serverful deployments. A good approach to scale resources also needs to ensure that the QoS requirements of the user as well as the provider are sufficiently met.

Resource scaling or elasticity of resources in cloud, edge and fog environments, usually refers to two dimensions as horizontal and vertical scaling. Horizontal elasticity of resources allows to scale-out or scale-in the number of application instances, while vertical elasticity allows to scale-up/down the amount of computing and other resources assigned to each application instance [104].

- *Horizontal scaling*: The existing commercial serverless platforms host containerized function instances in readily available VMs. Hence the horizontal scaling of serverless applications is affected by the availability of VMs with required resources. Experimentation done on AWS Lambda show that there is no significant change to cold start delay, when a new function instance is launched on a new VM previously not used for executions, and an existing VM [131]. This indicates that the service providers usually maintain a pool of ready VMs for function executions and thus VM start up time is not usually a determining factor for function scheduling approaches. But recent interest in using the serverless model for compute intensive workloads such as predictive analysis applications using pre-trained deep learning models, present situations where dynamic scaling of VM resources is required to manage the dynamic workloads with varying resource requirements [21].

Serverless platforms often utilize containers as the sandboxing mechanism for the isolation of applications from each other. Each function instance is usually run on a separate container with the required resource configurations. Thus, prior to application execution, the required resource setup generally includes launching a new container, setting up the runtime environment and application specific initialization. The time taken for all these steps is collectively knows as the cold start latency. Generally the coldstart latency of containers, which are lightweight resource units, is in the order of milliseconds. But studies have shown that in serverless executions, this delay is largely dependent on each function's runtime dependencies and at times could grow to be even a few seconds [131], [127]. In order to attain the intended benefits of serverless auto-scaling abilities, it is a necessity that the function cold start latencies are managed appropriately.

To alleviate the frequency of cold starts, serverless platforms often try to either reuse warm containers or create pre-warmed containers. The reuse of warm containers avoids any setup and initialization delays while pre-warmed containers avoid container launching delays. AWS, Azure and Google Cloud Functions maintain idle function instances for a particular time preiod, before they are recycled, in order to increase chances of container reuse [131]. Apache OpenWhisk maintains pre-warmed containers and also uses load balancing strategies to direct similar function executions to the same set of workers as much as possible, thus enhancing chances of container reuse [125]. In existing research work, many models are presented to predict the arrival rates of incoming function requests and the demand for particular function executions and thereby proactively setup and maintain a pool of warm containers across VMs [126], [120], [82], [18], [121], [133], [39]. Subsequently, load balancing algorithms are devised to benefit from these existing resource pools. In contrast to these works, Mohan et al. identify the processes of network creation and connection to be the major bottlenecks in container startup and propose a method for maintaining a pool of pre-created network elements which could be attached to a new function container whenever

needed [89]. This is done by using the concept of pause containers, which are network-ready empty containers which could be attached to other containers. Similarly in [119], Silva et al. try to reduce the container startup time by implementing a checkpoint mechanism for restoring snapshots from recently created function processes. Stein et al. propose a non-cooperative resource allocation heuristic for serverless environments which aims to predict the number of function instances required to be kept in order to maintain request waiting time below a threshold level [126]. Somma et al. use a Q-Learning based approach to determine the ideal number of function containers to scale-up/down at each instance, to maintain high resource efficiency and low application latency [123]. Gias et al. compare the idle time of a function instance in a FaaS platform to the Time To Live (TTL) value of a TTL caching system [53]. They present a model to decide on the most suitable idle time that each function instance is to be maintained in the system so as to meet the function response time requirements of the user. At times the reuse of warm containers may even cause additional latencies for example, when already fetched data in a function instance is out-of-date and required database connections have already reverted by the time a new request arrives at the container. Mechanisms to freshen up the warm instances prior to use is of importance in such instances [64].

The concurrency level, which determines the maximum number of requests that the system could process in parallel for each different function, is also an important factor in function scaling. Existing commercial platforms have set fixed limits on concurrency levels for a particular function [131]. Schuler et al. present a Q-learning based Reinforcement Learning (RL) approach to determine the best level of request concurrency, to achieve better performance in terms of system throughput and mean function latency [108].

- *Vertical scaling*: The core concept behind the serverless paradigm is to shift the complexities of application resource management from the developer to the service provider. Thus, the provider has the responsibility to autonomously manage application resource allocations as required, in contrast to allocating resources as per a detailed resource request under a serverful model [68]. For instance, AWS Lambda requires the user to only provide the amount of memory to be allocated per function instance and CPU power is stated to be allocated linearly in proportion to the requested memory [109]. CPU is known to be a source of contention in serverless environments [127], affecting application latencies. Hence, resource allocations to applications need to be monitored during runtime, to avoid potential SLA violations to the user. Further, the providers need to carefully manage the CPU and memory resources allocated to applications in order to achieve resource efficiency and avoid over/under provisioning of resources.

Adjusting CPU allocations to function instances in the runtime is a new research area in serverless computing. Suresh et al. study the impact of dynamically adjusting CPU-shares to containerized function instances in the runtime [127]. CPU-shares indicates the relative weight given to a container in terms of the proportion of CPU time it is given access to when CPU resources are limited [41]. As per their model, containers are launched on VMs when spare memory capacity is available to accommodate the container. Thus multiple containers co-located on a VM would share the same processor core and thus the CPU time available to a container varies over time. This could hinder satisfying user latency requirements to certain applications. Experiments show that fine-tuned regulation of the CPU-shares allocations to containers dynamically could result in better QoS satisfaction. In a previous work [88], we proposed a technique for dynamic CPU resource management to containers running serverless functions by applying the concept of cpu-quota and cpu-period enabled in Linux Kernel's Completely Fair Scheduler (CFS) [129]. The cpu-quota value sets the number of microseconds per cpu-period that the function instance's access to CPU resources is limited to, before it is throttled [41]. Thus this acts as an effective ceiling and a hard limit for CPU resources allocated to a function instance. This technique helps in allocating a guaranteed

CPU time for a function execution and also fine-grained management of underlying resources. Yu et al. propose a DRL based technique to dynamically harvest CPU and memory resources from idle functions and allocate them to under-provisioned functions [134]. As requests arrive, the agent evaluates the cluster state and each function's past cpu and memory consumption, request arrival rates etc., and decide on a new resource configuration for the existing containers.

## 3.2 Deployment Environment

The design of a serverless platform needs to address the inherent unique features offered by this computing model and also the associated key challenges. We characterize these factors in terms of the platform deployment model, resource isolation techniques, the incorporated pricing models, the nature of imposed runtime resource limitations and hardware heterogeneity. Identifying these influencing factors help in developing effective algorithms, system models and other techniques for the efficient management of resources in serverless environments.

*3.2.1 Deployment Model.* Serverless platforms could be deployed on public cloud, private cloud or on edge resources. The serverless model could also be used in conjunction with a serverful model, i.e.: a VM based deployment. The deployment model is concerned with the nature of infrastructure combination from each resource environment, the associated resource capabilities, pricing models and subsequent resource management decisions suited for each environment for better QoS. This section is focused on cloud and edge only deployments as well as hybrid deployment models with resources used interchangeably from different environments for better performance and efficiency.

- *Cloud:* This is the most common as well as the basic deployment model targeted by serverless computing when it first emerged. A cloud environment could be an on premise, user managed, private environment or a vendor managed public environment with seemingly unlimited resources. A private cloud environment gives better flexibility to the user to device scheduling techniques to achieve desired performance guarantees. In addition, privacy is less of a concern. While these environments usually enable reaping cost benefits in the long run for regular workload scenarios, the downside is the inability to meet sudden demand surges if additional hardware is not maintained. With a private/public hybrid cloud model, whenever the load goes beyond the capacity of local infrastructure, some of the functions could be dispatched and processed in the public cloud at a cost. In a hybrid model, one could use a commercial serverless provider's service together with an open source serverless framework deployed on a private network. The serverless scheduling challenge then is to decide when and which functions are to be off-loaded to the public cloud for better QoS guarantees and cost benefits. These decisions need to consider the data transfer times across networks, resource setup times, along with the load levels and required QoS guarantees. Das et al. [38] propose a hybrid cloud scheduling framework for multi-function serverless applications with AWS Lambda as the public cloud and OpenFaaS [94] deployed in the private cloud. The portable nature of the serverless workloads also makes it a viable candidate for adopting multi-cloud architectures to harness the benefits of different providers [13]. Liu et al. present a JointCloud [130] platform for serverless computing, which is capable of coordinating resources of multiple cloud vendors for request executions [83]. Service from high performant clouds is requested for latency-sensitive applications while cheaper clouds are used for jobs with lesser requirements. A primitive idea for a federated ecosystem is proposed in [30] for a decentralized serverless architecture for balancing load traffic at the edge. Further studies on the design of multi-cloud architectures in a serverless setting are seen in [122], [67].

- *Edge:* Edge computing leverages computing power and storage facilities close to the consumer in order to reduce application latencies and bandwidth usage. As discussed previously under serverless use cases, serverless computing seems to be a good fit for the deployment of applications

across resource constrained edge computing networks due to the ephemeral nature and portability of serverless functions. Gand et al. propose an architecture for clustered container applications for the edge, based on serverless computing [51]. Baresi et al. propose a serverless edge computing architecture for mobile applications with low latency and high throughput requirements [17]. They use mobile devices and mobile edge computing servers as their main computational elements and evaluate the feasibility of their approach via a mobile augmented reality use case. Containers which are often used as the isolation mechanism for serverless functions, are at times not viable for edge environments with limited resource capabilities due to their inherent setup and runtime overheads. Hall et al. propose a novel run time and an isolation mechanism for serverless functions called WebAssembly which reduces resource startup times and the resource provisioning requirements [61].

- *Edge/Cloud*: A hybrid deployment model leveraging resource capabilities of both the edge and the cloud is often times the most viable solution under many practical scenarios. But many challenges exist, for realizing the true benefit of such a setting. For an application deployed in a serverless edge/cloud infrastructure, which functions are to be deployed on the fog or cloud resources and which node containing the deployed function is best suited for accommodating a new request is a scheduling decision. The resource allocation and scheduling decisions for serverless applications in an edge/cloud computing network will depend on how compute intensive the function is, the size of data involved and data transfer costs over different network paths, the cost and the setup time of resources at each location, and the expected QoS levels [46], [97], [19].

- *VM based/ FaaS*: Traditionally VM based deployments were used to dynamically scale resources as per the demand. However challenges exist such as higher costs due to over-provisioning or SLA violations due to under-provisioning. Under the serverless model, functions are auto-scaled within containers which have a low startup latency. Further, user is billed per function invocation at a very granular level, thus avoiding resource over-provisioning costs. However, it is observed that deploying an entire application as serverless functions would not be cost effective at times. Lambda functions are expensive (cost increases linearly) when there is a fixed load (constant request arrival rate for a function) and a higher average request rate. Thus it is seen that functions are more cost efficient with demand variations and lower average request rates while VM based deployments serve better with higher arrival rates. Exploring hybrid models of VM and FaaS deployments is an interesting dimension [58]. Li et al. present a serverless architecture for dynamically switching the deployment of a microservice between FaaS and IaaS infrastructure [80]. The decision is taken in a contention-aware manner, based on the predicted performance of the service on a FaaS platform.

**3.2.2 Resource Isolation.** Serverless systems are multi-tenant systems and thus techniques for isolating applications from each other is important for reasons of both application performance and security. Container technologies are used as the sandboxing mechanism for function executions by many commercial serverless platforms. Studies show that co-located function instances show effects of resource contention with regard to CPU and network bandwidth, raising concerns on the effectiveness of the isolation mechanisms [131]. By design, components isolated by operating system (OS) level virtualization techniques (e.g., containers), share hardware and the host's OS kernel and thus are open to security vulnerabilities. AWS uses MicroVMs which are hardware-isolated lightweight virtual machines with their own mini-kernel [1]. They offer security and workload isolation from hardware-virtualization as with VMs, and the resource efficiency and smaller startup times of containers. Existing work in this area propose customized sandboxing techniques for function execution. Arguments are laid suggesting that stronger isolation mechanisms such as containers are needed only among different applications and weaker mechanisms such as having separate processes is sufficient for isolation among functions of the same application. Akkus et

al. propose a new sandboxing method where different applications run inside separate containers, but different functions of the same application run inside the same container in parallel processes [3]. Concurrent calls to the same function are handled inside the same container by spawning new processes by incrementing the memory allocation to the container. Since the memory footprint of a process is smaller than that of a container, this results in higher resource efficiency. Further, forking a new process inside a container only incurs a short startup latency. Oakes et al. present a container system optimized for serverless workloads [93]. Bottlenecks of container cold starts are identified to be caused by limitations in Linux isolation primitives and loading dependent packages.

**3.2.3 Billing Model.** Billing model refers to how costs are calculated for function executions in serverless environments. Since serverless offers a very fine-grained billing model, it is important to understand the fine details of billing criteria for different resources. This helps to make cost efficient resource allocation and scheduling decisions for both the provider and end user.

Serverless platforms offer a pay-as-you-execute billing model usually based on the CPU, memory and storage resource costs associated with each function execution. Additional costs may also be involved based on the nature of the application scenario and additional services consumed (e.g., state transitions costs of AWS Step Functions, Amazon Simple Queue Service (SQS) costs etc.). Although the basic billing model on the major commercial serverless platforms is the same, differences exist in finer details. Above its monthly free tier, AWS Lambda charges function executions per GB-s (rounded up to the nearest 1ms) of memory allocated and the number of execution requests [109]. The billing scheme of Azure functions is similar to AWS except that billing is done per GB-s of average memory consumed during an execution instead of the memory allocated. Google functions charge users for both the GB-s of memory provisioned and GHz-s of CPU provisioned.

Decisions on which functions of an application are suitable to be fused (combined) and which are best placed in the cloud or on the edge, are affected by the billing model [46]. Also, under a public/private hybrid cloud serverless model, function placement decisions in the private and the public cloud are based on the serverless billing model in the public cloud [38]. Gunasekaran et al. present a framework to use serverless functions and VM based executions interchangeably for better cost efficiency [58]. Instead of the static pricing schemes of the existing platforms, schedulers could also utilize dynamic pricing schemes which enable users requiring higher QoS levels (e.g., faster response time for applications with high delay sensitivity) to pay and acquire better services [60].

**3.2.4 Runtime Resource Limitations.** Serverless platforms impose various limitations on the allowed resource configurations for applications deployed on them. This is primarily aimed at improving flexibility in managing the available resources among multiple users without locking in a set of resources with a single user or application. These restrictions also prevent serverless systems in the public cloud from being subject to denial-of-service (DoS) attacks by malicious function requests flooding the resources attempting to overload the system and preventing legitimate requests from being fulfilled. On commercial serverless platforms, these limitations are primarily in the allowed memory, local disk space and cpu resource configurations, maximum allowed time for a function execution and the maximum number of parallel executions for a function without compromising on latency. AWS Lambda allows users to select the amount of memory available to a function during execution, and allocates CPU power linearly in proportion to the configured memory. Concurrency is also allowed to be configured for each function with different options available which could be explored based on the workload [109]. AWS Lambda provides a non-persistent local disk of a standard size for all the functions [131] and imposes a maximum function timeout duration. Azure functions on the other hand introduces different hosting plans for function apps such as the consumption, premium and dedicated plan, based on which the resource and time out limitations

will be determined [9]. Google Cloud Functions too, limits the maximum allowable memory for a function and imposes a cap on function duration [50]. Imposing limitations as discussed here, have detrimental effects for some application domains such as long running, compute intensive, data-driven applications [62]. For example, for a data analytics application, an automatic timeout would mean losing any data cached on the running instances, and the addition of a significant latency for retrieval of the same from a slow external storage. Limitations on the concurrent function executions could also affect adaptability in a data processing scenario requiring a massive number of parallel computations.

**3.2.5 Hardware Heterogeneity.** With the continuous expansion of the use of the serverless paradigm in different domains, there is also an escalation of varying demands in order to fulfill SLAs for the applications. Computing platforms with heterogeneity in the hardware layer are being increasingly sought after, especially for workloads requiring High Performance Computing (HPC) [92], [96]. Specifically, compute-intensive workloads such as deep learning, complex data analytics, blockchain, genetics, require support from accelerated hardware components, which even the latest generation of CPUs are unable to provide. Offerings from IaaS providers today have evolved to address this emerging need, by providing instances with access to accelerators such as Graphical Processing Units (GPUs), Tensor Processing Unit (TPUs), Field Programmable Gate Arrays (FPGAs) and even access to quantum computers. Inferentia and Titanium chips from AWS [112], [114] to train ML models, Amazon Elastic Inference service [111] to attach GPUs to VM instances, GPU-enabled nodes on Azure Kubernetes Service (AKS) [12], FPGA support in Azure [11], Google's cloud and edge TPU [32], [33] are a few examples. In addition, quantum-based processing elements have rapidly emerged in recent times and cloud service providers have started to offer such quantum processing capabilities as part of their cloud services as well [110]. However, none of the serverless platforms from these providers offer access to hardware accelerators at the moment [71]. As a result, research focus has been increasingly directed towards expanding the FaaS service offerings by adding access to hardware accelerators. Naranjo et al. introduce a GPU enabled serverless framework which links virtual GPUs with the OpenFaaS serverless framework via the rCUDA [42] remote GPU virtualization service [91]. Ringlein et al. propose a system architecture involving disaggregated FPGAs within a FaaS offering [102]. A distributed FPGA sharing system which realizes multi-tenancy for FPGAs in a cloud serverless environment is presented in the works of Bacis et al. [14]. For certain application scenarios, serving all the queries using expensive hardware accelerators may not be economically viable (e.g., ML inference queries). For such scenarios, hybrid approaches to opportunistically serve the incoming requests using a mix of GPU-enabled instances and traditional CPU-only instances, could be explored [138], [103]. Research efforts are also directed towards highlighting the adaptability of quantum computing in the design of serverless systems [54], [55].

### 3.3 Workload Management

Since the serverless provider is responsible for the autonomic management of resources for applications, it is imperative that the platform develops an understanding of the nature, requirements and behavior of the incoming workloads. Three aspects in which awareness of the incoming workload is important for making efficient resource management decisions, are discussed below.

**3.3.1 Application Model.** Application model is the nature of the scheduling unit of an application that is deployed on a serverless platform. An application could have a certain structure and also QoS requirements that are independently specified for each task or for the application unit as a whole.

- *Structure*: An application could be defined by a single task or a set of tasks compiled in the form of a Directed Acyclic Graph (DAG). A task would be denoted by a single function. Thus in a DAG, each node is a function representing a fine-grained task and each edge represents a dependency among two functions [26], [120], [74], [38]. A developer would specify the QoS requirements for a DAG based application either as a whole [120] or for each individual task. A serverless application could also be monolithic, composed of a single function, representing a single task [63], [8], [125], [86], [88], [69].

- *Scheduling Granularity*: Scheduling granularity refers to the way in which scheduling algorithms handle the execution of an application submitted by a user. When an application takes the form of a DAG, it could either be scheduled as a single unit or each individual function could be scheduled separately. In addition, the scheduler could decide to queue requests and schedule them in batches [141]. The scheduling granularity could be specified as a requirement of the developer or decided by a serverless platform, aiming for better resource efficiency.

AWS schedules each function independently, irrespective of whether it is part of a workflow or a single function application. Accordingly, a user request calling the first function in a workflow application and a request calling one function from another during a workflow execution, are treated in a similar manner. Both requests go through the same scheduling policies [109]. Azure Cloud Functions uses the concept of "function app", as the unit of scheduling and management of a serverless application. A function app is comprised of one or more functions with dependencies [9]. Studies suggest that orchestrating functions of the same application on an individual basis at a global system level could incur extra latency. Akkus et. al suggest a serverless architecture utilizing a hierarchical message queuing mechanism with a local message bus on each host which handles local interactions among functions of the same application and their orchestration [3]. Developers could also find the optimal way to fuse or combine a number of functions in a DAG for scheduling purposes, so as to reduce execution cost and latency associated with state transitions and network delays. This would be applicable to serverless deployments in the edge/fog environments as well due to enhanced delays over the network caused by data transfers across functions and associated costs [46]. When requests are less latency sensitive, the scheduler could decide to queue and schedule requests in batches or as a collection, for better resource efficiency.

**3.3.2 Workload Nature.** Serverless platforms are multi-tenant infrastructures and thus applications of multiple users compete for resources in a common shared environment. This could cause a lot of contention on the platform and lead to poor application performance if the resources are not properly managed as per the requirements of different applications.

By nature any generic application could be either CPU, memory or I/O intensive. In some commercial serverless platforms, function placement on a VM is treated as a bin packing problem to maximize memory utilization [131]. While this helps maximize resource utilization, it could also lead to CPU contentions if strict resource isolation strategies among applications are not adopted. Having an understanding on the rate at which requests arrive for applications too is an important factor.

The nature of the workload could be adopted in resource management decisions either when scheduling functions on VMs initially or in the subsequent management of limited resources. Mahmoudi et al. use a machine learning based approach to develop a predictive performance model which tries to predict the normalized performance of any workload when assigned to a specific VM[86]. Each time a new function is being deployed on a platform, a profiling step identifies the memory and CPU utilization levels of the function as well as the dependence on I/O features of the platform. Then predictive models are used to identify the VM that gives the best system performance for the function at that time slot. A good understanding of the nature of the

applications is also important when the execution happens in an edge/cloud environment. Compute intensive functions are better served at the cloud which has seemingly unlimited resources even with a higher latency due to network delays. Lightweight functions may be better served at the edge with a lower response time. The placement decision of functions on the edge/cloud devices requires a thorough understanding of the application behavior in each of the environments [46], [97].

If a serverless platform does not impose hard upper limits on the level of resources that a particular function instance could utilize, the co-location of many such functions having the same resource needs, could lead to deterioration of performance over time. Under such conditions, based on the state of the system and the resource needs of each application, dynamic management of applied resource limits to containers is an approach worth exploring more [88], [127]. Further, we could use Linux Kernel's cgroups to control and isolate resource usage of a collection of processes based on the requirements [129], [74]. These techniques could be adopted effectively with a better understanding on the needs of different workloads. Further, in addition to the resource requirements, modelling arrival rate of requests for different functions help in taking proactive resource scaling decisions [120].

*3.3.3 Data Locality and State Management.* Data locality generally refers to the movement of computation to the nodes which contain the data required for the task execution. For data intensive applications this could improve the makespan drastically due to reduced network delays. Serverless platforms are known to decouple data management from function execution. Thus serverless is said to be rather a data-shipping architecture (ships data to code) instead of shipping code to data [62]. This is a defining property of the serverless paradigm since the dis-aggregation of these two aspects allows serverless functions to scale in an independent manner [52]. For example, ensuring locality among serverless functions may mean executing functions which share data, in the same node or VM instance. While this would improve performance with fast shared memory, this could also reduce the flexibility of the provider to schedule functions and scale capacity [52]. Therefore scheduling and resource management techniques under the serverless model traditionally have not incorporated data-locality awareness in making their decisions. However, a significant repercussion of this dis-aggregated model is the challenge of managing the intermediary state for stateful applications. Generally applications are required to be executed as stateless functions in commercial serverless platforms, and they share state through external storage services (e.g., Amazon S3) [68], [62]. As such, for latency and bandwidth sensitive data-intensive applications such as machine learning, delays in network and storage layers make this a less than an ideal environment. Today, this realization has motivated research into the development of serverless frameworks that support stateful executions. Azure Durable Functions allow to write and orchestrate stateful workflows [22]. AWS Step Functions [109] and IBM composer [34] support the creation and coordination of state transitions for complex workflows. Shillaker et al. introduce a new abstraction for function isolation, which allows memory to be shared between functions in the same address space [118]. They propose a two-tier architecture, where the local tier allows sharing of state among functions in the same host and the global tier supports state sharing across machines. A data transfer and state management approach is presented in [31] for a stateful function chain in an edge network. Jia et al. present a new serverless runtime which exposes an API to a shared log service which enables serverless functions to manage their state efficiently [65]. This mechanism of shared logs is able to achieve state management with high durability, fault tolerance and consistency.

In addition to the above, many studies focus on achieving data locality via load balancing methods to direct new function requests to workers with pro-actively spawned containers or containers that could be re-used [120], [82]. Research works have also focused on attaining data locality among

functions in terms of the runtime package dependencies of functions, by routing functions requiring similar packages to the same node for execution [8]. Lee et al. propose a greedy load balancing algorithm which tries to maximize locality and improve the cache-hit ratio while also minimizing load imbalance among nodes [79]. The new sandboxing method proposed by Akkus et al. [3] where functions of the same application share the same container, is able to improve latency by increased data locality, since the libraries shared by all the functions are needed to be loaded from memory only once.

### 3.4 QoS Goal

The resource allocation, scheduling and scaling techniques are aimed at satisfying certain requirements upon the execution of applications. These could be constraints that applications need to satisfy or an optimization goal that determines the performance of the resource management techniques.

**3.4.1 Latency.** Latency refers to the delay between a user request submission and the serverless provider's response. The request queuing time, resource setup time and the function execution time collectively result in the response time for a serverless application. The ability to maintain low latency for function executions is a key concern in serverless environments, specially since a majority of individual functions have execution times less than a second, or of a few seconds [120]. One main cause for high application latency in serverless environments is the cold start delay in resource setup, which tends to become significant compared to application execution times [88]. Scheduling algorithms along with different container pool management techniques, customized sandboxing methods, and low-overhead runtimes to reduce resource setup times are explored in literature to reduce cold start delay [66], [8], [120], [82], [53], [3], [93]. Latency caused by CPU contention is also studied and dynamic resource control methods are proposed for serverless applications [88], [127], [74], [63].

**3.4.2 Throughput.** Throughput refers to the number of function requests processed by a serverless system in unit time. This is a good metric to demonstrate efficiency of the overall system and the ability to support high request arrival rates [86]. Throughput and mean latency of a system usually show a negative correlation. Thus techniques to reduce latency also improve throughput of a system. The allowed concurrency level for function instances in serverless platforms is also a determining factor for system throughput [108].

**3.4.3 Fault Tolerance.** Fault tolerance in cloud computing refers to the ability of a system to function uninterrupted, at the sight of failure in one or more of its components. Serverless platforms generally achieve fault tolerance by implementing retry based approaches. For example, a platform would automatically resend the request in case of failure of a function execution. Sreekanti et al. show that under certain circumstances, a retry based approach to fault tolerance may always not give accurate results, specially where state management is involved via external storage services [124]. They emphasize that the accuracy may be affected in case a parallel execution of a function, views results of a partially executed failed attempt of the same execution. A low-overhead fault tolerance scheme is proposed to guarantee read atomic isolation, which prevents parallel executions of functions from viewing partial outputs from any failed attempts. Zhang et al. introduce a runtime system and a library, to help developers to write stateful and fault tolerant serverless workflows [139]. Their approach encompasses a log-based request re-execution approach to achieve fault tolerance. The developed framework could be adapted to work with existing cloud providers in a federated environment.

**3.4.4 User Cost.** Function execution cost to the user is as per the billing model of serverless platforms as described in the previous section. Since billing is mostly correlated with the function execution times, efforts to reduce execution latency result in optimized cost to the user. But for compute intensive applications, response time and cost could be inversely related since the allocated CPU time would affect the response time and thus the execution cost. Experiments are done to observe the response time of functions for different resource allocations and associated costs. Such models could help a user decide on a suitable resource allocation scheme when there is a budget constraint, and also the cost to achieve a certain performance level [81], [60]. The scheduling granularity in terms of function fusion in serverless workflows could have an impact on the overall costs as well due to associated state transition costs and effects on function response times [46].

**3.4.5 Resource Efficiency.** Resource efficiency refers to maintaining a high utilization level for the underlying active resources of the provider at all times. The fine-grained serverless billing model implies that the user is charged only for the resource-time actually consumed by the application during its execution. Regardless of this, the provider has to maintain the overall infrastructure and as such, consolidating as many serverless applications as possible into a single host is in provider's best interest in order to yield better profits. On the other hand, packing too many requests on a single resource subsequently leads to poor performance. This reflects the typical conflicting objectives of the providers and consumers: minimization of cost and maximization of performance [74]. Thus finding an optimal resource consolidation strategy to the satisfaction of both parties is important. HoseinyFarahabady et al. propose a QoS aware resource allocation controller for serverless environments which tries to minimize QoS violations to users while maintaining a healthy CPU utilization level of 60%-80% in each host in order to reach an ideal balance between system performance and energy consumption [63]. Somma et al propose a function container auto-scaling technique to optimize throughput and resource efficiency [123]. Resource efficiency is also important in serverless platforms deployed on edge resources as well due to limited capacities at the edge.

**3.4.6 Energy Efficiency.** Recently, attention has also been given to the aspect of energy efficiency of running serverless systems, which is also connected to resource efficiency. Early surveys identify serverless computing to be promoting green computing due to the on-demand creation and release of resources used for function execution. Moreover, the model of billing per execution time incentivizes programmers to improve resource usage and execution time of their code [115]. On the other hand, breaking down an application into a set of functions and the practice of setting up resources on-demand is deemed to cause additional latency and an execution overhead, which affect function performance. Kansil et al. define a measure for energy efficiency, based on the mean time between two calls to the same function, mean time to setup resources and the mean time to execute the function [70]. They show that the resource saving in serverless systems is generally highest when function calls are irregular and have large time gaps in between, compared to the resource setup and function execution times. Deriving from this idea, Poth et al. present a further developed model, capturing the overheads of the components which manage the serverless system during function invocation and execution, which is aimed at helping design decisions of serverless applications and systems [98]. Gunasekaran et al. present a resource management framework to efficiently manage function chains on a serverless platform by improving container utilization and cluster wide energy consumption [59].

**3.4.7 Security.** In addition to the security challenges that are common to any computing environment, serverless systems are susceptible to certain threats that are unique to these environments. Shafiei et al. identify application-level and function-level authentication to be one such specific

challenge [115]. Application-level authentication refers to a mechanism which determines which users are allowed to access a certain application, while function-level authentication refers to the allowed invocations of one function from another. They also highlight the possible chance of replay attacks, where an intruder would capture a function execution request and execute it repeatedly, constraining the system resources and blocking legitimate users from accessing the service. In general, runtime limitations are in place, in terms of limited execution times and maximum allowed CPU and memory allocations for a function, in order to minimize the effect of such attacks. Function isolation mechanism too plays an important role in enabling data privacy among tenants on the same host. Container namespaces are commonly used by current serverless systems for providing isolation among functions [4].

#### **4 CLASSIFICATION OF RESOURCE MANAGEMENT TECHNIQUES USING TAXONOMY**

In Table 1, we review existing key works on serverless resource management, that identify most with the proposed taxonomy. In essence, we present here, the works which explore novel techniques for one or more of the primary aspects of resource management that we have identified.

Work	Deployment Environment			Workload Management				QoS Goal	Resource Management Technique		
	Deployment Model	Application Isolation	Pricing Model Awareness	Application Model		Workload Nature Awareness	Data Locality Awareness		Workload Modelling	Resource Scheduling	Resource Scaling
				Structure	Scheduling Granularity						
[63]	Cloud	Kernel thread	-	Task	Single Function	✓	-	Latency, Resource Efficiency	-	Feedback control system, Heuristic	-
[46]	Edge/Cloud	Container	✓	DAG	Function Fusion	✓	✓	Latency, User Cost	-	LARAC	-
[97]	Edge/Cloud	Container	-	Task	Single Function	✓	-	Latency	-	Greedy, UCB1, Bayesian UCB	-
[126]	Cloud	Container	-	Task	Single Function	✓	-	Latency, Resource Efficiency	-	Non-cooperative game theoretic heuristic	Heuristic
[8]	Cloud	Optimized container	-	Task	Single Function	-	✓	Latency	-	Greedy	-
[21]	Cloud	Container	✓	Task	Single Function	✓	✓	Latency, User Cost	Mathematical Modelling, Heuristic	Machine Learning	Machine Learning
[58]	VM/FaaS based	Container	✓	Task	Single Function	-	-	Latency, User Cost	Mathematical Modelling	Greedy	Heuristic
[69]	Cloud	Container	-	Task	Single Function	-	-	Latency, User Cost	-	Heuristic	-
[82]	Cloud	Container	-	Task	Single Function	✓	✓	Latency, Throughput	-	Greedy	-
[86]	Cloud	Container	-	Task	Single Function	✓	-	Latency, Throughput	ML Modelling	Heuristic	Heuristic
[2]	Cloud	-	✓	DAG	Single Function	✓	-	Latency, User Cost	Mathematical Modelling	-	-
[19]	Edge/Cloud	-	✓	Task	Single Function	-	✓	Resource Efficiency	-	Heuristic	-
[36]	Cloud	Container/MicroVM	✓	Task	Single Function	✓	-	Latency, User Cost	ML Modelling	-	-
[37]	Edge/Cloud	Container	✓	Task	Single Function	✓	-	Latency, User Cost	Mathematical Modelling	Heuristic	-
[38]	Cloud	Container	✓	DAG	Single Function	✓	-	Latency, User Cost	Mathematical Modelling	Greedy	-
[43]	Cloud	-	✓	DAG	Single Function	✓	-	Latency, User Cost	ML Modelling, Mathematical Modelling	-	-

Work	Deployment Environment			Workload Management				QoS Goal	Resource Management Technique		
	Deployment Model	Application Isolation	Pricing Model Awareness	Application Model		Workload Nature Awareness	Data Locality Awareness		Workload Modelling	Resource Scheduling	Resource Scaling
				Structure	Scheduling Granularity						
[59]	Cloud	Container	-	DAG	Single Function	✓	✓	Latency, Resource Efficiency, Throughput	ML Modelling, Mathematical Modelling	Heuristic	Heuristic
[60]	Cloud	-	✓	DAG	Single Function	-	-	Latency	-	MILP	-
[74]	Cloud	Process	-	Task/DAG	Single Function	-	-	Latency, Resource Efficiency	Feedback control system	Feedback control system, Heuristic	-
[81]	Cloud	-	✓	DAG	Single Function	✓	-	Latency, User Cost	Mathematical Modelling	-	-
[80]	VM/FaaS based	Container	-	Task	Single Function	✓	-	Latency, Resource Efficiency	Mathematical Modelling	Heuristic	Heuristic
[84]	Cloud	-	✓	Task	Single Function	✓	-	Latency, User Cost	Mathematical Modelling	Heuristic	Heuristic
[98]	Cloud	Container	-	Task	Single Function	✓	-	Energy Efficiency	Mathematical Modelling	-	-
[121]	Cloud	Container	-	Task	Single Function	-	✓	Latency	-	Heuristic	Heuristic
[123]	Cloud	Container	-	Task	Single Function	-	-	Latency, Resource Efficiency, Throughput	-	Heuristic	RL
[127]	Cloud	Container	-	Task	Single Function	✓	-	Latency, Resource Efficiency	-	Greedy	Heuristic
[128]	Cloud	Container	-	Task/DAG	Function Fusion, Single Function	-	-	Throughput, Resource Efficiency	-	Heuristic	-
[141]	Edge/Cloud	Container	-	Task	Collection	✓	-	Latency	Mathematical Modelling	Heuristic	-
[20]	Cloud	Container	-	DAG	Single Function	✓	-	Latency, Resource Efficiency	Mathematical Modelling	Heuristic	Heuristic
[79]	Cloud	Container	-	Task	Single Function	-	✓	Latency	-	Greedy	Heuristic
[88]	Cloud	Container	-	Task	Single Function	✓	-	Resource Efficiency, Latency	-	Greedy	Heuristic
[100]	Edge/Cloud	Container	✓	Task	Single Function	✓	✓	Latency, User Cost	-	Greedy	-

Work	Deployment Environment			Workload Management				QoS Goal	Resource Management Technique		
	Deployment Model	Application Isolation	Pricing Model Awareness	Application Model		Workload Nature Awareness	Data Locality Awareness		Workload Modelling	Resource Scheduling	Resource Scaling
				Structure	Scheduling Granularity						
[108]	Cloud	Container	-	Task	Single Function	-	-	Latency, Throughput	-	Heuristic	RL
[120]	Cloud	Container	-	DAG	Function Fusion, Single Function	✓	✓	Latency	Mathematical Modelling	Heuristic	Heuristic
[134]	Cloud	Container	-	Task	Single Function	-	-	Latency	-	DRL	Heuristic

Table 1. Classification of Resource Management Techniques

## 5 GAP ANALYSIS AND FUTURE DIRECTIONS

This detailed review on resource management under the emerging serverless computing model on edge, fog and cloud resources highlights open problems with great potential for exploration in future work. We discuss these areas in detail along the broader categories we have identified in this article, laying the groundwork for both research and development work in the coming years.

### 5.1 System Design Characteristics

Serverless model has been explored under different deployment models with hybrid infrastructures with geographical as well as performance distribution. These include a mix of edge, fog and cloud infrastructures along both private and public domains. One notable challenge for developers in adapting to the serverless model is the concern over the vendor lock-in effect due to the unique function signatures, naming conventions and other compatibilities required by each provider. Thus today, interest is building up on studies on multi-provider serverless support with reduced provider lock-in for applications [7]. This is in line with the concept of adopting multi-cloud infrastructure solutions by organizations looking out for the most cost effective model. Additionally, there is an increased interest in analyzing hybrid deployment models realizing benefits from both serverless and traditional serverful models [38], [58], [97]. Further research into the most efficient modes of deployment for serverless applications will have great future prospects.

In the current commercial serverless platforms users are not allowed to specify the runtime hardware environment for their function executions. In terms of processing capabilities, they only offer access to CPU processing for the deployed applications. With increased exploitation of serverless for different application domains, there has been a rise in demand for access to specialized hardware where required. Research on such flexible serverless frameworks is growing [91] and has great potential.

The serverless model is built on the premise that the provider is at liberty to control the underlying infrastructure as they wish. This opens up interesting opportunities for them to seek maximum resource efficiency by leveraging idling resources from other running services as well as less attractive machines not conducive for leasing out to IaaS customers [68]. Methods of consolidating and coordinating such resources for function deployments is a novel idea open for exploration.

To date the billing models in existing serverless platforms offer only fixed pricing schemes to all consumers. Customers having different requirements may be willing to pay more or less for enhanced or standard levels of service, calling for the need for dynamic pricing strategies [60]. A generalized level of service may not be sufficient for time-sensitive critical applications requiring a high level of performance such as in the medical domain. Further research into such customized pricing models would enhance the meaning of the term "Function-as-a-service", luring in more diversified user pools.

### 5.2 Workload Management and QoS Goals

Serverless platforms still face many challenges when being explored for some applications with complex data flow dependencies [20]. Lack of an efficient function to function communication mechanism complicates the process of orchestrating complex workflow structures. Hierarchical message queuing and brokering mechanisms [115] for handling local communication requirements among functions of the same application are interesting propositions. On the other hand, these solutions also have the downside of reducing extreme flexibility features offered by the whole serverless architecture. Nevertheless, most of the real-world applications would generally have complex workflow structures. As such, exploring solid mechanisms for orchestrating such applications creates a new research dimension of significant importance [6].

Further research could also be extended into providing more flexible QoS offerings to users, accompanied by suitable pricing models. Allowing too many specific demands from users could lead to poor resource efficiency for the provider and as such a compromise needs to be arrived at, after thorough investigation. Many resource scheduling models proposed for the serverless model focus on meeting the latency and budget constraints of consumers and rarely on the efficient resource usage at the provider. Maintenance of large resource pools for mitigating application latency deterioration would have very low attractiveness to a provider if it leads to a large resource wastage. Thus, critically analyzing the QoS requirements from a service provider's point of view needs to be a key research goal.

Modelling the aspect of energy efficiency in serverless environments in the cloud and more importantly in edge environments is a new avenue being explored from recent times, with a great potential for future work [98].

### 5.3 Resource Management Techniques

Increased level of abstraction in specifying resource usage by developers leaves the serverless provider with the need to infer resource requirements and code dependency requirements in making appropriate resource allocation and scheduling decisions. Serverless platforms cater to needs of diverse applications with vastly differing characteristics and requirements. Understanding and profiling of workloads as per their resource needs and arrival patterns could help in predicting their future behavior. This in turn would largely assist resource scheduling and scaling decisions that could benefit both the consumers and providers. Research focus has already been directed towards modelling application performance and cost in serverless environments deployed in the cloud and at the edge, with different resource configurations for applications [37], [81]. Although these models help in taking static scheduling decisions such as deciding on initial resource configurations and placement decisions, most of these approaches do not address dynamic factors that would affect performance in the runtime. Applications may not perform as required due to co-located application interference, machine performance variations and degradation due to over-utilization. Moreover, dynamic variations in workload patterns need to be taken into account when taking these high impact decisions. As such, employing intelligent methods for capturing the nature of these dynamic environments is of high relevance. Recent works using reinforcement learning, for identifying and rectifying shortcomings in resource allocation decisions, pave the path for a new research direction [134].

Although initial efforts are made in extending serverless model across the edge and fog computing networks, much further investigation is required in this area for taking effective resource management decisions. Given the involved delays across networks, transporting data and code to the edge for individual function executions as per the cloud serverless model would be ineffective specially for latency sensitive applications. Thus there is need for intermediary storage or caching services. Limitations caused by heterogeneity of edge and fog devices, on handling compute intensive function executions need to be considered in scheduling and scaling decisions. Data locality concerns inherent with the serverless model may be even more relevant for geographically distributed edge/fog computing networks.

Cold starts arising from the unique auto-scaling capability is a growing research area. Numerous solutions ranging from proactive resource scaling to reduce coldstart frequencies, to designing optimized sandbox solutions are proposed. A further interesting approach in this regard is to incorporate intelligent techniques in deciding optimum function concurrency levels and methods for proactive resource provisioning to further diminish this challenge [108].

## 6 SUMMARY

In this paper we have presented a comprehensive review on the aspect of resource management, referring to unique characteristics of this new serverless computing model. We propose a taxonomy covering the broader concept of resource management in edge, fog and cloud infrastructures, along the categories of system design decisions, approaches for incoming workload identification and management, and the QoS goals of involved parties. We also discuss three key aspects of resource management techniques and analyse the existing works using the proposed taxonomy. This taxonomy presents a clear view for serverless system designers on the essential features for consideration for a fully-fledged effective system. Further this provides a learning platform for researchers studying resource allocation, scheduling and scaling techniques in the serverless domain, based on which their work could demonstrate progress in the field. Finally, we provide a gap analysis referring to identified and addressed challenges, emphasizing on the vast potential for further research work.

## REFERENCES

- [1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight virtualization for serverless applications. In *Proceedings of the 17th USENIX symposium on networked systems design and implementation ({nsdi} 20)*. 419–434.
- [2] Nabeel Akhtar, Ali Raza, Vatche Ishakian, and Ibrahim Matta. 2020. COSE: Configuring Serverless Functions using Statistical Learning. In *Proceedings of the INFOCOM - IEEE Conference on Computer Communications*. IEEE, 129–138.
- [3] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. 2018. SAND: Towards High-Performance Serverless Computing. In *Proceedings of the USENIX Annual Technical Conference*. 923–935.
- [4] Zaid Al-Ali, Sepideh Goodarzy, Ethan Hunter, Sangtae Ha, Richard Han, Eric Keller, and Eric Rozner. 2018. Making serverless computing more serverless. In *Proceedings of the 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 456–459.
- [5] Ahsan Ali, Riccardo Pincioli, Feng Yan, and Evgenia Smirni. 2020. Batch: machine learning inference serving on serverless platforms with adaptive batching. In *Proceedings of SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 972–986.
- [6] Aitor Arjona, Pedro García López, Josep Sampé, Aleksander Slominski, and Lionel Villard. 2021. Triggerflow: Trigger-based orchestration of serverless workflows. *Future Generation Computer Systems* (2021).
- [7] Austin Aske and Xinghui Zhao. 2018. Supporting multi-provider serverless computing on the edge. In *Proceedings of the 47th International Conference on Parallel Processing Companion*. 1–6.
- [8] Gabriel Aumala, Edwin Boza, Luis Ortiz-Avilés, Gustavo Totoy, and Cristina Abad. 2019. Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms. In *Proceedings of the 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 282–291.
- [9] Microsoft Azure. 2021. Azure Functions documentation | Microsoft Docs. <https://docs.microsoft.com/en-us/azure/azure-functions/>. (Accessed on 01/01/2021).
- [10] Microsoft Azure. 2021. Azure IoT Edge documentation | Microsoft Docs. <https://docs.microsoft.com/en-us/azure/iot-edge/?view=iotedge-2018-06>. (Accessed on 01/14/2021).
- [11] Microsoft Azure. 2021. Deploy ML models to FPGAs - Azure Machine Learning | Microsoft Docs. <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-fpga-web-service>. (Accessed on 11/17/2021).
- [12] Microsoft Azure. 2021. Use GPUs on Azure Kubernetes Service (AKS) - Azure Kubernetes Service | Microsoft Docs. <https://docs.microsoft.com/en-us/azure/aks/gpu-cluster>. (Accessed on 11/17/2021).
- [13] Ataollah Fatahi Baarzi, George Kesidis, Carlee Joe-Wong, and Mohammad Shahradd. 2021. On Merits and Viability of Multi-Cloud Serverless. In *Proceedings of the ACM Symposium on Cloud Computing*. 600–608.
- [14] Marco Bacis, Rolando Brondolin, and Marco D Santambrogio. 2020. BlastFunction: an FPGA-as-a-Service system for Accelerated Serverless Computing. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 852–857.
- [15] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. 2017. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*. Springer, 1–20.
- [16] Luciano Baresi and Danilo Filgueira Mendonça. 2019. Towards a serverless platform for edge computing. In *Proceedings of the IEEE International Conference on Fog Computing (ICFC)*. IEEE, 1–10.

- [17] Luciano Baresi, Danilo Filgueira Mendonça, and Martin Garriga. 2017. Empowering low-latency applications through a serverless edge computing architecture. In *Proceedings of the European Conference on Service-Oriented and Cloud Computing*. Springer, 196–210.
- [18] David Bermbach, Ahmet-Serdar Karakaya, and Simon Buchholz. 2020. Using application knowledge to reduce cold starts in FaaS services. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 134–143.
- [19] David Bermbach, Setareh Maghsudi, Jonathan Hasenburg, and Tobias Pfandzelter. 2020. Towards auction-based function placement in serverless fog platforms. In *Proceedings of the IEEE International Conference on Fog Computing (ICFC)*. IEEE, 25–31.
- [20] Vivek M Bhasi, Jashwant Raj Gunasekaran, Prashanth Thinakaran, Cyan Subhra Mishra, Mahmut Taylan Kandemir, and Chita Das. 2021. Kraken: Adaptive Container Provisioning for Deploying Dynamic DAGs in Serverless Platforms. In *Proceedings of the ACM Symposium on Cloud Computing*. 153–167.
- [21] Anirban Bhattacharjee, Ajay Dev Chhokra, Zhuangwei Kang, Hongyang Sun, Aniruddha Gokhale, and Gabor Karsai. 2019. Barista: Efficient and scalable serverless serving system for deep learning prediction services. In *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 23–33.
- [22] Sebastian Burckhardt, Chris Gillum, David Justo, Konstantinos Kallas, Connor McMahon, and Christopher S Meiklejohn. 2021. Durable functions: semantics for stateful serverless. *Proceedings of the ACM on Programming Languages* 5, OOPSLA (2021), 1–27.
- [23] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).
- [24] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2018. A case for serverless machine learning. In *Proceedings of the Workshop on Systems for ML and Open Source Software at NeurIPS*, Vol. 2018.
- [25] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2019. Cirrus: A serverless framework for end-to-end ml workflows. In *Proceedings of the ACM Symposium on Cloud Computing*. 13–24.
- [26] Benjamin Carver, Jingyuan Zhang, Ao Wang, and Yue Cheng. 2019. In search of a fast and efficient serverless dag engine. In *Proceedings of the IEEE/ACM 4th International Parallel Data Systems Workshop (PDSW)*. IEEE, 1–10.
- [27] Gustavo André Setti Cassel, Vinicius Facco Rodrigues, Rodrigo da Rosa Righi, Marta Rosecler Bez, Addressa Cruz Nepomuceno, and Cristiano André da Costa. 2021. Serverless computing for Internet of Things: A systematic literature review. *Future Generation Computer Systems* (2021).
- [28] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. The rise of serverless computing. *Commun. ACM* 62, 12 (2019), 44–54.
- [29] Bin Cheng, Jonathan Fuerst, Gurkan Solmaz, and Takuya Sanada. 2019. Fog function: Serverless fog computing for data intensive iot services. In *Proceedings of the IEEE International Conference on Services Computing (SCC)*. IEEE, 28–35.
- [30] Michele Ciavotta, Davide Motterlini, Marco Savi, and Alessandro Tundo. [n.d.]. DFaaS: Decentralized Function-as-a-Service for Federated Edge Computing. ([n. d.]).
- [31] Claudio Cicconetti, Marco Conti, and Andrea Passarella. 2021. On Realizing Stateful FaaS in Serverless Edge Networks: State Propagation. In *Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 89–96.
- [32] Google Cloud. 2021. Cloud Tpu | Cloud TPU | Google Cloud. <https://cloud.google.com/tpu>. (Accessed on 11/17/2021).
- [33] Google Cloud. 2021. Edge TPU - Run Inference at the Edge | Google Cloud. <https://cloud.google.com/edge-tpu>. (Accessed on 11/17/2021).
- [34] IBM Cloud. 2021. IBM Cloud Docs. [https://cloud.ibm.com/docs/openwhisk?topic=openwhisk-pkg\\_composer](https://cloud.ibm.com/docs/openwhisk?topic=openwhisk-pkg_composer). (Accessed on 11/19/2021).
- [35] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. 2021. Sebs: A serverless benchmark suite for function-as-a-service computing. In *Proceedings of the 22nd ACM International Middleware Conference*.
- [36] Robert Cordingly, Wen Shu, and Wes J Lloyd. 2020. Predicting Performance and Cost of Serverless Computing Functions with SAAF. In *Proceedings of the IEEE International Conference on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, 640–649.
- [37] Anirban Das, Shigeru Imai, Stacy Patterson, and Mike P Wittie. 2020. Performance Optimization for Edge-Cloud Serverless Platforms via Dynamic Task Placement. In *Proceedings of the 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 41–50.
- [38] Anirban Das, Andrew Leaf, Carlos A Varela, and Stacy Patterson. 2020. Skedulix: Hybrid Cloud Scheduling for Cost-Efficient Execution of Serverless Applications. In *Proceedings of the 13th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 609–618.

- [39] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. 2020. Xanadu: Mitigating cascading cold starts in serverless function chain deployments. In *Proceedings of the 21st International Middleware Conference*. 356–370.
- [40] Giuseppe De Palma, Saverio Giallorenzo, Jacopo Mauro, and Gianluigi Zavattaro. 2020. Allocation Priority Policies for Serverless Function-Execution Scheduling Optimisation. In *Proceedings of the International Conference on Service-Oriented Computing*. Springer, 416–430.
- [41] Docker. 2021. Runtime options with Memory, CPUs, and GPUs | Docker Documentation. [https://docs.docker.com/config/containers/resource\\_constraints/](https://docs.docker.com/config/containers/resource_constraints/). (Accessed on 01/04/2021).
- [42] José Duato, Antonio J Pena, Federico Silla, Rafael Mayo, and Enrique S Quintana-Orti. 2010. rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In *Proceedings of the International Conference on High Performance Computing & Simulation*. IEEE, 224–231.
- [43] Simon Eismann, Johannes Grohmann, Erwin Van Eyk, Nikolas Herbst, and Samuel Kounev. 2020. Predicting the costs of serverless workflows. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 265–276.
- [44] Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina Abad, and Alexandru Iosup. 2021. The State of Serverless Applications: Collection, Characterization, and Community Consensus. *IEEE Transactions on Software Engineering* (2021).
- [45] Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L Abad, and Alexandru Iosup. 2020. A review of serverless use cases and their characteristics. *arXiv preprint arXiv:2008.11110* (2020).
- [46] Tarek Elgamal. 2018. Costless: Optimizing cost of serverless computing through function fusion and placement. In *Proceedings of the IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 300–312.
- [47] Jonatan Enes, Roberto R Expósito, and Juan Touriño. 2020. Real-time resource scaling platform for Big Data workloads on serverless environments. *Future Generation Computer Systems* 105 (2020), 361–379.
- [48] Geoffrey C Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2017. Status of serverless computing and function-as-a-service (faas) in industry and research. *arXiv preprint arXiv:1708.08028* (2017).
- [49] Google Cloud Functions. 2021. Cloud Functions | Google Cloud. <https://cloud.google.com/functions/>. (Accessed on 01/01/2021).
- [50] Google Cloud Functions. 2021. Quotas | Cloud Functions Documentation | Google Cloud. <https://cloud.google.com/functions/quotas>. (Accessed on 03/19/2021).
- [51] Fabian Gand, Ilenia Fronza, Nabil El Ioini, Hamid R Barzegar, and Claus Pahl. 2020. Serverless Container Cluster Management for Lightweight Edge Clouds. In *CLOSER*. 302–311.
- [52] Pedro García-López, Marc Sánchez-Artigas, Simon Shillaker, Peter Pietzuch, David Breitgand, Gil Vernik, Pierre Sutra, Tristan Tarrant, and Ana Juan Ferrer. 2019. ServerMix: Tradeoffs and Challenges of Serverless Data Analytics. *arXiv* (2019), arXiv–1907.
- [53] Alim Ul Gias and Giuliano Casale. 2020. COCOA: Cold Start Aware Capacity Planning for Function-as-a-Service Platforms. In *Proceedings of the 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 1–8.
- [54] Sukhpal Singh Gill. 2021. Quantum and blockchain based Serverless edge computing: A vision, model, new trends and future directions. *Internet Technology Letters* (2021), e275.
- [55] Sukhpal Singh Gill, Adarsh Kumar, Harvinder Singh, Manmeet Singh, Kamalpreet Kaur, Muhammad Usman, and Rajkumar Buyya. 2020. Quantum computing: A taxonomy, systematic review and future directions. *Software: Practice and Experience* (2020).
- [56] Vicent Giménez-Alventosa, Germán Moltó, and Miguel Caballer. 2019. A framework and a performance assessment for serverless MapReduce on AWS Lambda. *Future Generation Computer Systems* 97 (2019), 259–274.
- [57] Martin Grambow, Tobias Pfandzelter, Luk Burchard, Carsten Schubert, Max Zhao, and David Bermbach. 2021. BeFaaS: An Application-Centric Benchmarking Framework for FaaS Platforms. In *Proceedings of the 9th IEEE International Conference on Cloud Engineering*.
- [58] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Mahmut Taylan Kandemir, Bhuvan Uргаonkar, George Kesidis, and Chita Das. 2019. Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud. In *Proceedings of the 12th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 199–208.
- [59] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Nachiappan C Nachiappan, Mahmut Taylan Kandemir, and Chita R Das. 2020. Fifer: Tackling Resource Underutilization in the Serverless Era. In *Proceedings of the 21st International Middleware Conference*. 280–295.
- [60] Vipul Gupta, Soham Phade, Thomas Courtade, and Kannan Ramchandran. 2020. Utility-based Resource Allocation and Pricing for Serverless Computing. *arXiv preprint arXiv:2008.07793* (2020).
- [61] Adam Hall and Umakishore Ramchandran. 2019. An execution model for serverless functions at the edge. In *Proceedings of the International Conference on Internet of Things Design and Implementation*. 225–236.

- [62] Joseph M Hellerstein, Jose Faleiro, Joseph E Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2019. Serverless computing: One step forward, two steps back. In *Proceedings of the Conference on Innovative Data Systems Research*.
- [63] 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 *Proceedings of the International Conference on Service-Oriented Computing*. Springer, 241–255.
- [64] Erika Hunhoff, Shazal Irshad, Vijay Thurimella, Ali Tariq, and Eric Rozner. 2020. Proactive Serverless Function Resource Management. In *Proceedings of the 6th International Workshop on Serverless Computing*. 61–66.
- [65] Zhipeng Jia and Emmett Witchel. 2021. Boki: Stateful Serverless Computing with Shared Logs. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM*. 691–707.
- [66] Zhipeng Jia and Emmett Witchel. 2021. Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 152–166.
- [67] Anshul Jindal, Michael Gerndt, Mohak Chadha, Vladimir Podolskiy, and Pengfei Chen. 2021. Function delivery network: Extending serverless computing for heterogeneous platforms. *Software: Practice and Experience* (2021).
- [68] 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. (2019).
- [69] Kostis Kaffes, Neeraja J Yadwadkar, and Christos Kozyrakis. 2019. Centralized Core-granular Scheduling for Serverless Functions. In *Proceedings of the ACM Symposium on Cloud Computing*. 158–164.
- [70] Ali Kanso and Alaa Youssef. 2017. Serverless: beyond the cloud. In *Proceedings of the 2nd International Workshop on Serverless Computing*. 6–10.
- [71] Anurag Khandelwal, Arun Kejariwal, and Karthikeyan Ramasamy. 2020. Le Taureau: Deconstructing the serverless landscape & A look forward. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2641–2650.
- [72] Jeongchul Kim and Kyungyong Lee. 2019. Functionbench: A suite of workloads for serverless cloud function service. In *Proceedings of the 12th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 502–504.
- [73] Youngbin Kim and Jimmy Lin. 2018. Serverless data analytics with flint. In *Proceedings of the IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 451–455.
- [74] 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.
- [75] Kyriakos Kritikos and Paweł Skrzypek. 2018. A review of serverless frameworks. In *Proceedings of the IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 161–168.
- [76] Kubeless. 2021. Kubeless. <https://kubernetes.io/>. (Accessed on 11/22/2021).
- [77] Jörn Kuhlenskamp, Sebastian Werner, Maria C Borges, Karim El Tal, and Stefan Tai. 2019. An evaluation of faas platforms as a foundation for serverless big data processing. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 1–9.
- [78] Hyungron Lee, Kumar Satyam, and Geoffrey Fox. 2018. Evaluation of production serverless computing environments. In *Proceedings of the 11th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 442–450.
- [79] Youngsoo Lee and Sunghye Choi. [n.d.]. A Greedy Load Balancing Algorithm on Serverless Platforms Maximizing Locality. ([n. d.]).
- [80] Zijun Li, Quan Chen, Shuai Xue, Tao Ma, Yong Yang, Zhuo Song, and Minyi Guo. 2020. Amoeba: QoS-Awareness and Reduced Resource Usage of Microservices with Serverless Computing. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 399–408.
- [81] 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.
- [82] Wei Ling, Lin Ma, Chen Tian, and Ziang Hu. 2019. Pigeon: A Dynamic and Efficient Serverless and FaaS Framework for Private Cloud. In *Proceedings of the International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 1416–1421.
- [83] Jinfeng Liu, Zeyu Mi, Zheng Huang, Zhichao Hua, and Yubin Xia. 2020. Hcloud: A serverless platform for jointcloud computing. In *Proceedings of the IEEE International Conference on Joint Cloud Computing*. IEEE, 86–93.
- [84] Nima Mahmoudi and Hamzeh Khazaei. 2020. Performance Modeling of Serverless Computing Platforms. *IEEE Transactions on Cloud Computing* (2020).
- [85] Nima Mahmoudi and Hamzeh Khazaei. 2021. SimFaaS: A Performance Simulator for Serverless Computing Platforms. *arXiv preprint arXiv:2102.08904* (2021).

- [86] Nima Mahmoudi, Changyuan Lin, Hamzeh Khazaei, and Marin Litoiu. 2019. Optimizing serverless computing: introducing an adaptive function placement algorithm. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*. 203–213.
- [87] Pascal Maissen, Pascal Felber, Peter Kropf, and Valerio Schiavoni. 2020. FaaSdom: A benchmark suite for serverless computing. In *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*. 73–84.
- [88] Anupama Mampage, Shanika Karunasekera, and Rajkumar Buyya. 2021. Deadline-aware Dynamic Resource Management in Serverless Computing Environments. In *Proceedings of the IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 483–492.
- [89] Anup Mohan, Harshad Sane, Kshitij Doshi, Saikrishna Edupuganti, Naren Nayak, and Vadim Sukhomlinov. 2019. Agile cold starts for scalable serverless. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*.
- [90] Ingo Müller, Renato Marroquin, and Gustavo Alonso. 2020. Lambda: Interactive data analytics on cold data using serverless cloud infrastructure. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 115–130.
- [91] Diana M Naranjo, Sebastián Risco, Carlos de Alfonso, Alfonso Pérez, Ignacio Blanquer, and Germán Moltó. 2020. Accelerated serverless computing based on GPU virtualization. *J. Parallel and Distrib. Comput.* 139 (2020), 32–42.
- [92] Hai Duc Nguyen, Zhifei Yang, and Andrew A Chien. 2020. Motivating High Performance Serverless Workloads. In *Proceedings of the 1st Workshop on High Performance Serverless Computing*. 25–32.
- [93] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2018. SOCK: Rapid task provisioning with serverless-optimized containers. In *Proceedings of the USENIX Annual Technical Conference*. 57–70.
- [94] OpenFaas. 2021. Home | OpenFaas - Serverless Functions Made Simple. <https://www.openfaas.com/>. (Accessed on 11/22/2021).
- [95] Apache OpenWhisk. 2021. Documentation. <https://openwhisk.apache.org/documentation.html>. (Accessed on 01/01/2021).
- [96] Nathan Pemberton and Johann Schleier-Smith. 2019. The serverless data center: Hardware disaggregation meets serverless computing. In *Proceedings of the 1st Workshop on Resource Disaggregation*, Vol. 4.
- [97] Duarte Pinto, João Pedro Dias, and Hugo Sereno Ferreira. 2018. Dynamic allocation of serverless functions in IoT environments. In *Proceedings of the IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE, 1–8.
- [98] Alexander Poth, Niklas Schubert, and Andreas Riel. 2020. Sustainability Efficiency Challenges of Modern IT Architectures—A Quality Model for Serverless Energy Footprint. In *Proceedings of the European Conference on Software Process Improvement*. Springer, 289–301.
- [99] Prometheus. 2021. Overview | Prometheus. <https://prometheus.io/docs/introduction/overview/>. (Accessed on 11/25/2021).
- [100] 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.
- [101] Ali Raza, Ibrahim Matta, Nabeel Akhtar, Vasiliki Kalavri, and Vatche Isahagian. 2021. SoK: Function-As-A-Service: From An Application Developer’s Perspective. *Journal of Systems Research* 1, 1 (2021).
- [102] Burkhard Ringlein, François Abel, Dionysios Diamantopoulos, Beat Weiss, Christoph Hagleitner, Marc Reichenbach, and Dietmar Fey. 2021. A Case for Function-as-a-Service with Disaggregated FPGAs. In *Proceedings of the IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE Computer Society.
- [103] Sebastián Risco and Germán Moltó. 2021. GPU-Enabled Serverless Workflows for Efficient Multimedia Processing. *Applied Sciences* 11, 4 (2021), 1438.
- [104] Fabiana Rossi, Matteo Nardelli, and Valeria Cardellini. 2019. Horizontal and vertical scaling of container-based applications using reinforcement learning. In *Proceedings of the 12th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 329–338.
- [105] Josep Sampé, Gil Vernik, Marc Sánchez-Artigas, and Pedro García-López. 2018. Serverless data analytics in the IBM cloud. In *Proceedings of the 19th International Middleware Conference Industry*. 1–8.
- [106] Joel Scheuner and Philipp Leitner. 2020. Function-as-a-service performance evaluation: A multivocal literature review. *Journal of Systems and Software* 170 (2020), 110708.
- [107] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J Yadwadkar, Raluca Ada Popa, Joseph E Gonzalez, Ion Stoica, and David A Patterson. 2021. What serverless computing is and should become: The next phase of cloud computing. *Commun. ACM* 64, 5 (2021), 76–84.
- [108] Lucia Schuler, Somaya Jamil, and Niklas Kühl. 2021. AI-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments. In *Proceedings of the 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 804–811.

- [109] Amazon Web Services. 2020. AWS Lambda - Developer Guide. <https://docs.aws.amazon.com/lambda/latest/dg/lambda-dg.pdf>. (Accessed on 08/31/2020).
- [110] Amazon Web Services. 2021. Amazon Braket Quantum Computers - Amazon Web Services. <https://aws.amazon.com/braket/quantum-computers/>. (Accessed on 11/26/2021).
- [111] Amazon Web Services. 2021. Amazon Elastic Inference - Amazon Web Services. <https://aws.amazon.com/machine-learning/elastic-inference/>. (Accessed on 11/17/2021).
- [112] Amazon Web Services. 2021. AWS Inferentia - Amazon Web Services (AWS). <https://aws.amazon.com/machine-learning/inferentia/>. (Accessed on 11/17/2021).
- [113] Amazon Web Services. 2021. AWS IoT Greengrass - Amazon Web Services. <https://aws.amazon.com/greengrass/>. (Accessed on 01/08/2021).
- [114] Amazon Web Services. 2021. AWS Trainium - Amazon Web Services (AWS). <https://aws.amazon.com/machine-learning/trainium/>. (Accessed on 11/17/2021).
- [115] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. 2019. Serverless computing: A survey of opportunities, challenges and applications. *arXiv preprint arXiv:1911.01296* (2019).
- [116] Mohammad Shahrad, Rodrigo Fonseca, Iñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *Proceedings of the USENIX Annual Technical Conference*. 205–218.
- [117] Vaishaal Shankar, Karl Krauth, Kailas Vodrahalli, Qifan Pu, Benjamin Recht, Ion Stoica, Jonathan Ragan-Kelley, Eric Jonas, and Shivaram Venkataraman. 2020. Serverless linear algebra. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 281–295.
- [118] Simon Shillaker and Peter Pietzuch. 2020. Faasm: Lightweight isolation for efficient stateful serverless computing. In *Proceedings of the USENIX Annual Technical Conference*. 419–433.
- [119] Paulo Silva, Daniel Fireman, and Thiago Emmanuel Pereira. 2020. Prebaking Functions to Warm the Serverless Cold Start. In *Proceedings of the 21st International Middleware Conference*. 1–13.
- [120] Arjun Singhvi, Arjun Balasubramanian, Kevin Houck, Mohammed Danish Shaikh, Shivaram Venkataraman, and Aditya Akella. 2021. Atoll: A Scalable Low-Latency Serverless Platform. In *Proceedings of the ACM Symposium on Cloud Computing*. 138–152.
- [121] Khondokar Solaiman and Muhammad Abdullah Adnan. 2020. WLEC: A Not So Cold Architecture to Mitigate Cold Start Problem in Serverless Computing. In *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 144–153.
- [122] Boubaker Soltani, Afifa Ghenai, and Nadia Zeghib. 2018. Towards distributed containerized serverless architecture in multi cloud environment. *Procedia computer science* 134 (2018), 121–128.
- [123] Gaetano Somma, Constantine Ayimba, Paolo Casari, Simon Pietro Romano, and Vincenzo Mancuso. 2020. When Less is More: Core-Restricted Container Provisioning for Serverless Computing. In *Proceedings of the IEEE INFOCOM - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 1153–1159.
- [124] Vikram Sreekanti, Chenggang Wu, Saurav Chhatrapati, Joseph E Gonzalez, Joseph M Hellerstein, and Jose M Faleiro. 2020. A fault-tolerance shim for serverless computing. In *Proceedings of the 15th European Conference on Computer Systems*. 1–15.
- [125] Manuel Stein. 2018. *Adaptive Event Dispatching in Serverless Computing Infrastructures*. Ph.D. Dissertation. Brunel University London.
- [126] Manuel Stein. 2018. The serverless scheduling problem and NOAH. *arXiv preprint arXiv:1809.06100* (2018).
- [127] 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 *Proceedings of the IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 1–10.
- [128] Ali Tariq, Austin Pahl, Sharat Nimmagadda, Eric Rozner, and Siddharth Lanka. 2020. Sequoia: Enabling quality-of-service in serverless computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 311–327.
- [129] Paul Turner, Bharata B Rao, and Nikhil Rao. 2010. CPU bandwidth control for CFS. In *Proceedings of the Linux Symposium*. Citeseer, 245.
- [130] Huaimin Wang, Peichang Shi, and Yiming Zhang. 2017. Jointcloud: A cross-cloud cooperation architecture for integrated internet service customization. In *Proceedings of the IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 1846–1855.
- [131] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the curtains of serverless platforms. In *Proceedings of the USENIX Annual Technical Conference*. 133–146.
- [132] Sebastian Werner, Jörn Kuhlentkamp, Markus Klems, Johannes Müller, and Stefan Tai. 2018. Serverless big data processing using matrix multiplication as example. In *Proceedings of the IEEE International Conference on Big Data (Big Data)*. IEEE, 358–365.

- [133] Zhengjun Xu, Haitao Zhang, Xin Geng, Qiong Wu, and Huadong Ma. 2019. Adaptive function launching acceleration in serverless computing platforms. In *Proceedings of the 25th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 9–16.
- [134] Hanfei Yu, Hao Wang, Jian Li, and Seung-Jong Park. 2021. Harvesting Idle Resources in Serverless Computing via Reinforcement Learning. *arXiv preprint arXiv:2108.12717* (2021).
- [135] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. 2020. Characterizing serverless platforms with serverlessbench. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 30–44.
- [136] Vladimir Yussupov, Jacopo Soldani, Uwe Breitenbücher, Antonio Brogi, and Frank Leymann. 2021. FaaSSten your decisions: A classification framework and technology review of function-as-a-Service platforms. *Journal of Systems and Software* (2021), 110906.
- [137] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. 2016. Apache spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (2016), 56–65.
- [138] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving. In *Proceedings of the USENIX Annual Technical Conference*. 1049–1062.
- [139] Haoran Zhang, Adney Cardoza, Peter Baile Chen, Sebastian Angel, and Vincent Liu. 2020. Fault-tolerant and transactional stateful serverless workflows. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 1187–1204.
- [140] Hong Zhang, Yupeng Tang, Anurag Khandelwal, Jingrong Chen, and Ion Stoica. 2021. Caerus: NIMBLE Task Scheduling for Serverless Analytics.. In *NSDI*. 653–669.
- [141] Michael Zhang, Chandra Krintz, and Rich Wolski. 2020. Edge-adaptable serverless acceleration for machine learning Internet of Things applications. *Software: Practice and Experience* (2020).