Cluster Computing (2024) 27:5571–5610
https://doi.org/10.1007/s10586-023-04264-8

A survey on the scheduling mechanisms in serverless computing: a taxonomy, challenges, and trends

Mohsen Ghorbian[1] · **Mostafa Ghobaei-Arani**[1] · **Leila Esmaeili**[1]

Received: 15 September 2023 / Revised: 24 December 2023 / Accepted: 28 December 2023 / Published online: 18 February 2024

Abstract
In recent years, serverless computing has received significant attention due to its innovative approach to cloud computing. In this novel approach, a new payment model is presented, and a microservice architecture is implemented to convert applications into functions. These characteristics make it an appropriate choice for topics related to the Internet of Things (IoT) devices at the network's edge because they constantly suffer from a lack of resources, and the topic of optimal use of resources is significant for them. Scheduling algorithms are used in serverless computing to allocate resources, which is a mechanism for optimizing resource utilization. This process can be challenging due to a number of factors, including dynamic behavior, heterogeneous resources, workloads that vary in volume, and variations in number of requests. Therefore, these factors have caused the presentation of algorithms with different scheduling approaches in the literature. Despite many related serverless computing studies in the literature, to the best of the author's knowledge, no systematic, comprehensive, and detailed survey has been published that focuses on scheduling algorithms in serverless computing. In this paper, we propose a survey on scheduling approaches in serverless computing across different computing environments, including cloud computing, edge computing, and fog computing, that are presented in a classical taxonomy. The proposed taxonomy is classified into six main approaches: Energy-aware, Data-aware, Deadline-aware, Package-aware, Resource-aware, and Hybrid. After that, open issues and inadequately investigated or new research challenges are discussed, and the survey is concluded.

Keywords Serverless computing · Function as a service · Scheduling algorithms · Energy-aware scheduling · Data-aware scheduling · Deadline-aware scheduling · Package-aware scheduling · Resource-aware scheduling · Makespan · Serverless edge computing · Resource management · Performance evaluation

1 Introduction

Cloud computing has gained significant attention in recent years as an innovative and compelling method of deploying cloud applications. Serverless computing has been employed as a result of the recent evolution of enterprise application architectures into microservice-based architectures. Furthermore, with the help of this technology, developers will have access to a simplified programming model, simplifying the process of creating cloud applications and eliminating most—if not all—of the operational concerns associated with infrastructure configuration. Cloud-native code that responds to events can be deployed rapidly [1, 2]. The function being executed by serverless computing must be stateless and idempotent, which means that it may be re-executed without causing any harm if it fails. Therefore, the discussion of the system's design is now shifting towards strategies for managing containers and developing software to maximize the system's performance based on a function-centric infrastructure [3, 4]. Cloud providers can utilize serverless computing to manage the entire development process and reduce operational costs by optimizing and managing cloud resources efficiently from the cloud provider's perspective [5, 6]. Serverless Computing enables application developers to break up large applications into smaller components, allowing these components to be scaled individually.

✉ Mostafa Ghobaei-Arani
mo.ghobaei@iau.ac.ir

[1] Department of Computer Engineering, Qom Branch, Islamic Azad University, Qom, Iran

However, this poses a new problem: managing many functions coherently and allocating resources for them [7, 8]. Depending on the required functionality, serverless architectures can be used interchangeably with traditional architectures. The decision to use serverless technology will likely depend on other non-functional considerations, including how operations are carried out, the cost, and the application workload characteristics [9, 10]. Serverless performance management practices can result in various issues, including inconsistent and inaccurate limitations, inefficient resource allocation, inadequate runtimes, mid-chain function drops, concurrency collapses, and undocumented function priorities from current practices. Therefore, to alleviate these problems and create an efficient resource management system, it is necessary to use a resource allocation scheduler suitable for serverless and chained functions. The scheduler is one of the main components of the system resources management, allowing its performance to be compared across the serverless resources management service providers. One purpose of this feature is to assist the user in selecting the appropriate scheduler for the environment in which it is an implementation system. As a result, identifying the purpose and environment of the scheduling implementation is crucial to ensure that it accomplishes its objectives in a system [11].

1.1 Research motivation and challenges

Serverless computing systems can utilize different schedules based on the type of workload and implementation objectives. Depending on the context and application, a scheduler can deliver the best performance, while if it is used in another context and application, it may deliver the worst results. Thus, selecting a specific scheduler based on the goals of each environment and program is more efficient than choosing a variety of schedulers. Due to the emerging nature of serverless technology, using of this will inevitably face many challenges, particularly when allocating resources. In contrast, to implement a successful and efficient resource allocation process, it is necessary to determine an appropriate schedule to ensure that resources are not used inefficiently during the implementation process. In the field of IoT, serverless computing has emerged as one of the most promising research areas [12]. In light of this, there is no doubt that selecting a suitable and efficient scheduler in serverless computing is a good fit for IoT applications, especially when it intersects with the discussion of edge computing and fog computing infrastructures [13]. Therefore, issues using a suitable scheduler to carry out the resource-allocating process effectively and as efficiently as possible to achieve maximum efficiency in this technology is critical. However, there has been little research conducted in the area of providing schedulers in serverless environments. However, authors have attempted to provide appropriate schedulers with the intended goals; they propose appropriate solutions for existing challenges. Despite the importance of this topic, to our knowledge, no comprehensive and detailed study has been explicitly published regarding the use of appropriate schedules in the context of serverless computing.

1.2 Our contribution

This research will systematically review existing studies for scheduling in the serverless computing field and evaluate various scheduling techniques to provide a comprehensive and systematic assessment of the mechanisms for selecting and implementing schedulers based on their characteristics and implementation. Several main contributions are presented in this review, which can be summarized as follows:

- Reviewing published articles on serverless computing related to the scheduling approach provides insights into each study's scheduling methodologies and strategies.
- We are analyzing and evaluating the latest scheduling approaches and techniques in serverless computing.
- We are examining current approaches and developing a classification based on these findings.
- We are discussing any underexplored or underserved future research challenges that could be addressed to improve scheduling techniques for serverless computing environments.

1.3 Organization of the paper

This paper is organized as follows: Sect. 2 presents background on serverless computing scheduling and various features of scheduling applications; an overview of some related survey articles is presented in Sect. 3, along with a comparison of these articles. The research method used in this study is described in Sect. 4; Sect. 5 concludes with a classification of the techniques discussed, a summary description of the plans, and a comparison of these techniques. In Sect. 6, some discussions and comparisons are also presented. There are some open issues outlined in Sect. 7 that need to be addressed in the future. Finally, in Sect. 8 the conclusion is presented.

2 Background

In this section, we provide a brief overview of serverless computing and the scheduling process and will explain the most important parameters involved in the scheduling process.

2.1 Overview of serverless computing

The serverless computing model enables cloud providers to provide infrastructure management for customer resources based on their needs. Serverless applications still require servers, but developers are no longer required to manage them. Thus, serverless enables developers to concentrate on developing serverless applications by implementing auto-scaling based on resource consumption [14]. Hence, Providers of cloud services that advocate using function-as-a-service (FaaS) in an organization should be capable of assisting groups seeking to utilize FaaS. It is possible to execute small, modular fragments using a serverless architecture and a function-based model with the help of FaaS. So, Software developers can use these services to perform client-required functions. The serverless architecture eliminates the need for developers to maintain dedicated servers since applications are only activated when used, eliminating the need to support dedicated servers [15]. So, at the end of the function execution, once the function has been completed successfully, it can be terminated quickly to free up the same amount of computing resources for other functions. Using FaaS, it is possible to access applications on demand. Also, the applications can be executed by a platform that coordinates and manages the application resources so that they remain secure throughout the entire process of running the application. Thus, cloud service providers can simplify operations, reduce costs, and improve scalability by managing servers to implement applications [16]. This property permits them to concentrate more on developing application code than managing servers. Generally, a FaaS model is an excellent choice for simple, repetitive tasks, such as scheduling routine tasks, processing queued messages, or handling web requests regularly. Functions in an application are collections of tasks defined independently as code pieces and executed separately as individual tasks. The most efficient way to utilize resources is to scale a single function rather than an entire application because it does not require a whole micro-service or application [17]. Cloud computing is a method of federating many resources into a single machine, and it is similar to parallel computing in many ways, such as clustering and grid computing. Still, The main characteristic of cloud computing is, however, virtualization. Thus, computing resources can be scheduled as a service for clients in this approach. A significant development in computing is the availability of a wide range of highly flexible devices, known as nodes, which can be deployed anytime and anywhere; only a fee is incurred when they are employed. Hence, with traditional computing environments or data centers, achieving this goal would not be possible due to the limitations of the existing technology [18]. It is essential to note that unused, underused, and inactive resources significantly impact energy waste. Therefore, optimally scheduling cloud resources is one of a company's most challenging tasks in managing cloud resources [19]. There are many factors to consider when allocating resources to cloud workloads, such as the applications hosted in the cloud and the energy consumption of computing resources. Appropriate techniques for allocating resources are lacking to address the challenges associated with uncertainty, dispersion, and heterogeneity within a cloud environment. Developing more appropriate strategies for giving resources within the cloud environment is necessary [20]. By using a cloud-based approach that automatically manages computing resources based on their consumption as an influencing factor of service quality, it is possible to resolve this issue. In addition to increased scalability and faster development, this approach can reduce costs [21].

2.2 Serverless computing features

A serverless computing approach will be able to respond to all these requests and provide excellent resource management conditions with these features. Several essential advantages of the serverless computing approach include scalability, security, visibility, faster development, and reduced costs.

2.2.1 Scalability

The high scalability offered by serverless computing technology means that developers will no longer have to worry about the impact of heavy traffic since the system is highly flexible and scalable. This property of serverless computing is one of its most attractive features. Compared to previous architectures, this architecture can address all the concerns about scalability more efficiently than previous architectures [22]. Scalable applications can handle the demands of many clients without experiencing a performance loss whenever the number of requests increases. As a result, auto-scaling instances must be designed in such a way that they can handle traffic fluctuations regardless of the number of users and requests. Consequently, using only resources essential to the project is more efficient than wasting unnecessary resources [23].

2.2.2 Visibility

It is essential to monitor a system to keep it healthy. Collecting metrics such as CPU utilization, error logs, and network traffic can use these data as inputs into incident alert tools to prevent system failures. Sending out alarms and notifications to staff can alert them to security, outages, and errors that have occurred [24]. Hence, regarding the visibility of systems and how they perform under varying circumstances, serverless computing achieves different goals than serverless testing and monitoring. As a result, serverless observability can provide insights into system efficiency under various circumstances. A serverless computing system allows for two types of visibility:

- Testing: testing allows for identifying known problems.
- Monitoring: monitoring allows for evaluating the system's health according to available metrics.

So, Instrumentation can be used to gather as much information as possible from an application to identify and resolve unknown problems. Due to the disparate, isolated, and highly transient nature of event-driven functionality, maintaining visibility in serverless applications is essential and challenging [25].

2.2.3 Faster development

By utilizing serverless computing technology and approach, developers can focus more on developing code for the applications they are creating, increasing production rates as rapidly as possible. Finally, this property increases their efficiency while developing the software they are growing, resulting in more excellent performance [26]. By doing so, developers can spend less time on deployment and have a faster development turnaround, thus increasing their productivity. So, compared to traditional technologies, large backend systems can be constructed in less time by abstraction and reduction of complexity due to technology abstraction and reduction of complexity. As a result, the product development process is typically accelerated, resulting in shorter delivery times, faster deliveries, and more significant business growth. Also, the ability to rapidly update existing products with minimal friction and expense and to constantly experiment with new ideas [27].

2.2.4 Security

Security of the system has become more challenging in light of the rapid evolution of serverless applications and their changing structure. Since more information and resources are available in this approach, some novel challenges and complexities arise. Unlike its forerunners, serverless security appears inherently secure due to its characteristics, such as short function duration. In addition, due to its architecture structure, it may also inherit security features developed for other virtualization platforms [28]. Security challenges associated with serverless approaches can create new, distinctive security threats. Furthermore, the development of serverless applications will require a significant change in mindset on the part of developers, both in terms of how applications are developed and their protection against malicious attacks. As more resources equate to more permissions, a unique approach is required to introduce security policies into provider systems. Thus, an organization with more resources has more permissions to manage, making it more challenging to determine the permissions for each interaction [29]. It is essential to provide visibility into serverless applications because they use various cloud services across multiple versions and regions, making it difficult to find and solve a problem. Thus, the visibility feature can be utilized as a solution to security challenges if deployed in a secure environment. Hence, using the visibility feature, which combines two parts of tests with automatic monitoring, one can detect configuration risks and eliminate function permissions using automated methods [30].

2.2.5 Reduced costs

Clients can save their application's life cycle costs using serverless computing. In addition to simplifying the development process, serverless computing also improves the development process's efficiency by eliminating idle computing time. Also, clients can receive services at the lowest possible cost, which is both an easy process for them to follow and cost-effective [31]. The service can be scaled up to serve millions of clients simultaneously with no additional cost or effort. It is not necessary to provision, manage, or update server infrastructure during project maintenance, so clients will only pay the cost for what they are using. The cloud service provider handles all this, so clients don't have to worry about it [32].

2.3 Serverless computing architecture

An approach known as serverless architecture is wherein a cloud provider runs code pieces and dynamically assigns resources to the customer's requirements. Additionally, this approach can be applied to a wide range of scenarios, as functions can be developed that alter resource configurations to accomplish specific infrastructure management tasks that can be achieved by using the model in particular techniques. In other words, serverless computing utilizes the full potential of cloud computing because it allocates resources in real-time to meet the actual requirements of clients and scales up and down as required by the client in real-time. Therefore, the client only has to pay for those resources that are used and do not have to pay for those resources that are not necessary [33]. By using serverless computing, all resources will automatically be scaled back to zero when the application is inactive or when clients have no requests. In serverless architectures, a third-party cloud service provider provides computing services based on the functions of their cloud services while managing infrastructure surveillance as part of their cloud architecture. Ephemeral containers, typically composed of multiple components, are commonly used to achieve this functionality. In addition to database events, file uploads, and queues, they can also be triggered by a wide range of other events, including monitoring signals, cron jobs, and Hypertext Transfer Protocol (HTTP) requests [34]. The client will not have to worry about the server in a serverless computing model since the provider's architecture abstracts away the server from the client's point of view. The architecture of serverless computing is illustrated in Fig. 1.

- *Client* Serverless functionality relies heavily on the client interface. In addition, interface designs must support extremely high or shallow volumes of data transfers. An effective interface should handle short bursts of stateless interactions [35].
- *Security* Running a serverless system without a security service is impossible due to the need to ensure security for numerous requests simultaneously. On the other hand, it is impossible to keep track of previous interactions due to its stateless nature, so ensuring an authentication process has been implemented before returning a response is critical. Serverless systems typically provide clients with temporary credentials through the security service, including authentication and database [36].
- *API Gateway* A function-as-a-service service and its client interface are linked using the API gateway. When the client triggers an event, the API gateway relays the information to the function-as-a-service service to trigger a function [37].
- *Functions as a service* Function-as-a-service, the most important component of serverless computing, assigns resources to specific scenarios based on their needs. It is a serverless method of running a function in any cloud environment, allowing developers to focus on writing code rather than worrying about infrastructure requirements or building and maintaining it themselves [38].
- *Backend as a service* A cloud-based service handles the backend of an application in the backend as a service model. By utilizing several critical backend features, clients can create an exceptionally robust backend application using this component, resulting in an unusually practical backend application that will be developed efficiently and effectively [39].

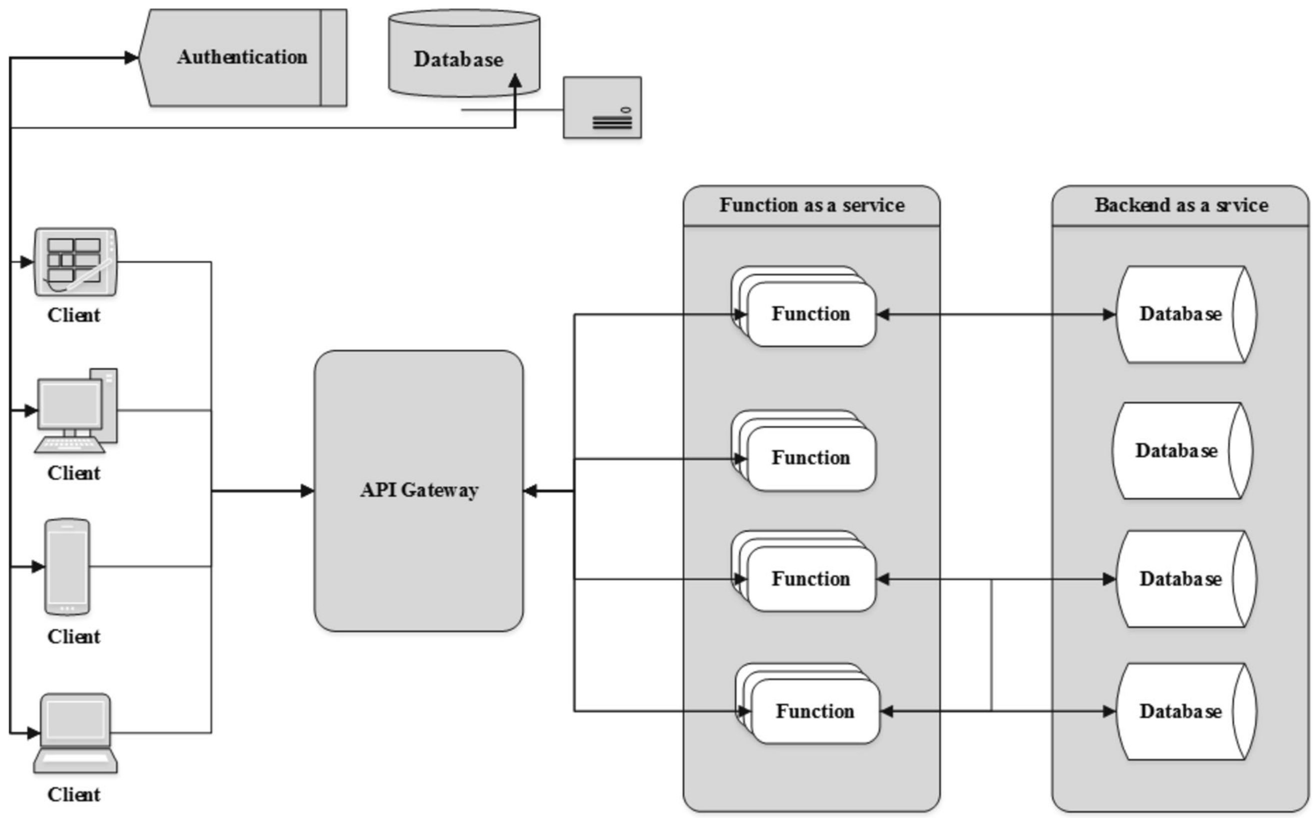

Fig. 1 Serverless architecture's primary components

An event trigger mechanism is used in serverless computing. By implementing this approach, an application may need to fetch and transmit data in various situations to function at its optimum since it might require it to fetch and transmit data under different conditions. When a client creates an event trigger, it is not uncommon for the application to fetch and transmit a particular piece of data to create the event trigger inquiry [40]. Typically, this is called in the world of programming an event. When a client initiates an action to trigger an event, the application dispatches it to the cloud service provider in reaction to the client's action when the application is launched. According to rules defined for the execution of a function, cloud services allocate dynamic resources for the execution of the function. Before proceeding with the process, the client must consider one point: once the function has been invoked, it will provide the result of its execution to the client. It is impossible to allocate resources without a request from the client, and it is impossible to store data without such a request. Consequently, this allows the application to run in real-time while reducing storage and cost requirements. This data may be presented to the client at any time to provide the client with the most current and updated data [41].

2.4 Scheduling in serverless computing

The execution pattern of a serverless system shifts the responsibility of managing application resources from clients to the cloud service provider as part of the serverless architecture model. Since serverless models use real-time allocation of resources, providers of serverless applications must be able to manage resource allocation autonomously during application execution [42]. When a serverless platform is in its initial stages, the management can be challenging due to the limited information available about the resources different functions need. Consequently, in this case, they may need help making informed resource allocation decisions, which, in turn, may result in them being unable to make informed resource allocation decisions. Serverless architecture analysis revealed that CPU utilization could often be a source of contention between applications, mainly if the applications are computationally intensive, resulting in high response latency [43]. Due to this, providers must be aware that arbitrarily determined resource allocation policies may result in a conflict of resources for applications during their runtime, thus violating the service level agreement signed by the client. Therefore, it is crucial to ensure that the provider's resources are managed dynamically to ensure the smooth operation of the provision of services [44, 45]. Scheduling algorithms are vital to serverless computing because they can minimize response times and maximize resource utilization by minimizing response times. A scheduling algorithm allows computing resources to be allocated dynamically based on the consumption requirements according to a consumer's requirements. Several strategies and methodologies have been developed to maximize resource scheduling efficiency and achieve the best results. Therefore, various scheduling strategies and approaches determine resource allocation. Scheduling mechanisms for serverless computing generally include two approaches: affinity and load-balancing scheduling. A load-balancing scheduling approach distributes the load equally among all the workers in the system in order to prevent worker overloading. In this approach, for each request that the system receives as a load, a function must be executed as a program within a container so that the system can respond to the request. There are various approaches to assigning load to workers using the scheduling method through load balancing. They are:

- In the round-robin distributed approach, the scheduling mechanism ensures that the load is distributed evenly among the workers.
- In the least load assignment approach, the scheduling mechanism ensures that the load is assigned to the employee who has the fewest requests.
- In the random distribution approach, the scheduling mechanism ensures that all workers are assigned a random workload.

The selection of the suitable load-balancing approach is based on the system implementation goals. In the affinity-based approach, dispatches send the same kind of request to the same worker when the worker is not overloaded. These approaches can avoid overloading workers and efficiently reuse discontinued containers at times when workloads appear to be stable. It is presumed that requests can quickly retrieve available containers on selected workers [46]. In Fig. 2, the serverless computing scheduling mechanism is illustrated based on the execution of functions in containers on the workers.

According to Fig. 2, the requests are first sent to the scheduler. Serverless computing requires the execution of functions to respond to each of the sent requests. As the execution environment in serverless computing calculations is assumed to be a functions execution-based environment in the form of containers, each of these functions must be executed in a container in order to respond to requests. Hence, for the purpose of executing the functions, a container needs to be created in each worker and used as an environment to run the functions. The scheduler in this mechanism is responsible for distributing the requests. How loads are distributed or, in other words, how requests are distributed among workers depends on the scheduling mechanism's approach to distributing requests. After

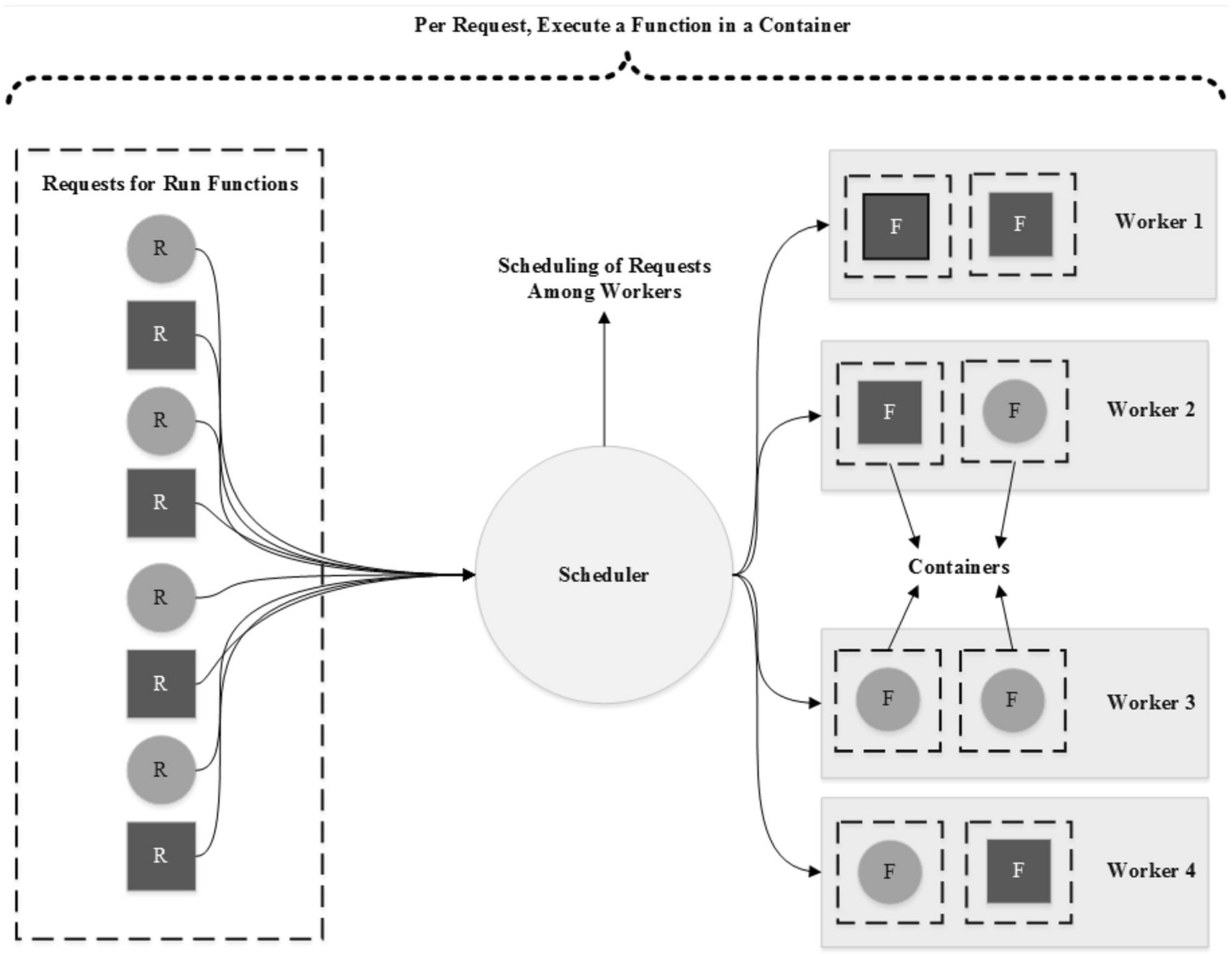
Per Request, Execute a Function in a Container
Requests for Run Functions
R
Scheduling of Requests Among Workers
Scheduler
F
Worker 1
Worker 2
Containers
Worker 3
Worker 4

Fig. 2 Scheduling mechanisms in serverless computing

determining the load distribution approach, the scheduler tries to distribute the requests among the workers based on the chosen approach. Scheduling algorithms in serverless computing ensure the system's goals are achieved. Therefore, serverless computing service providers must choose a scheduler approach based on sufficient knowledge to ensure they can achieve a better and more effective result.

2.5 Scheduling metrics

Understanding the mechanisms of the algorithms, it is also possible to define a set of criteria for evaluating the performance of scheduling algorithms across a wide range of implementations [47]. In the following, some of the most critical factors that can be used to demonstrate the performance of a scheduling algorithm will be discussed.

2.5.1 Response time

During the implementation of scheduling algorithms, the response time factor of serverless computing is an essential factor. Various factors determine the response times, including the scheduler's efficiency, the algorithms' complexity, the serverless platform characteristics, the simultaneous nature of the application, and the communication delay. Therefore, to determine whether this factor is efficient, a sequence of sequential requests must be sent to the scheduling algorithm so that the scheduling algorithm evaluates by response time. When the workload contains heavy computing functions, the mean response time (application performance) decreases with the number of executors increases. The metric is calculated based on the Eq. (1) [48].

$$Response\ time = \frac{1}{\mu} \times \sum_{a=1}^{\mu} Re_a \tag{1}$$

In this equation, μ is the number of requests that have been processed for function execution, and Re_a is the response time of the function execution request.

2.5.2 Throughput

The performance throughput of scheduling algorithms can be evaluated based on the number of requests received for resource allocation for functions with different executors. Hence, this case can be suitable when evaluating factor throughput in scheduling algorithms for resource allocation for requested functions. Various implementations can show that schedulers can quickly become a bottleneck when request rates are high, making it difficult for executor resources to utilize fully. So, it's crucial to note that selecting scheduling algorithms inappropriately by providers can limit scalability and result in low throughput due to high invocation overhead. The calculation of this metric is based on the Eq. (2) [49]. The underlying assumption for throughput is that for large enough, the average service time of terminated jobs in the same type of job (local a or offloaded μ.) is based on its mean value. Then, the system operates like a system with a predictable schedule for which the scheduling process is throughput. Take into account any arrival rate function execution $\vartheta \epsilon \Theta$ in $N = \{n \epsilon N\}$ machines. Since Θ it is an open set, there exists $\psi > 0$ such that $\vartheta = (1+\psi)\vartheta \epsilon \Theta$. Then

$$Throughput = \left(\frac{\vartheta_{\mathrm{H},n}}{1+\Psi} : H \epsilon \zeta, n \epsilon \mathrm{N} \right) \tag{2}$$

is a decomposition of ϑ, and for any $n\epsilon$N.

$$\left(\sum_{H:n\epsilon H} \frac{\vartheta_{\mathrm{H},n}}{a} + \sum_{H:n\epsilon H} \frac{\vartheta_{\mathrm{H},n}}{\mu} \right) \leq \frac{1}{1+\Psi} \tag{3}$$

In this equation $\vartheta = (\vartheta_1, \vartheta_2, \vartheta_3, \ldots, \vartheta_{N+1})$ be defined as

$$\vartheta_n = \sum_{H:n\epsilon H} \vartheta_{\mathrm{H},n}, n\epsilon \mathrm{N} = \sum_{n=1}^{N} \sum_{H:n \notin H} \vartheta_{\mathrm{H},n} \tag{4}$$

Then, ϑ is a possible arrangement of arrival allocation for any function so that the average is the arrival rate $H = \{\zeta \epsilon H\}$ allocated function to the queue.

2.5.3 Latency

In a serverless computing platform, the latency factor is when a client requests a function when the scheduling algorithm attempts to allocate resources and when the function is executed. During the execution function, request time, request analytics time, and resource allocation are included in the duration. It is of particular importance for latency-sensitive applications. Ideally, the scheduling algorithm should be able to handle and manage all requests made from different locations and with varying amounts of resources in real-time if it receives a large number of requests. Thus, it means the scheduling algorithm should be capable of performing accurate predictions by obtaining current information and sufficient information about the execution environment. Hence, the scheduling algorithm can allocate resources appropriately, and it is possible to improve resource management to determine the most appropriate actions. Finally, reducing the server resource consumption can simultaneously reduce the server's resource consumption and the job latency. It is calculated based on Eq. (5) [50].

$$Latency = \lambda_n + v_\rho^\eta \vartheta_\eta \forall_\eta \epsilon \Lambda_\vartheta \tag{5}$$

where v_ρ^η is determine the processing delays of η represents the precedence constraint of the schedule of the jobs of N_e η on ϑ that is one function from aggregate existing functions $V = \{\vartheta \epsilon V\}$. Also, in this equation makes sure that the total delay incurred by a job that is used for the execution of any function of $N_e \eta \epsilon \Lambda_\vartheta$ does not surpass its latency threshold.$\lambda_n \epsilon$ Z to determine the time slot at which the job of $N_e \eta \epsilon \Lambda_\vartheta$ starts execution on function ϑ.

2.5.4 CPU usage

The CPU usage factor allows for deploying resource control mechanisms in serverless computing architecture that can be utilized to evaluate resource management in serverless architecture providers. Therefore, by using this factor, providers can control the amount of CPU consumed during the execution of functions in an application, thus allowing for the evaluation of resource utilization. In other words, this factor makes it possible to evaluate the performance of scheduling algorithms in serverless computing systems. By measuring the number of CPU resources consumed by the requested functions, this evaluation factor can be used to efficiently execute a system, resulting in a significant reduction of the number of CPU resources consumed and an improvement in the overall performance of the serverless computing system. This metric is calculated according to Eq. (6) [51].

$$CpuUsage(k_b, \Delta t) = \frac{\sum_{\{\forall x_a \to k_b\}} \left(C_a \cdot section\left(R_b^a, \Delta t\right) \right)}{C_b \cdot \Delta t} \tag{6}$$

In this equation, $\mathcal{N}$ nodes $k_b \in K$ that are required be functions by microservices represent instances $x_a \in X$. Hence, each node k_b is bounded by compute resources, specifically CPU limitation C_b. On the other side, C_a is CPU usage and $section\left(R_b^a, \Delta t\right)$ provides the section of runtime R of the service x_a on the node k_b that is placed inside Δt.

2.5.5 Energy consumption

It is important to consider this factor when evaluating the performance during the resource allocation process of a scheduling algorithm in a serverless environment. This factor can determine the amount of energy a scheduling algorithm consumes. By analyzing the structure of a serverless computing implementation environment carefully, it is possible to calculate the energy consumption of tasks in heterogeneous or heterogeneous environments according to the factor. By utilizing this factor, it will be possible to develop a solution that reduces the amount of energy consumed by serverless computing to the greatest extent possible [52]. Besides providing information regarding the proportions and amounts of resources used by a scheduling algorithm during the allocation process, this factor can also be used to demonstrate its effectiveness. The ideal solution is minimizing resource consumption when allocating resources in scheduling algorithms that would reduce the total cost. Using this factor, a serverless computing provider can select the appropriate algorithm to be implemented during the provision of services, as each application requires a unique approach or, in other words, a suitable algorithm. When providing efficient services, this factor is vital in reducing resource consumption as much as possible. In this case, the metric is calculated in accordance with Eq. (7) [53].

$$Energy = \left(\frac{C_\rho}{Energy_P \times (1-\varphi_P)} + \frac{C_R}{Energy_R \times (1-\varphi_R))} + \frac{C_I}{Energy_I \times (1-\varphi_I))}\right) \tag{7}$$

It is important to note that different functions consume varying amounts of energy based on their application. As a consequence, some functions require a processor to operate. In contrast, some applications require a disk for storing data, while others require the exchange of information, which may result in varying levels of energy consumption. According to this equation, $Energy_P \times (1-\varphi_P)$ is the energy used to execute a processor-based function, $Energy_R \times (1-\varphi_R)$ is the amount of energy used for transfer rates-based function, and $Energy_I \times (1-\varphi_I)$ is the amount of energy used for a disk IO-based function. Also, in this equation, the amount of processing energy consumed, data transfer rate, and disk IO rate are represented by φ_P, φ_R, and φ_I, respectively. The variables C_ρ, C_R, and C_I indicate the amount of energy required by a machine based on the processing type, IO rate, and data transfer rate.

2.5.6 Cost savings

Platforms based on serverless computing offer a pay-per-use model in which users are billed according to the amount of computation memory and time their functions require. Despite some scheduling algorithms in serverless computing environments striving to maximize resource efficiency during function execution, algorithms may neglect to take application-specific factors into account. Consequently, they may violate service level agreements (SLAs), not maximize resource utilization, not achieve optimal resource utilization simultaneously, and cause increased costs. By defining policies in the format of scheduling algorithms, serverless service providers can minimize the cost of resource consumption while meeting the client's application requirements, namely, the deadline and attention to specific details [54]. The proposed algorithms must be sensitive to deadlines, efficiently increase the provider's resource utilization, and dynamically manage resources to improve response times to functions, thereby solving the increasing cost challenge [55]. Hence, optimizing resource costs for end clients by reducing providers' time to respond to functions is possible. Thus, an ideal mechanism for the serverless service provider to choose a scheduling algorithm included placing functions and allocating adequate resources to the containerized function implementation to maintain the resource cost at an optimal level while meeting the desired client requirements. This metric is calculated using Eq. (8) [56]. In serverless computing, the utility of node n has been defined as the revenue derived from the execution of computation functions minus the processing costs and the penalty for overflow. In other words, the utility has been defined as:

$$Cost = C_m^q q_m(\beta) - C_m^y y_m(\beta) - C_m^z Z_m(\beta) \tag{8}$$

where C_m^q, C_m^y and C_m^z are factors affecting costs. $C_m^q q_m(\beta)$ is the amount of revenue that the serverless computing node m receives from the service provider as a result of performing serverless tasks for the cloud center. $q_m(\beta)$ is defined as the reduction in processing delay over the provider center.

2.6 A brief overview of virtualization and containerization technologies

The process of deploying and implementing applications is different in different environments. Hence, in this section, we discuss how to deploy and run applications using two approaches: virtualization and containerization. With these two approaches, as well as serverless computing, applications can be implemented and deployed across a variety of frameworks and environments.

2.6.1 Containerization approach

Due to the fact that containers present virtualization at the operating system (OS) level, they are more lightweight and agile than traditional virtual machines (VMs) [23]. The term container refers to an isolated, lightweight environment that packages an application along with its requirements in an organized manner. It is important to note that containers share the kernel of the host OS. However, each container has its own network connectivity, file system, and processes. It makes it simpler to deploy applications in a variety of environments when you use containers, as they provide a reproducible runtime and consistent environment. Compared with virtual machines, they are more lightweight, have a faster startup time, and consume fewer resources than VMs. Containers are often implemented in conjunction with container orchestration frameworks, such as Docker and Kubernetes [57].

2.6.2 Virtualization approach

An OS can be implemented as a virtual machine (VM) by using software simulation that plays a hypervisor role on a physical computer system. A VM consists of its virtual hardware, such as memory, storage, CPU, and network interface. A VM provides strong isolation among applications and can run multiple operating systems (OSs) or versions of the same OS simultaneously. VMs are flexible and portable, yet they can consume a lot of resources due to the requirement of running an entire OS on each VM [58].

Compared to VMs, containerized applications provide portability and lightweight isolation, and serverless offer cost efficiency and automatic scalability for event-driven applications. It is important to consider the specific needs of the application, the development mechanism, considerations regarding resource utilization, and the level of isolation required of the team when selecting between them. Comparison of serverless, container, and VMs is shown in Table 1.

2.7 Machine learning in serverless computing

Machine learning is one of the fields that have contributed to the improvement and efficiency of serverless computing systems. Combining these two technologies provides the foundation for the use of artificial intelligence algorithms in serverless computing systems. In addition to improving resource management and runtime optimization, this combination also enhances predictive capability and accuracy. Incorporating these two technologies allows systems to act as more intelligent units, take advantage of each other, and interact more effectively with their input environment. In addition to improving the development prospects for serverless computing systems, this approach is effective in a wide range of applications, such as image processing, language translation, and data prediction [59]. One of the fundamental challenges associated with serverless computing is scheduling, which is exacerbated by fluctuations caused by changes in requests and workloads. Using machine learning to optimize the scheduling process in serverless computing is an innovative and smart solution. By using this approach, resource managers are able to analyze execution patterns, predict resource requirements, and improve system performance. In serverless computing, machine learning can be used to improve the scheduling process in a number of ways, including:

- *Increasing the potential for response* The use of machine learning in scheduling serverless computing also increases the responsiveness of the systems since they can more easily adapt to changes in requests and the environment and are better able to respond efficiently to user needs and system functions.
- *Improving resource demand prediction accuracy* In serverless computing scheduling, machine learning improves the prediction of resource demand. An accurate analysis of data and resource requirements over time allows machine learning models to estimate better the amount of computing resources required at a given moment.
- *Optimizing smarter decisions* The use of machine learning in serverless computing allows the systems to react in real-time to current information and environmental conditions and to make intelligent decisions based on needs and constraints.
- *Managing resources intelligently* Machine learning technology can be integrated into serverless computing scheduling to facilitate intelligent resource management. By estimating resource needs accurately and implementing intelligent decisions, resources can be managed in the most optimal manner possible to avoid differences between system needs and resource allocations [60].
- *Enhancements to the estimation of runtimes* In serverless computing scheduling, machine learning has a fundamental role to play in improving execution time estimates. Machine learning models are capable of identifying execution patterns and providing more accurate estimates of the time required to execute code by carefully analyzing past execution time data.
- *Predicting future needs* Serverless computing can utilize machine learning as a powerful tool for predicting future needs in order to optimize the scheduling process. By analyzing past data and request change patterns, machine learning models can predict future needs in the best possible way in order to prepare the scheduling process in the system to deal with future challenges in the best possible way.

Table 1 Comparison of serverless, containerization, and virtualization technologies

Comparison criteria	Serverless	Containerization	Virtualization
Quick start	Instant	Fast	Slow
Resource consumption	Low	Medium	High
Vertical scalability	Automatic	Scalable	Manual/expense
Horizontal scalability	Scalable	Limited	Limited
Operation	Local	Shared	Separate
Operational issues	Low	Moderate	Complex
Configuration capability	Limited	Medium	Abundant
Transferability	Easy	Easy	Medium
Flexibility	Very High	High	Good
Platform limitations	None	Limited	Limited
Cost	Inexpensive	Inexpensive	Expensive
Management	Easy	Medium	Complex
Support time	Medium	Medium	Long
Cost model	Fixed resources	Fixed resources	Pay-per-use model
Security	Secure	Separate	Separate
Reconfiguration required	Low	Low	Moderate
Isolation	Low level	High level	Full

- *Enhancing credibility* The use of machine learning in the scheduling of serverless computing can increase the system's reliability. Users are assured of a high degree of accuracy in estimations, smarter decisions, and forecasting of resource needs, thus ensuring that the system can provide quality services in the shortest amount of time [61].

3 Related works

This section will analyze recent papers reviewed on scheduling in serverless computing. In the following, we will seek to give each survey article's primary advantages and disadvantages. So, we will take an in-depth view of some articles in the literature. Not reviews work studies scheduling in Serverless computing. Therefore, it can be said that this work is one of the first studies in this domain. One of the notable scheduling studies on serverless computing was accomplished by Alqaryouti and Siyam [62]. Their survey considered various propositions incidental to scheduling tasks in clouds. These propositions were classified according to their target functions by minimizing execution time, execution cost, and multi-targets (time and cost). Applying a hybrid perspective to serverless computing alongside the IaaS approach in the form of one technique leads can significantly reduce issues arising from dependency because issues arising from dependency on the system are from the underutilization of resources. Therefore, their suggested algorithm compresses the schedule by re-ordering the tasks for maximal usage of the scheduled indolent slots related to the dependency restrictions. As a result, as a benefit of this study, solutions were proposed for the problem of scheduling aspects of workflows from the clouds focused professionally. Moreover, the article goes through some disadvantages as follows:

- Inadequate coverage and weak of recently published papers at the time of publishing
- Lack of presentment of an organized template for choosing articles.
- Ignoring some paramount agents when presented with solutions proposed to the problem of scheduling, paramount agents like productivity boost, Cost-effective in Scheduling
- Ignoring one of the essential features of scheduling on serverless computing in the name of effortless efficiency in reviewing the entire article and the proposed solutions.
- Not to pay attention to the future path to study the situations and challenges ahead.

Kjorveziroski et al. [63], in their research, reviewing works disseminated between the years 2015 and 2021, presented a systematic survey study of the recent progressions they created In connection with the subject of serverless computing to the edge of the network. They could determine eight fields in which existent serverless edge research was concentrated. Hence, they used the obtained classifications to study the selected articles and, in the following, could present increased interest in using serverless computing at the network's edge. One subject that is an occupied study, specifically in recent years, is the placement of a real edge-fog-continuum. However, many furtherance's are required in intelligent scheduling algorithmic and efficiency improvements. Improvement of efficient scheduling algorithms that are skilled in managing vast content of function instantiations and eliminate in brief quantities of time amongst diverse infrastructures; as a producing an edge–cloud continuum, one of the open issues presented needs to be solved for serverless computing. The significant advantage of this research is the excellent constitution of the article, encasing all correlated works together. Regardless, this article toils from some drawbacks, as follows:

- Lack of careful attention to any of the issues raised
- Absence of regard and review for the influential factors in each of the issues raised
- Instead of focusing on the central issues related to the article, it highlights trivial issues, such as how to access the articles.
- Deficiency of examples of the application of issues raised in the actual environment
- Absence of regard and review for the influential factors in each of the issues raised.

Saurav and Benedict [64] researched scheduling in scientific workflows. This study aims to present a workflow management system that applies a scheduling strategy for processes that require high scalability in the face of computational workloads and compact data. These computations are used in parallel and distributed systems. Because this system has a heterogeneous architecture, the discussion of awareness of the energy level is crucial. In this research, an attempt at the scientific workflow, along with the distinctive challenges in the path of energy-aware implementation, has been examined. On the other, they review the presented analysis of state-of-the-art workflow scheduling algorithmic regulation and the multi-target improvement issues. Moreover, they explained the importance of energy-aware runtime structures. And they finally suggested a reference architecture and runtime for energy-aware scientific workflows. Researching professionals, thorough scheduling, and reviewing corresponding main details are the influential results of this study. Regardless, the article suffers from some drawbacks, as follows:

- Inadequate coverage and weak of recently published papers at the time of publishing
- Not to pay enough attention to the future path to study the situations and challenges ahead.
- This research focuses on scheduling algorithms in large-scale systems and, considering this issue does not examine the performance of algorithms in the scalability process and only expresses generalities.
- There is no mention of how scheduling algorithms are implemented in bottlenecks. There will always be a node or nodes in the network that will act as a bottleneck. Therefore, these nodes, which are also heterogeneous, make the system service process difficult.

Shafiei et al. [65] for this survey showed a complete outline of the recent advances and the past developments in study fields associated with serverless computing. They first explored serverless applications and outlined the challenges they have fronted. Then, expansion applications in eight parts individually converse the targets and the viability of the serverless model in each of those parts. Moreover, they categorized those challenges into nine issues and explored the suggested solutions for each of them. Finally, they suggested the fields that require additional attention from the research society and determined available troubles. As a benefit, the paper is covered by many suitable, outstanding quality, recently published articles and excellent paper classification. Hence, they observed and outlined quality of experience facets that directly influence user satisfaction, which can take crucial scheduling metrics in serverless computation. Regardless, the article suffers from some drawbacks, as follows:

- Loss to review tools and implementation environments related to the subjects presented
- Despite the expression of the payment model function-as-a-service as one advantage of serverless computing, in no way has this payment model been compared to other payment models. There is no comparison between payment models in service delivery mechanisms such as infrastructure-as-a-service, platform-as-a-service, function-a-a-service, and software-as-a-service has not been made.
- In this survey, this feature has not been considered despite the expression of auto-scaling as one feature advantage of serverless computing. There are several types of scaling, and there is no mention of what scaling process is used on serverless computing.
- Given the nature of the performance and implementation of serverless computing, the expression of serverless best practices undertaking execution could have shown a better understanding of how to implement serverless computing.

Xie et al. [66], in their article, attempt attempted to present a comprehensive and organized outline of serverless edge computing networks in the aspect of networking. The first step is to study the design principles for combining a serverless model in edge computing networks. Then, the cooperation process and deployment and execution are shown in the next step. Finally, to entirely use the suggested serverless edge computing scheme, issues such as lifecycle control and service deployment, resource awareness, service scheduling, incentive instruments, exceptions, etc., are explored. Furthermore, some potential investigation subjects that should be regarded for future research are outlined. This article has many advantages, including a detailed study of the issues raised. In addition, it tries to introduce the topics that will be used as research topics in the future. Regardless, the article suffers from some drawbacks, as follows:

- The reviewed articles bank is Inadequate.
- The subjects are presented completely abstractly, and no details are noted.
- Lack of quality study articles

Cassel et al. [67] presented a comprehensive and systematic study that gives us insight into how Entities called functions are off-loaded to various devices And how these entities can interact and collaborate. Moreover, they tried to review the crucial elements utilized to set and run functions, the principal challenges, and open inquiries for this issue. In the continuation of this review, programming languages, storage services, and protocols related to the solutions are also suggested. This article has many positive points, like demonstrating an example of using this technology in the real world, study opportunities, and new-found challenges in the reciprocal hybrid of serverless and IoT, furthermore emphasizing technologies that seem promising for the subsequent years. However, providing a complete and comprehensive classification can be called the most prominent feature of this article. Regardless, the writing suffers from some drawbacks, as follows:

- Despite providing a robust classification, none of the items shown in the classification is described and presented only as a list.
- Because orchestrators are used as a crucial part of the implementation process in a server-free computing environment, this article does not refer to implementers of this technology, such as Kubernetes.
- Attention to issues such as the use of serverless computing, how to communicate across multiple domains in edge computing networks, and the challenges ahead can provide deeper insights into the capabilities of the serverless computing process. Unfortunately, this article ignores this issue.

In [68], a comprehensive study on the types of scheduling algorithms involved in serverless computing is conducted. Through examining various types of scheduling algorithms that are used in serverless computing, they attempted to provide a comprehensive overview of existing approaches and scheduling algorithms as well as their advantages and applications. Additionally, they examine various implementation frameworks that allow serverless computing to be used. There are many positive aspects in this article, such as providing a comprehensive understanding of how scheduling algorithms are applied in serverless computing. However, there are also some disadvantages:

- A lack of a comprehensive and comprehensive classification of scheduling algorithms for serverless computing
- An analysis of a limited number of articles related to scheduling algorithms
- The investigation of a limited number of parameters and metrics used in the scheduling process
- Lack of accurate and complete comparison of scheduling algorithms

According to the above articles in the field of scheduling in serverless computing, a side-by-side comparison in terms of the type of review, Objectives to be examined, Implementation layers, investigating the evaluation parameters used, and the year of publication of the reviewed articles is summarized in Table 2.

4 Research methodology

This section presented instructions for exploring relevant articles on scheduling in serverless computing. The primary segments for creating a knowledge-great review are searching, collecting, organizing, and analyzing related papers. Conducting a systematic technique that involves restricting the benchmarks of the issues and gathering and assessing those specific issues will be available to investigators. This plan describes the mechanism of discovering important issues in relevant fields.

4.1 Question formalization

The purpose of this research is to survey essential factors and techniques employed in studied articles at a particular time, along with the primary topics and challenges associated with scheduling on serverless computing. Since covering the complete examination of scheduling in serverless computing and showing related open issues is the primary purpose of the current survey, several related questions must be answered to focus on connected worries. Related research questions are shown in Table 3.

4.2 Data Analysis and papers choices

The systematic running structure of opting for and analyzing papers is outlined as follows:

- Published papers associated with scheduling in serverless computing mechanisms between 2018 and 2023

Table 2 Comparison of side-by-side review papers on scheduling in serverless computing

Resources	Type of review	Objectives to be examined	Implementation layer			Investigating the evaluation parameters							Publication
			Cloud	Fog	Edge	Throughput	Response time	Cost savings	Latency	Energy consumption	CPU usage	Deadline	
[62]	Review	Investigating the scheduling problem of cloud-based scientific workflows Providing solutions to the rescheduling problem	✔	✘	✘	✘	✔	✔	✘	✘	✘	✔	2018
[63]	Review	Investigation of the scheduling process for serverless computing in the context of the Internet of Things Investigation of the transfer process of serverless computing to the edge of the network	✘	✘	✔	✘	✔	✔	✔	✘	✘	✘	2021
[64]	Survey	Investigation of problems related to the energy consumption of scientific workflows in computing environments Examination of the energy-aware scheduling process in cloud-based applications	✔	✘	✘	✔	✘	✔	✘	✔	✔	✘	2021
[65]	Survey	Examining the scheduling challenges associated with serverless computing Assessing the reasons and goals behind service providers' migration to serverless services	✔	✘	✘	✘	✔	✔	✔	✔	✔	✔	2021
[66]	Review	Investigation of serverless edge computing networks based on scheduling Investigating critical technical challenges of serverless computing	✔	✘	✔	✘	✔	✔	✘	✔	✘	✘	2021
[67]	Review	Examining the way in which functions are loaded on different devices and how they interact with one another Incorporating serverless computing into the IoT applications	✔	✔	✔	✘	✘	✘	✔	✔	✘	✘	2021
[68]	Review	Analyzing different scheduling algorithms An examination of various serverless computing platforms	✔	✔	✘	✘	✔	✘	✘	✔	✘	✔	2023

Table 3 Research questions

Index	Questions	Descriptions
TQ1	What classification is used for scheduling in serverless computing?	The systematic scheduling classification in serverless computing could be presented as Energy-aware, Resource-aware, Package-aware, Data-aware, Deadline-aware, and Hybrid Strategies. Showing this systematic classification Causes the discovery of unconsidered fields to cover suitably. This question is answered in Sect. 6
TQ2	What performance metrics are usually applied to scheduling in serverless computing?	Studied articles assumed different metrics to perform predefined conditions. The numerous significant ones are as follows: Latency, Cost, Execution time, CPU utilization, Energy, Bandwidth, Response time, Throughput, Deadline, Memory utilization, and quality of service (QoS). Comprehending these metrics Causes us to find available crucial subjects. This question is answered in Sect. 6
TQ3	Which case studies are used in the scheduling of serverless computing approaches?	The proposed methods can pursue different goals. Some of these goals can be specific, and some can be general-purpose and used for more than one approach. Investigators can study the discovery of this benchmark to examine it appropriately in particular fields. This question is answered in Sect. 6
TQ4	What evaluation tools are utilized for scheduling in the serverless computing approach?	The studied scheduling in serverless computing articles possibly uses different assessment tools to simulate and implement the suggested strategy. Such information can be used to select the best and most valuable tools for scheduling in serverless computing. This question is answered in Sect. 6
TQ5	What programming languages are utilized for scheduling in the serverless computing approach?	The studied scheduling in serverless computing articles possibly uses diverse programming languages to implement the suggested strategy. Such knowledge can be used to select the best and most influential programming languages for scheduling in serverless computing. This question is answered in Sect. 6
TQ6	What are the future research directions and open perspectives for scheduling in the serverless computing approach?	This question attempts to explain more orientations in this field. These orientations can help researchers understand their topics of interest and use them to solve their research problems. This question is answered in Sect. 7

- To identify significant synonyms and keywords for scheduling in serverless computing, can apply the following considered string words are applied:
    1. (“Scheduling” OR “Energy-aware scheduling” OR “Resource-aware scheduling” OR “Package-aware scheduling” OR “Data-aware scheduling” OR “Deadline-aware scheduling” OR “Hybrid”) AND (“Serverless”) OR (“Serverless Computing”) OR (“Serverless Scheduling”) OR (“FaaS”) OR (“Function-as-a-Service”)).
- The search was done in October 2023, using restrictions on the time scope from 2018 to 2023. The final results showed 444 papers. In the following, by studying some crucial sections, like the Abstract, Goals, Contributions, and Conclusion, at the beginning of this process and as the first step, 180 articles unrelated to the research subject were identified and consequently eliminated. In the Next step, Because of worthless papers with inferior content, study System models, Approaches, Implementation mechanisms, Results of the research, and Solutions provided for the future in the remaining papers. In continuation of the review at the beginning, 131 papers were inscribed as unsuitable; six papers were surveyed, nine papers were repeated, and two books were left out of this process. Therefore, 148 papers have been deleted. Finally, 116 remaining journal papers were inscribed as marked relevant for the study specimen, in which 23 papers were irrigated to serverless computation and have been deleted. In selecting relevant articles, 39 articles were irrigated to scheduling and have been deleted. In the end, the remaining 54 papers related to scheduling in serverless computing approaches are included in the survey.

An illustration of the flowchart for the incorporation and elimination of options is shown in Fig. 3.

Appropriate papers associated with scheduling in serverless computing have been reviewed in reputable scientific databases, as shown in Table 4.

Also, the detailed distribution of these 54 chosen papers between 2018 and October 2023 is shown in Fig. 4. As depicted in the chart, most papers were published between 2021 and 2022 in the corresponding field with an

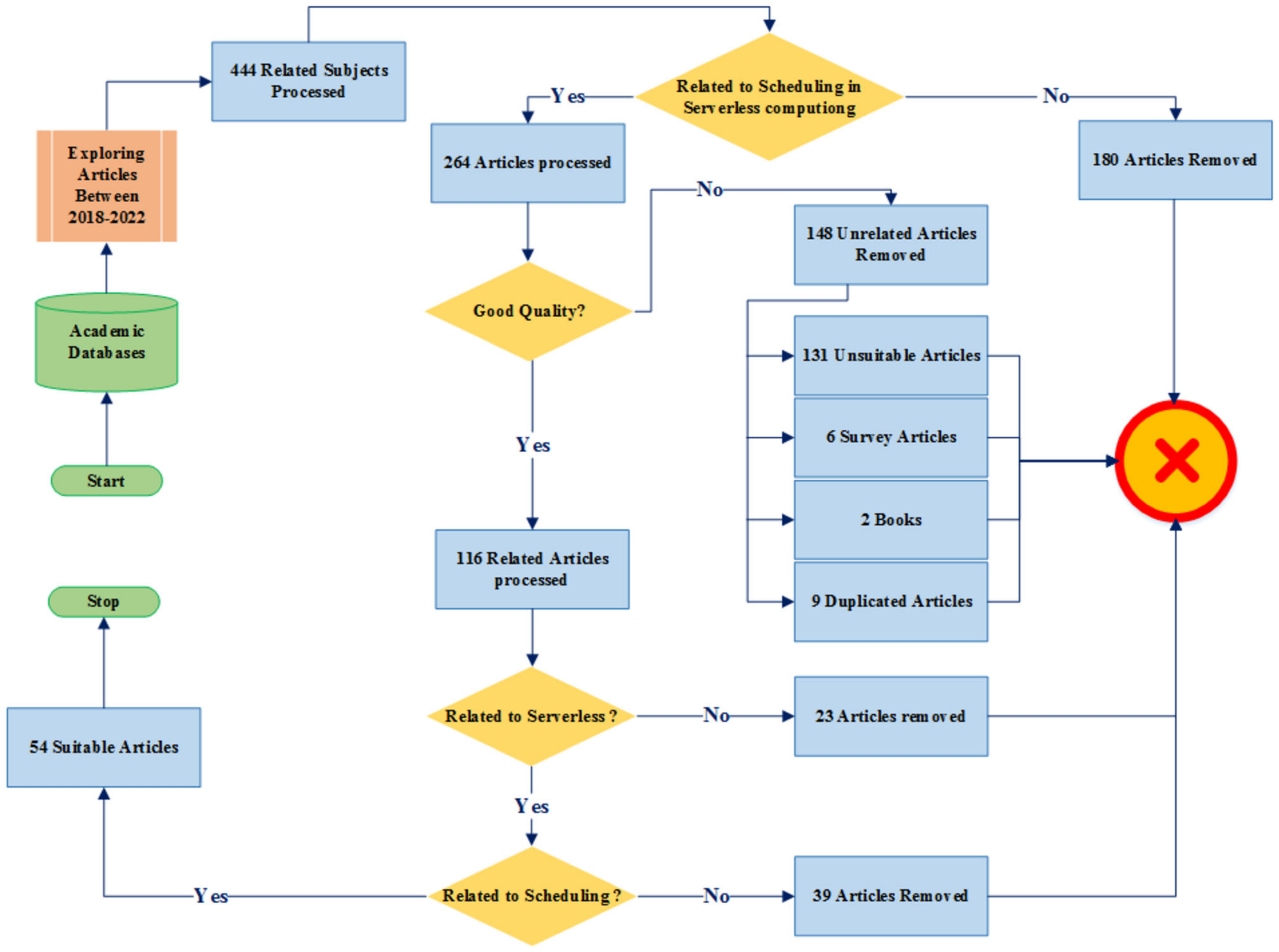

Fig. 3 Choosing criteria and evaluating frameworks

Table 4 Involved credible scholarly databases

Database	Links related to databases
Springer	http://link.springer.com
ACM	http://www.acm.org
Wiley	http://onlinelibrary.wiley.com
IEEE explorer	http://ieeexplore.ieee.org
Elsevier	https://www.elsevier.com
MDPI	https://www.mdpi.com
Other Publication	–

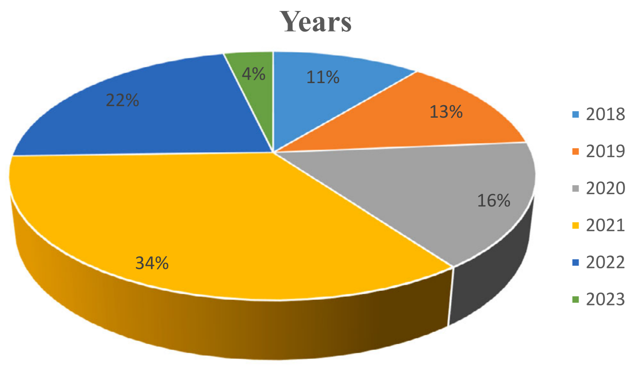

Fig. 4 Shows the percentage of investigation articles diversity with the publisher based on the publication year

accelerating interest by researchers. Over this period, 2021 has the highest percentage of published articles. In addition, as it is depicted in Fig. 5, the percentages of the mentioned papers were compared to the number of articles and publishers' titles for each year. As shown, during 2018–2023, IEEE has taken the highest position among other publishers.

5 Scheduling mechanisms in the serverless computing

Serverless computing is an event-driven mechanism in which structures called applications are illustrated by the events that trigger them. Service provider frameworks, as

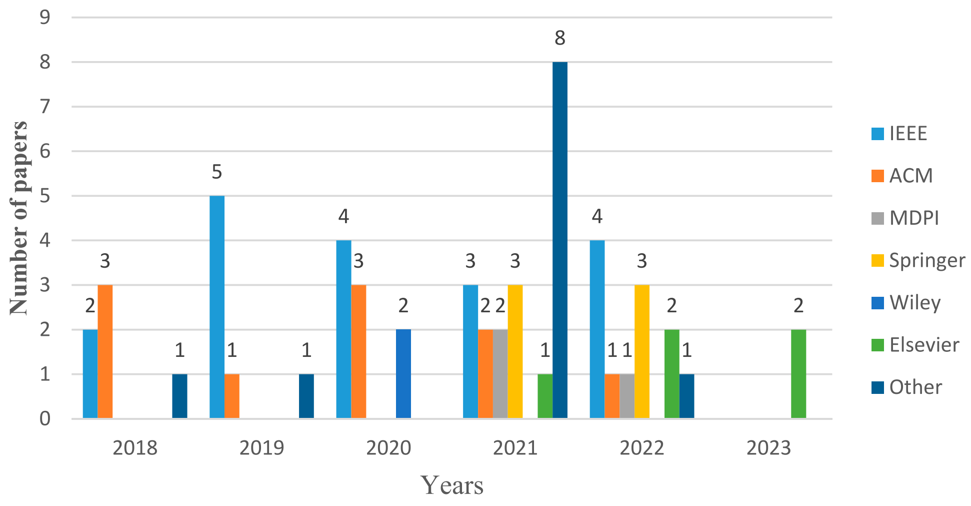

Fig. 5 Shows the percentage of investigation articles diversity with the publisher based on the years and Database

serverless platform functions, tried displaying occurrences via straightforward function abstractions and making out even acts logic among their cloud environments. This structure permits developers to turn big applications into smallish functions, permitting application parts to scale separately [69]. However, this mechanism raises fresh trouble in the logical management of an extensive collection of functions, especially in optimizing resource allocation and managing resources distributed between functions. Serverless computing, usually because of the payment model it has provided for providing its services been raised as a cost-saving tool [70]. Serverless computing has been suggested as a cost-saving mechanism because of the payment model for providing its services. One of the essential ways to save costs is the proper and optimal allocation of resources to functions [71]. Therefore, it is necessary to use a mechanism called scheduling for the optimal allocation of resources. The scheduling process is a strategic-based framework for allocating and distributing resources to devices in a network [72]. This section categorizes and studies the scheduling in the serverless computation domain for chosen papers by the suggested taxonomy. Different classifications of scheduling on serverless computing have been proposed in the literature. In this survey paper, a scheduling mechanism is used to detect the main subjects of resource allocation in serverless computing. So, a subject discovery criterion is used, recognizing six principal classifications. This measure includes issues in diverse classes with various factors and restrictions that cannot be reasonably regarded as a single issue. All techniques present in the study have been categorized into one of these six principal classes, as shown in Fig. 6. As explained earlier, scheduling algorithms are organized into six classes: energy-aware, Data-aware, Deadline-aware, Package-aware, Resource-aware, and hybrid. In the ensuing, we demonstrate some explanations associated with the kinds of scheduling in serverless computing matching the suggested taxonomy, and relevant papers will be concisely studied for each method.

5.1 Energy-aware scheduling mechanisms

This part describes related features to the energy-aware method, which is considered a serverless computing scheduling strategy. Next, the studies papers on this domain are investigated. Scheduling resources efficiently on a provider using energy-aware scheduling is possible. So that the scheduler will be able to understand how the decisions they make will impact the amount of energy consumed, enabling the scheduler to make decisions based on the amount of power utilized and the quantity of energy available [73]. The central concept behind energy-aware scheduling in cloud environments involves the use of passive structures, such as containers or cold-state environments, for the systems to consume as little energy as possible. Employing this approach finally reduces the amount of energy consumed throughout the system [74].

5.1.1 Overview of energy-aware scheduling plans in serverless computing

In this part, the various plans will be studied and then recapitulated at the end of this section. A resource scheduling technique with low energy consumption was proposed by Kallam et al. [73]; the technique is developed based on detailed investigations of a MapReduce framework's structure, operation, and energy consumption troubles associated with scheduling jobs in a heterogeneous environment. This procedure ensures that all storage nodes are efficiently checked to achieve energy conservation by developing a resource schedule for all the jobs. As a result of the proposed method, the average processing time was

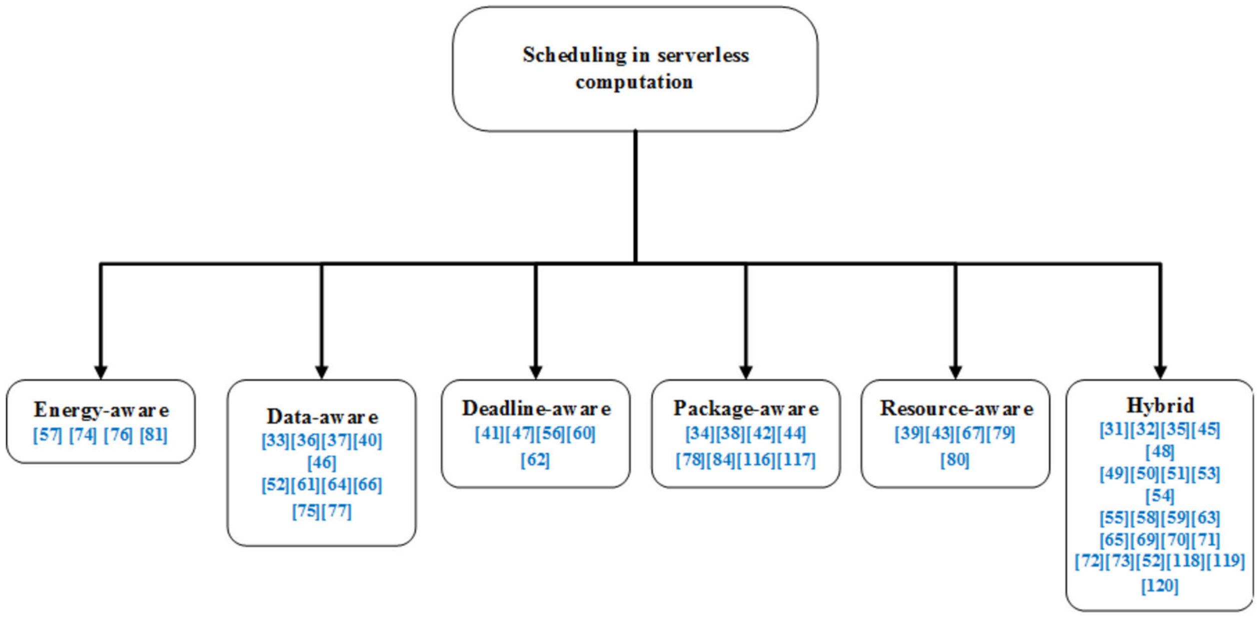
Scheduling in serverless computation
Energy-aware [57] [74] [76] [81]
Data-aware [33][36][37][40] [46] [52][61][64][66] [75][77]
Deadline-aware [41][47][56][60] [62]
Package-aware [34][38][42][44] [78][84][116][117]
Resource-aware [39][43][67][79] [80]
Hybrid [31][32][35][45] [48] [49][50][51][53] [54] [55][58][59][63] [65][69][70][71] [72][73][52][118][119] [120]

Fig. 6 Proposed taxonomy of scheduling in serverless computing

significantly reduced compared to the state-of-the-art technique.

Aslanpour et al. [74] analyzed case studies performed in the cloud with the idea that to eschew cold start functions, they should invoke functions by transmitting fake requests, which are considered warm functions. This study aims to ensure that the up-and-running approachability of edge nodes is as high as possible while ensuring that the change does not adversely affect data transmission quality or service quality.

Gunasegaram et al. [75] propose a solution to the problem of colossal container overprovisioning and microservice-agnostic scheduling, which results in poor resource usage, particularly during high workload fluctuations, thereby proposing Fifer as a solution to this issue. Moreover, Fifer minimizes overall latency by creating containers in advance, thus avoiding cold starts. In contrast to most serverless platforms, Fifer has state-of-the-art schedulers that help improve container utilization and reduce cluster-wide energy requirements.

Aslanpour et al. [76] tried to review the issue of efficiently managing energy consumption in very resource-limited edge nodes in their investigation. Their findings demonstrate that while the idle state and CPU consume the most energy in edge devices, the connection process also consumes a significant amount of energy.

Table 5 provides a comprehensive comparison of advantages, disadvantages, performance metrics, programming languages, implementation layers, and evaluation tools of the technical studies reviewed, with a focus on energy-aware scheduling algorithms and approaches.

5.2 Data-aware scheduling mechanisms

In this part, an overview of the data-aware scheduling strategy in serverless computing is provided. With its pliable and varied nature, both in terms of provisioning and allocating resources, the cloud environment poses great difficulties when managing the resources effectively, especially when dealing with data movement among multiple areas simultaneously. For cloud environments to be able to manage their resources effectively and efficiently, it is necessary to consider these points [77]. Thus, it can be concluded that by using a data-aware scheduling model (data affinity), applications can be scheduled intelligently according to the available information and using data-aware scheduling. Therefore, implementing data-aware scheduling as part of workflow applications can also result in significant performance improvements and efficient resource allocation based on the characteristics of variable workloads [78].

5.2.1 Overview of data-aware scheduling mechanisms

The various techniques are studied and summarized in this part at the end of this section.

Rausch et al. [77] propose a container scheduling mechanism enabling platforms to maximize edge infrastructure utilization. The study concluded that the trade-off between data and computational movement is crucial when implementing data-intensive functions in serverless edge computing.

Table 5 Comparison of peer-reviewed technical studies on energy-aware scheduling algorithms and approaches

References	Advantages	Disadvantage	Metrics	Programming language	Implementation layer	Evaluation tools
[73]	Reducing total energy consumption without generating scheduling overhead	Insufficient steps to implement integrated energy efficiency	Energy	NA	Cloud	Simulation
	Reduced execution time in comparison to advanced methods	Data distribution and replication dependencies are not considered	Time			
[74]	Enhancing the availability of the bottleneck node	Heterogeneous edge nodes are not investigated	Energy	Python	Edge	Implementation
	Increasing the efficiency and portability of resources	Lack of consideration of scalability challenges	Throughput			
[75]	Minimizing overall response latency	The proposed approach is not compared to similar approaches	Energy	Python	Cloud	Simulation
	Reducing energy consumption while maintaining SLOs	Using inappropriate evaluation parameters when evaluating	Latency			
[76]	Enhancing the efficiency of energy consumption	Edge computing needs are not addressed on a broader scale	Energy	Python	Edge	Implementation
	Calculating the energy consumption of the CPU, memory, storage, bandwidth, and protocol	Renewable energy sources are not considered	Cost			

Wu et al. [78] have proposed a method of on-the-fly computing that utilizes efficient multiplexing techniques to reduce the complexity of cloud programming and eliminate the complexity of data analysis by using on-the-fly data collection. Due to its on-demand execution capability, it can provide instant multitenancy and response. In addition to demonstrating the possibility of incorporating the on-the-fly computing method for processing remote sensing data into a serverless system with high reliability and performance, the authors also described how they could use the model to process data collected through sensor networks efficiently.

Yu et al. [79] propose a serverless platform of scalable and low latency called Pheromone. This platform advocates a data-centric orchestration approach in which the data flow triggers function invocations. Pheromone uses a two-level, shared-nothing scheduling hierarchy to schedule functions near the feed-in. Compared to open-source and commercial platforms, Pheromone significantly reduces latencies. Furthermore, Pheromone provides an easy way to implement many applications, including real-time querying, stream processing, and MapReduce sorting.

Das [80] demonstrated that combining Ant Colony Optimization (ACO) with a map-reduce application increases the serverless platform's efficiency. The proposed model aims to employ MapReduce on Amazon Web Services serverless infrastructure by using a scheduling algorithm called Ant Colony Optimization (ACO). According to the paper, appropriate task scheduling may be able to address the current lack of support for big data applications on serverless platforms.

Seubring et al. [81] propose a data locality-aware scheduler method for serverless edge platforms. The proposed method uses the metadata available in the edge network to optimize the scheduling of functions and constrain execution to the local network. Finally, it shows that moving the execution closer to the data reduces the bandwidth (cost) and time spent moving the data.

Jindal et al. [82] present a Function-Delivery-as-a-Service (FDaaS) framework. A FDaaS allows functions to be delivered to the target platform in a way that fits the platform's requirements. Moreover, FDaaS allows collaboration between multiple target platforms and data localization and reduces data access latency by transferring data nearer to the target platform. Finally, it shows that employing scheduling functions for an edge platform reduced overall energy consumption without violating SLO requirements.

Nestorov et al. [83] present a high-performance model capable of predicting the performance of inter-functional data exchanges using serverless workloads with a high degree of accuracy. As part of this model, parallelism, data locality, resource requirements, and scheduling policies are also considered to evaluate the performance of data-intensive workloads. The results show that deploying and scheduling workloads in this model can enhance performance.

Przybylski et al. [84] present scheduled individual calls of functions passed to a load balancer to optimize metrics related to response times. In addition, they demonstrated that employing data-driven scheduling strategies improved performance compared with the baseline FIFO or round-robin scheduling. In this way, they could adapt SEPT and SERPT strategies without experiencing any noticeable increase in computational or memory consumption.

García-López et al. [85] propos ServerMix system. The ServerMix system uses a combination of serverless and serverful components to accomplish an analytics task. In the first phase of the research, three fundamental trade-offs have been examined, including those that relate to the serverless computing model of today and their relationship with data analysis. Finally, this paper explores how ServerMix can improve overall computing performance by simplifying the effects of disaggregation, isolation, and scheduling.

HoseinyFarahabady et al. [86] demonstrate when a serverless platform is used to execute several data-intensive functional units, it face many challenges regarding workload consolidation. Finally, Performance evaluations are conducted using modern workloads and data analytic benchmark tools in their four-node platform, illustrating the efficiency of the proposed solution to mitigate the QoS violation rate for high-priority applications.

Tang and Yang [87] present a Lambdata framework for data-aware scheduling. Lambdata is a serverless computing system that allows developers to declare a cloud function's data intents, including reading and writing data. It is demonstrated in Lambda that once data intents have been defined, the Lambdata mechanism performs a variety of optimizations to improve speed, such as caching data locally and scheduling functions based on the locality of the code and the data. In a Lambdata evaluation, turnaround time has been sped up by 1.51x, and monetary costs have been reduced.

Table 6 provides a comprehensive comparison of advantages, disadvantages, performance metrics, programming languages, implementation layers, and evaluation tools of the technical studies reviewed, with a focus on data-aware scheduling algorithms and approaches.

5.3 Deadline-aware scheduling mechanisms

In this part, an outline of the deadline-aware scheduling approach in serverless computing is discussed. The use of a serverless computing approach has been found to have many advantages. Aside from these advantages, they can also be used to schedule tasks ahead of deadlines. Hence, this helps resource allocation operations run more efficiently for all request tasks since many tasks can be completed by a specific date. On the other hand, they tried to present suitable resource allocation for all tasks with deadline execution. Therefore, to maintain the efficiency of resource allocation operations, it is possible to schedule tasks using deadlines so that a sufficient number of tasks can be completed by the deadline, ensuring that operations run as efficiently as possible [88]. The scheduling process will likely be biased toward task type selection with shorter expected execution times due to the operation mechanics of this strategy. Because shorter tasks are more susceptible to implementation in this strategy, they are estimated to take a shorter time to complete, increasing their chances of success [89]. This strategy is scheduled and implemented so that meeting the latency deadline is one of its priorities. It is necessary to address this issue to minimize the possibility of many deadlines falling by the wayside due to a lack of schedule for future events [90].

5.3.1 Overview of deadline-aware scheduling mechanisms

In this part, the various approaches are reviewed and then outlined after this section. Singhvi et al. [88] present Archipelago, a serverless architecture that supports multiple tenants and performs low-latency tasks via a DAG of functions, each with a deadline for latency. Finally, they demonstrated that with this framework, they were in a position to reduce overall latency by 36X compared to the existing serverless frameworks. And with these frameworks, they could satisfy the latency specifications for more than 99% of actual workloads for applications.

Mampage et al. [90] present a policy for dynamic resource management and function assignment in applications based on serverless computing from the viewpoint of both the client and the provider. Finally, they found that a dynamic CPU-share policy outperformed a static resource allocation policy by 25% in fulfilling deadlines for the needed functions compared to a static approach.

Wang et al. [91] propose employing a model-driven approach. Their work presents a LaSS serverless platform that runs latency-sensitive computations at the edge. Finally, shown by their simulations, LaSS can be programmed to re-provision container capabilities as rapidly as possible while preserving fair share distributions based

Table 6 Comparison of peer-reviewed technical studies on data-aware scheduling algorithms and approaches

References	Advantages	Disadvantage	Metrics	Programming language	Implementation layer	Evaluation tools
[77]	Improved job placement quality as compared to advanced Kubernetes scheduling	Inadequate consideration of the dynamic nature of edge systems	Throughput	Python	Cloud	Simulation
	Establishing accurate timing parameters to achieve goals	A lack of context-aware scalability	Time		Edge	
[78]	Increasing the speed of response	Inattention to differences in execution times between nodes	Latency	Python	Cloud	Implementation
	Optimizing the reading and conversion of data structures	Evaluating the project without taking into account essential parameters, such as the cost of implementation	Time			
[79]	Reducing performance interaction delays during data exchange	Disregarding the energy consumed in storing and retrieving data	Latency	Python	Cloud	Implementation
	The reduction of real-time querying and execution of a variety of applications	Lack of use of history-aware approaches to reduce event response times	Throughput	C++		
[80]	Reducing the overhead caused by data storage	The scheduling function provided is not optimal as compared to similar approaches	Cost	Python	Cloud	Implementation
	Faster storage of data	Inattention to the process of storing big data in a serverless environment	Time			
[81]	Reduced time spent moving data	An evaluation of different workloads has not been conducted	Throughput	Python	Edge	Simulation
	Providing high throughput with minimal resource consumption	Loading strategies aren't incorporated into schedulers	Cost			
[82]	Implementation of the process of data localization	Lack of investigation into the performance of the proposed approach on different types of data	Energy	Python	Cloud	Implementation
	Reducing the consumption of energy in general	Incorporating AWS Lambda into the proposed approach provides better performance	Time		Edge	
[83]	Providing a solution to the problem of data sharing	Automating the configuration of work details can improve the performance	Latency	NA	Cloud	Simulation
	Deploying intensive workloads more quickly	Data-intensive workloads are not used in the evaluation process	Time			
[84]	Enhancing the performance of data-driven scheduling decisions	Increasing workload pressure on the system results in reduced performance of the proposed approach	Latency	C++	Cloud	Simulation
	Reducing the response time to requests		Time			
[85]	Improve computing performance by reducing separation processes and data separations	A lack of performance in the face of big data	Latency	Python	Cloud	Implementation
	Assisting with the scheduling process for a variety of programs	The algorithm does not perform well when applied to real data sets	Cost			

Table 6 (continued)

References	Advantages	Disadvantage	Metrics	Programming language	Implementation layer	Evaluation tools
[86]	Enhance the total throughput	The proposed approach has not been tested with a variety of modern workloads	Throughput	Java	Cloud	Implementation
	A reduction in the number of violations of QoS for applications	Failure to evaluate the performance of the proposed approach using important evaluation parameters	Time			
[87]	Enhancing the scheduling process' speed	The proposed approach has not been tested against a variety of workloads in order to determine its performance	Cost	Python	Cloud	Simulation
	Reducing the cost of operations		Time			

on their simulations. A LaSS can also accurately predict the resources required for serverless functions when workloads are highly dynamic.

Rama Krishna et al. [92] suggested a SledgeEDF. They used the extension of real-time scheduling to a serverless runtime. For this reason, Sled EDF was suggested to pursue advanced research in this area. SledgeEDF improved the latency of the server-free function by introducing a scheduler for handling a large volume of mixed high-priority workloads simultaneously. As a result of implementing an admissions controller and fine-tuning module configurations, the runtime was able to satisfy 100% of the deadlines.

Pawel and Rzadca [93] present a framework algorithm that can be applied to cloud-based applications on the FaaS system. They also made some desirable modifications to the architecture. Finally, it provides successors for each task in the queue, enabling the scheduler to know which environments must be configured in advance. Based on the simulation results, it was evident that these methods can enhance the efficiency of FaaS. Also, scheduling considering constraints and startup times is more effective and efficient when the system is under heavy load.

Table 7 provides a comprehensive comparison of advantages, disadvantages, performance metrics, programming languages, implementation layers, and evaluation tools of the technical studies reviewed, with a focus on deadline -aware scheduling algorithms and approaches.

5.4 Package-aware scheduling mechanisms

This part provides an introduction to the package-aware scheduling strategy used in serverless computing. In serverless computing, functions are run in pre-allocated containers or VMs on pre-allocated servers. These applications can operate in pre-allocated containers, allowing them to begin operations quickly and efficiently. The functions in question require one or more separate packages to function correctly. These packages can be separated or related to ensure their proper functioning because function execution mechanisms depend entirely on an entity known as a package to function correctly. These functions can start a little slower when needed on a large package [94]. One of the computational nodes in serverless technologies is responsible for retrieving required packages and application libraries upon invocation requests. Finally, if the computation node receives a request for their invocation, a try is required, and the mentioned node will retrieve related library packages. On one or more worker nodes, there is a possibility of installing a preloaded package to make it possible to reuse the execution environment of one or more worker nodes over another worker node. By utilizing this opportunity, it has been possible to reduce the start-up time for the tasks and the turnaround time to complete them, thereby allowing them to be completed in a shorter amount of time and becoming more efficient. In addition to predicting future library requirements based on the libraries already available and installed, it is also possible to use this scheduling strategy to predict future library needs [95].

5.4.1 Overview of package-aware scheduling mechanisms

In this part, the variety of strategies is analyzed and summarized after this section. Dayong and D He [94] present an approach that employs a scheduling algorithm to complete the scheduling process despite the problems encountered with pod-by-pod scheduling. Finally, they demonstrated that the scheduling process with the new algorithm is significantly more efficient compared to pod-by-pod scheduling in terms of pod start-up latency, and pod start-up latency has dramatically decreased compared to pod-by-pod scheduling.

Table 7 Comparison of peer-reviewed technical studies on deadline -aware scheduling algorithms and approaches

References	Advantages	Disadvantage	Metrics	Programming language	Implementation layer	Evaluation tools
[88]	Reducing the execution time of requests	Lack of consideration for the system's performance under a variety of workloads	Latency	Python	Cloud	Implementation
	Reducing overhead expenses	The proposed approach has not been tested in systems with non-homogeneous members	Time			
[90]	Response time to requests has been improved	A lack of dynamic implementation of the resource management process	Cost	NA	Cloud	Simulation
	Reducing the consumption of resources	The efficiencies of the proposed approach in heterogeneous cloud environments have not been evaluated	Time			
[91]	Reducing the time required for resource allocation	Only Poisson entry and service processes are considered in this approach	Latency	Python	Edge	Implementation
	Resources are allocated in an optimal manner	Failure to consider the composition of serverless functions when making scheduling	CPU	JavaScript		
[92]	Utilization of resources in the most efficient manner	Low latency is not guaranteed in edge systems	Throughput	C++	Cloud	Implementation
	Providing real-time admissions management		Latency		Edge	
[93]	Providing superior performance over other scheduling algorithms	Failure to test and verify workflows on parallel machines	Time	NA	Cloud	Simulation
	Reduction of response latency by a significant amount	Ignoring important parameters	Latency			

T Gustavo et al. [95] raise the issue of the vanilla load balancers used by FaaS schedulers that do not strive to reduce package and file transmissions. Finally, to demonstrate the utility of the proposed approach, they have included a package-aware scheduling algorithm for the scheduler.

Gabriel et al. [96] undertook a research effort that focused on problems that the existing FaaS schedulers faced. The proposed scheduler PASch using the OpenLambda framework was implemented and evaluated through real-world experiments and simulations. Finally, results from the evaluations of the proposed PASch have demonstrated that it provides a 1.29 $\times$ speedup over the least loaded balancer and a 23 $\times$ reduction in 80th percentile latency.

B Ting et al. [97] present a novel Attribute-aware Neural Network Approach (ANAM) to cope with scenarios where a client's desire for the selected item is not explicitly tracked. Finally, their findings concluded that the purpose ANAM approach is an effective tool for recommending the basket base for the future.

Chetabi et al. [98] have proposed reinforcement learning algorithms, SARSA and SFSchlr. SFSchlr is a function scheduling algorithm that can be used in a function as a form service environment. SFSchlr, which they introduced, executes the learning operation online and is aware of the data dependencies. They have implemented and evaluated SFSchlr with two scheduling algorithms as well as a new dependency-aware scheduler, demonstrating that the proposed algorithm improves function turnaround times by up to 58% and resource utilization by up to 69.5%.

Ebrahimpour et al. [99] have focused on timing issues and cold starts. Keeping operating environments warm can decrease cold start times but increase costs. In this paper, the authors propose an approach to creating a trade-off using four different types of decision-making at runtime based on a heuristic method and analyzing function dependency graphs, function call frequencies, and other environmental variables. Compared to the constant time method (that is, the method used by Amazon), the proposed method shows a 32% improvement.

Table 8 provides a comprehensive comparison of advantages, disadvantages, performance metrics, programming languages, implementation layers, and evaluation tools of the technical studies reviewed, with a focus on package -aware scheduling algorithms and approaches.

5.5 Resource-aware scheduling mechanisms

The client in a serverless environment sends a request to the service provider as an event when they request the service from the provider. Hence, the service provider is not only obliged to respond to these requests but also must allocate the resources necessary to process them. It is imperative to note that one of the most critical factors that need to be considered when dealing with these requests is the basis on which the provider responds to them. In other words, it is crucial to consider how resources are allocated from the service provider to the consumer. Hence, a provider uses a scheduling mechanism to allocate resources to requests received to accomplish this task [100]. There are several different kinds of serverless applications, and the way these applications use resources varies greatly. These are two examples of resource allocation events in research computing. There is a possibility that several CPU-consuming functions might be affected by a slowdown in performance because when they are co-located on one physical node, they are in direct competition for resources. Therefore, if these functions are located on one physical node, resource contention may result in delays in the execution of invoked functions [101].

5.5.1 Overview of resource-aware scheduling mechanisms

Different strategies will be discussed and summarized in the following section. Suresh and Gandhi [100] present an approach for scheduling functions at the function level that minimizes the cost of provisioning resources while meeting the performance requirements of customers by incorporating resource-aware scheduling. Based on the study's results, they found that compared to existing baselines, the approach significantly improved resource efficiency by between 36 and 55% while ensuring that application latency remains acceptable, in addition to this significant increase in resource efficiency.

Yuvaraj et al. [101] propose a machine-learning model that enables the parallel processing of the tasks assigned to the queue of events and the serverless service's dispatcher framework to identify the most efficient way to dispatch the tasks. The simulation results show that the proposed GWO-RIL can reduce running time and adapt to various loads under various conditions based on the simulation results.

Cheng and Zhou [102] present a framework whereby streaming applications can be automatically scheduled and allocated resources based on their needs through an ARS (FaaS) service. In light of the results of this study, a more flexible and efficient scheduling method has been developed, enabling resources to be made autonomously available on demand and dealing with fluctuations in streaming data and unforeseen conditions in real-time through a flexible and efficient approach.

Lakhan et al. [103] present a novel, cost-effective, and stable framework for the IoT using a blockchain-enabled fog cloud infrastructure. Based on the simulation results, the proposed algorithms are superior to all existing baseline strategies regarding performance when implementing the applications compared to existing baseline approaches.

Kim et al. [104] developed a fine-grained CPU cap management approach incorporated into an intelligent resource manager that continuously adjusts the CPU usage limits (or CPU caps) for different applications with exact/similar performance requirements. Lastly, their research has shown that resource managers outperform other heuristics in average response time and skewness by decreasing them by 94 and 44%, respectively, while not overusing the CPU resources.

Table 9 provides a comprehensive comparison of advantages, disadvantages, performance metrics, programming languages, implementation layers, and evaluation tools of the technical studies reviewed, with a focus on resource -aware scheduling algorithms and approaches.

5.6 Hybrid

Due to the limited number of articles and studies related to the scheduling process in serverless computing, in the previous articles, we tried to perform the scheduling process, which has more studies, in the form of separate and specialized taxonomies to be analyzed. Therefore, in this section, some studies examine the types of timings whose number does not reach more than one or two. We may even find articles in which scholars have tried to Use a combination of timings. There are very few of them, so it has been decided to ensure they are placed in a section under the term hybrid. Hence, this taxonomy section will examine schedulers of various types.

5.6.1 Hybrid scheduling mechanisms

A variety of approaches will be examined and analyzed in the following section. Patterson et al. [105] present

Table 8 Comparison of peer-reviewed technical studies on package -aware scheduling algorithms and approaches

References	Advantages	Disadvantage	Metrics	Programming language	Implementation layer	Evaluation tools
[94]	Reduced latency caused by pod launch	The proposed approach in the public cloud platform has not been checked	Latency	NA	Cloud	Simulation
	Allocation of resources in the most efficient manner	The proposed approach has not been tested with different workloads	Time			
[95]	Optimization of the scheduling process close to the data	The proposed approach has not been compared with other scheduling algorithms	Latency	Go	Cloud	Simulation
	Allocation of resources in the most efficient manner	Evaluating without considering important parameters	Time			
[96]	Reducing the time it takes to respond to requests	The proposed approach has not been evaluated with a variety of dynamic workloads	Latency	Python	Cloud	Simulation
	Optimizing performance and providing better results	Failure to consider different balancing parameters	Cost			
[97]	Resource optimization for consumption	Failure to investigate the effect of multiple factors on the decision-making process	Time	NA	Cloud	Simulation
	Increasing the speed of response to requests	A lack of evaluation of the proposed approach against a variety of workloads	CPU			
[98]	Increasing resource availability and reducing overall costs	The proposed approach has not been tested with different loadings	Time	Python	Cloud	Implementation
	Increasing the efficiency of resource utilization	The evaluation process does not take into account important parameters	CPU			
[99]	Reducing the time it takes to respond to requests	The proposed scheduler can be improved by applying deep-learning algorithms	Time	JavaScript	Cloud	Simulation
	Implementing requests at a lower cost	Failure to select the appropriate evaluation parameters	Cost			

HiveMind, the first distributed coordination framework that allows users to execute sophisticated task workflows between edge and cloud resources in a fast, scalable, and programmable way. The researchers demonstrated that HiveMind is significantly better suited for performance prediction and battery performance than existing decentralized and centralized technologies while incurring significantly less network traffic than existing decentralized and centralized technologies.

Soltani et al. [106] propose an approach that consists of designing a migration-based distributed mechanism for executing long-term Serverless functions in a distributed environment. Using this approach, one can continuously transfer a function from one cloud platform to another whenever that function approaches its maximum duration limit. To conclude, they created a case study illustrating how the proposed approach can be applied with the help of a generic application for machine learning built on top of the scientific framework ANTDROID.

Zhang et al. [107] present a new serverless approach that uses the interaction between the client and the fog cloud to provide better service when analyzing video using DNNs by taking advantage of the benefits of serverless architecture. Finally, they showed that VPaaS had been thoroughly tested on many standard video datasets, concluding that it is more accurate than other available solutions. Maintaining high accuracy while decreasing radio transmission times by up to 62.5%, cloud monetary costs by up to 50%, and bandwidth consumption by up to 21% is the key to reducing bandwidth usage, radio transmission times, and cloud monetary costs.

Table 9 Comparison of peer-reviewed technical studies on resource-aware scheduling algorithms and approaches

References	Advantages	Disadvantage	Metrics	Programming language	Implementation layer	Evaluation tools
[100]	Utilizing resources in a more efficient manner	Disregarding the difference in the execution time of different functions	CPU	Python	Cloud	Implementation
	Responding to requests more quickly	Failure to test the proposed approach with a variety of workloads	Time		Edge	
[101]	Reducing energy consumption	Using deep learning techniques can enhance the performance of the proposed method	Energy	Java	Cloud	Simulation
	Reducing the time it takes to respond to requests		CPU			
[102]	Providing high-performance HPC Cloud resources	Evaluations that fail to take into account important parameters	Energy	NA	Cloud	Simulation
	Responding to requests more quickly	Lack of comparison between the proposed approach and other vital approaches	Time			
[103]	Using blockchain technology to ensure the security of the system	Failure to ensure that the proposed approach is applicable in a variety of contexts	Cost	Java	Cloud	Implementation
	Improve the efficiency of resource allocation	A lack of accuracy in the results obtained	Time	Python	Fog	
[104]	Reduced skewness and average response time	Resources such as CPUs have not been considered in the proposed approach	CPU	Python	Cloud	Implementation
	Utilization of CPU resources in the most efficient manner	The proposed approach has not been tested on a variety of workloads	Time			

Gadepalli et al. [108] present a new Wasm-based framework called aWsm that allows serverless computing at the edge using Wasm. Lastly, they present a preliminary assessment of the performance and reliability of aWsm using a set of MiBench micro benchmarks as functions, characterized by the use of a small amount of memory (ranging from 10 to 100 bytes) and an average startup time (12 s to 30 s) for a sample of these benchmarks.

Fard et al. [109] present an approach that measures microservices' memory and CPU requirements, regardless of whether a micro-service runs on a private cluster or an enterprise cloud. Based on the results of the experiments, they found that the proposed method can provide a very high level of scalability and simultaneously increase the utilization of both the memory and the CPU, leading to a higher throughput when compared to the standard system.

Zhang et al. [110] propose a system for deploying and offloading IoT applications that combine serverless technology with teleoperability, aiming to expand the serverless model. Finally, they concluded that the STOIC approach reduced the execution time (response latency) and resulted in 92% to 97% placement accuracy with a very low response latency.

Aytekin and Johansson [111] present an implementation of a parallel resource allocation method for solving the problem of the limitations and performance of serverless runtimes. Finally, they have shown that they can achieve relative speedups of up to 256 workers with efficiencies of over 70%, even with as few as 64 workers.

Huang et al. [112] present a robust joint cloud framework for IoT systems by utilizing a serverless model called HCloud, which uses trusted collaborative clouds. Finally, the evaluation results found that HCloud could significantly enhance the performance of serverless workloads at negligible costs compared to other options.

Zhang et al. [113] propose the HyperFaaS framework to minimize the overhead associated with implementing ICS and reduce the costs associated with its performance. To achieve this goal, they tried to use request streaming in this work and to pipe data directly from the worker to the server without parsing any HTTP headers to accomplish this task. To evaluate this framework, in the first implementation of the software, they tried using only the memory and CPU resources will be analyzed to evaluate the software.

Denninnart et al. [114] present a method to guarantee that the implementations of serverless computing can be robustly used when using HC systems, especially if the HC systems are overcrowded at the time of deployment. Following applying the pruning mechanism to several variants of homogeneous and heterogeneous computing systems, a substantial improvement in the system's robustness has been observed. The assessment results indicate that the system's robustness has improved (up to 35%).

Ling et al. [115] proposed a new framework for the operation of devices and the edge of cloud environments in the telecommunication industry called Lite-Service. Finally, the empirical results have shown that the Lite-Service framework is efficient for scheduling and building telecom applications on the device, the edge, and the cloud, based on their empirical results.

Silab et al. [116] argue that by providing multi-step decision-making processes, machines can allocate tasks to achieve their goals as effectively as possible, reducing average response times to users. Finally, the evaluation showed that it is possible to increase the speed of the decision-making process by nearly 21% using the proposed cache-assisted serverless architecture.

Tychalas and Karatza [117] discuss how load-balancing methods can decrease the operational costs of a system and introduce a new method (SaMW) that reduces expenses while maintaining a relatively low Mean Response Time while lowering the operating costs of the system. They also showed reduced total expenses and network traffic. In the Low Load scenario, the Utilization of Remote Fast Containers is reduced to 1%, while the Utilization of Slow Remote Containers is reduced to 2%.

Mujezinović and Ljubović [118] present a more integrated approach for achieving cloud-like functionality that can improve upon existing techniques that are efficient and based around cloud computing. Using the AWS Fargate technology, they proposed an approach based on a producer–consumer model that can meet a wide range of needs. Finally, they demonstrated that this concept could acquire high-frequency data in a system.

Denninnart and Salehi [119] present SMSE, the first serverless platform explicitly designed to deliver multimedia streaming services, because they believe multimedia streaming to be one of the most popular applications in the information technology field. Finally, implementing SMSE prototypes in real-world settings demonstrated that SMSE could reduce containerization overhead and make time (up to 30%) when providing multimedia functionality, compared to general-purpose serverless cloud platforms that provide function provisioning methods.

Ao et al. [120] describe how the Sprocket system can be used on the AWS Lambda serverless cloud platform by explaining how it was designed and implemented. They demonstrated that, under various conditions, Sprocket meets its performance goals of high parallelism, low latency, and low cost (10 s to process a 3600-s video 1000-way parallel for less than $3).

Wen et al. [121] propose StepConf, a platform for automating the configuration of resources for functions based on workflow execution. Finally, they examined StepConf's performance on AWS Lambda and found that when compared with baselines, StepConf saved 40.9% in costs while meeting SLOs.

Nesen and Bhargava [122] propose a framework that allows data from multimodal sources to be processed using a serverless computing infrastructure to extract features and patterns from the data. They concluded that serverless approaches might be advantageous when workloads are uneven and unpredictable because they outsource scalability issues to a third party while keeping costs low.

Wu et al. [123] propose multisite transactional causal consistency (MTCC) protocols for decreasing the latency of IO operations and guaranteeing application-wide consistency in serverless Function-as-a-Service (FaaS) architectures. Finally, they demonstrate orders of magnitude improvements in performance thanks to caching while protecting against consistency anomalies that may arise in the future.

Carver et al. [124] propose Wukong, a novel serverless parallel computing approach that combines locality-enhanced, decentralized scheduling (based on Amazon Lambda), delayed I/O, and task clustering to provide high efficiency, data locality, and optimal scalability while remaining cost-effective and efficient. They have demonstrated that Wukong exhibits optimal scaling behavior, decreases running time by up to 98.53% compared to numpywren, and significantly reduces network I/O by order of magnitude.

Tang et al. [125] present a partially observable stochastic game (POSG) for task scheduling competition that is proposed to provide serverless edge computing nodes to and non-cooperatively schedule jobs based on the locally observed state of the system. Edge-computing heterogeneous nodes in the network are rational individuals interested in optimizing their scheduling utility but are limited to using local observations. Also, with the help of the dual-depth recurrent Q-network (D3RQN) approach, they developed a multi-agent job scheduling algorithm that approximates the optimal solution to the proposed partial observability problem.

De Palma et al. [126] presented an expressive language for specifying serverless scheduling policies that expressed constraints on the topologies of schedulers and processing nodes. The authors implemented their approach as an extension to the OpenWhisk framework, and by implementing relevant scenarios, they demonstrated that their extension is on par with vanilla OpenWhisk and sometimes performs better.

Lakhan et al. [127] developed a serverless and bound-based Boltzmann machine algorithmic framework for mobility-aware security dynamic service composition in healthcare workflows. According to this study, the deep stochastic neural network trains probabilistic representations at every stage of the workflow task sequencing, including service composition, scheduling, and security. In terms of safety and application cost, the system-based methods developed outperformed traditional methods by 25% and 35%, respectively.

Table 10 provides a comprehensive comparison of advantages, disadvantages, performance metrics, programming languages, implementation layers, and evaluation tools of the technical studies reviewed, with a focus on hybrid scheduling algorithms and approaches.

6 Discussion

This section demonstrates a discussion and rational assessment of the existing scheduling strategies in serverless computing. The rational investigation and reports are based on the present TQs in Sect. 4:

- **TQ1** What classification is used for scheduling in serverless computing?

  Based on the suggested taxonomy, an actuarial comparison between scheduling strategies in serverless computing is illustrated in Figure 7. Six strategy cases are considered according to the represented taxonomy for scheduling in serverless computing. The systematic classification of scheduling in serverless computing could be presented as Energy-aware, Resource-aware, Package-aware, Data-aware, Deadline-aware, and Hybrid Strategies take 8%, 10%, 8%, 22%, 10% and 42% coverage, respectively. As depicted in Figure 7, most of the investigation chosen papers are related to the Hybrid strategy, with a percentage coverage of 42 in the literature. But if we want to consider a specific strategy, data-aware scheduling has been chosen as the most widely used specific strategy for research by covering 2 of the considered studies.
- **TQ2** What performance metrics are usually applied to scheduling in serverless computing?

  As the performance metrics for scheduling strategies in serverless computing, some characteristics are investigated and accommodated in Fig. 8. It is necessary to mention that because some investigation papers were multi-goals, some metrics may be observed in more than one paper. The investigation of these characteristics illustrates that latency has the most usage on scheduling strategies in serverless computing with 27% and then energy with 20% in the second rank; Throughput, Response time, Cost, and CPU with 16%, 14%, 12% and 11% rank, respectively, the next places in the chart. Therefore, many characteristics like Cost and CPU with low coverage attention are considered open challenges in scheduling strategies in serverless computing.
- **TQ3** Which case are studies used to schedule serverless computing approaches?

  The case studies of the scheduling strategies in serverless computing are shown in Fig. 9. Following is a collection of existing case studies that were employed as examples of the application of scheduling in serverless computing. This collection consists of General applications, vehicular networks, Machine learning, Stream Processing, IOT, Anomaly Detection, Multimedia Processing, Healthcare, Farm, Face Detection, and Telecom. The methods presented can be used to achieve a variety of objectives. Some goals can be specific, while others can be more general and used for various purposes. As a result, some papers have included more than one case study in the context of their research. Most of the studies selected in the literature utilized General Application for their case studies, with a percentage coverage of 52 of the literature. Nevertheless, if we want to consider a specific case study, Multimedia processing has been chosen as the most widely used case study, with 15% of the studies considered. Particular technologies like smart farms, healthcare, machine learning, telecommunications, face detection, anomaly detection, and stream processing do not attract as much attention as others.
- **TQ4** What evaluation tools are utilized for scheduling in the serverless computing approach?

  According to Figure 10, 38% of study papers used the AWS tool. Also, 26% of the research articles have not determined tools for their proposed model. In addition, OpenWhisk, OpenFaaS, Kubernetes, and Azure tools with 16%, 12%, 6%, and 2% rank, respectively, the following places in the chart.
- **TQ5** What programming languages are utilized for scheduling in the serverless computing approach?

  Several programming languages are employed to implement scheduling strategies in serverless computing, as shown in Fig. 11. There is also a need to point out that some of the research papers were multi-tool implemented, which means that some research used more than one programming language. In investigating these programming languages, it was revealed that

Table 10 Comparison of peer-reviewed technical studies on hybrid scheduling algorithms and approaches

References	Advantages	Disadvantage	Metrics	Programming language	Implementation layer	Evaluation tools
[105]	Optimizing the consumption of energy	Failure to address the cold start issue	Energy	Python	Edge	Simulation
	Support for a large number of edge devices without degrading performance	The evaluation process was conducted without consideration of the important factors	Throughput	C++		
[106]	Reducing the response time through a migration approach	The lack of consideration for a fairer distribution of functions among nodes than just random selection	Energy	Java	Cloud	Simulation
	Responding to requests with less delay		Time			
[107]	Reducing bandwidth consumption	Topology not suitable for the proposed approach	Throughput	Python	Cloud	Simulation
	Reducing the cost of operations	Performing analyses without considering the importance of privacy	Time		Fog	
[108]	Introducing a new approach to serverless management at the edge	The lack of a profile module to track and analyze the performance of functions	Latency	Javascript	Cloud	Implementation
	A reduction in the average startup time	Lack of experimentation with different scheduling strategies	Time			
[109]	Providing a scalable mechanism that utilizes both memory and CPU at the same time	Inability to scale microservices in the cloud using the proposed approach	CPU	Go	Cloud	Simulation
	Reducing implementation complexity	Failure to evaluate the performance of the proposed approach in light of various workloads	Cost			
[110]	Reduce overall execution time	Inability to implement the proposed approach on heterogeneous hardware	CPU	Go	Cloud	Implementation
	Keeping the system responsive when excessive requests are received	Failure to test the performance of the proposed solution with different workloads	Latency		Edge	
[111]	Support for intensive calculations	Inaccurate calculation of execution times	Time	NA	Cloud	Implementation
	Increasing the speed at which requests are responded to	A lack of incoming network connections during runtime	Energy			
[112]	Optimizing the process of resource consumption	The failure to use important parameters in the evaluation process	Throughput	Python	Cloud	Simulation
	Reducing the costs associated with responding to requests		Cost			
[113]	Optimizing the use of consumption resources such as CPU and memory	The cold start challenge is not taken into consideration	CPU	NA	Cloud	Simulation
	Responding to requests at a lower cost	The proposed approach has not been tested with different workloads	Cost			

Table 10 (continued)

References	Advantages	Disadvantage	Metrics	Programming language	Implementation layer	Evaluation tools
[114]	An implementation of the proposed approach on heterogeneous and homogeneous computing systems	Energy and cost parameters are not examined in the proposed approach	Time	NA	Cloud	Simulation
	Significant improvement in system robustness		Throughput			
[115]	Reduced execution time despite changes in load	Failure to investigate a reservation-based decentralized resource management process that can be much more efficient	Latency	C++	Cloud	Simulation
	Increasing performance and adapting to different scenarios		Throughput		Edge	
[116]	Improve the average response time	Using intelligent schedulers can reduce some of the dependencies between functions	Latency	Python	Cloud	Simulation
	Making decisions in a more efficient manner	Using machine learning techniques can improve the current decision-making process	Time	Go		
[117]	Reduce the cost of operations	In the Microservices paradigm, heterogeneous computing systems can reduce energy consumption and monthly operating expenses	Energy	NA	Cloud	Simulation
	Reducing the average response time		Cost			
[118]	Optimization of the allocation pattern	Not paying attention to topics related to automatic deployment	Time	Python	Cloud	Implementation
	Enhancing the scalability of the system	Tests of the proposed approach with different workloads were not conducted	CPU			
[119]	Reduced overhead and construction time associated with containerization	Continuity of edge-to-cloud approach is not verified	Latency	NA	Cloud	Implementation
	Enhancements to multimedia processing	Failure to incorporate important evaluation parameters into the evaluation process	Cost			
[120]	Improving the response time of the system to requests	Using process parallelization in this approach enables applications to be launched on-demand with minimal startup delay	Latency	Python	Cloud	Implementation
	Reducing the costs associated with responding to requests		Cost			
[121]	Improving system performance in the process of responding to requests	The cold start challenge is not taken into consideration	Energy	Python	Cloud	Implementation
	Workflow optimization	The proposed approach has not been tested with different workloads	Cost			

Table 10 (continued)

References	Advantages	Disadvantage	Metrics	Programming language	Implementation layer	Evaluation tools
[122]	Optimizing the system's cost and speed	The proposed approach does not utilize a serverless recommendation mechanism	Time	NA	Cloud	Implementation
	Providing the ability to process data from a variety of sources	A lack of consideration of important evaluation parameters in the proposed approach	Cost			
[123]	Increasing the speed of response to requests	Not paying attention to topics related to automatic deployment	Time	Python	Cloud	Simulation
	Increasing the efficiency of resource allocation		Latency			
[124]	Reducing the execution time	Inability to scale microservices in the cloud using the proposed approach	Time	Python	Cloud	Implementation
	Reducing the costs associated with responding to requests		Throughput			
[125]	Utilization of deep learning technology to enhance the proposed approach	The proposed approach has not been tested with different workloads	Energy	NA	Edge	Simulation
	Optimization of resource allocation	Inaccurate calculation of execution times	CPU			
[126]	Assuring an equitable allocation of resources	Utilizing technologies such as deep learning can optimize the scheduling performance and automate decision-making	Latency	JavaScript	Cloud	Implementation
	Reducing the time it takes to respond to requests		Time		Edge	
[127]	Reducing the costs associated with responding to requests	Uncertainty regarding the use of resources	CPU	Java	Edge	Simulation
	Increasing the safety of the proposed approach	The proposed approach is not suitable for use in dynamic environments	Time			

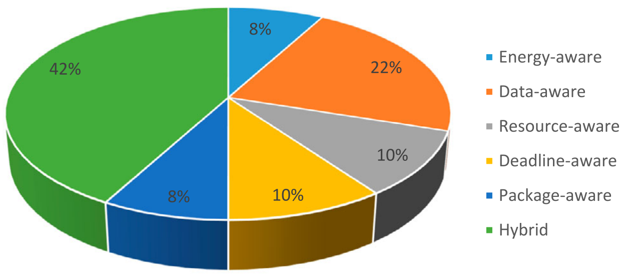

Fig. 7 Classification of scheduling in serverless computing

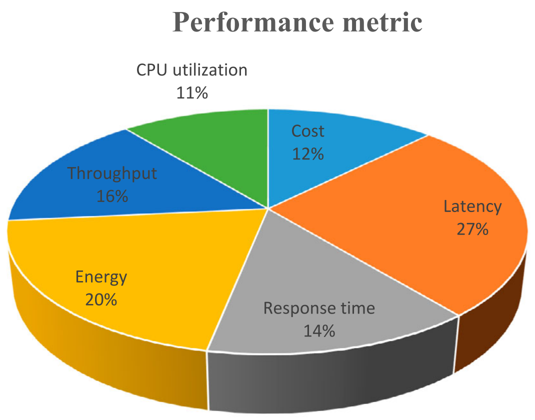

Fig. 8 Performance metrics of the scheduling strategies in serverless computing

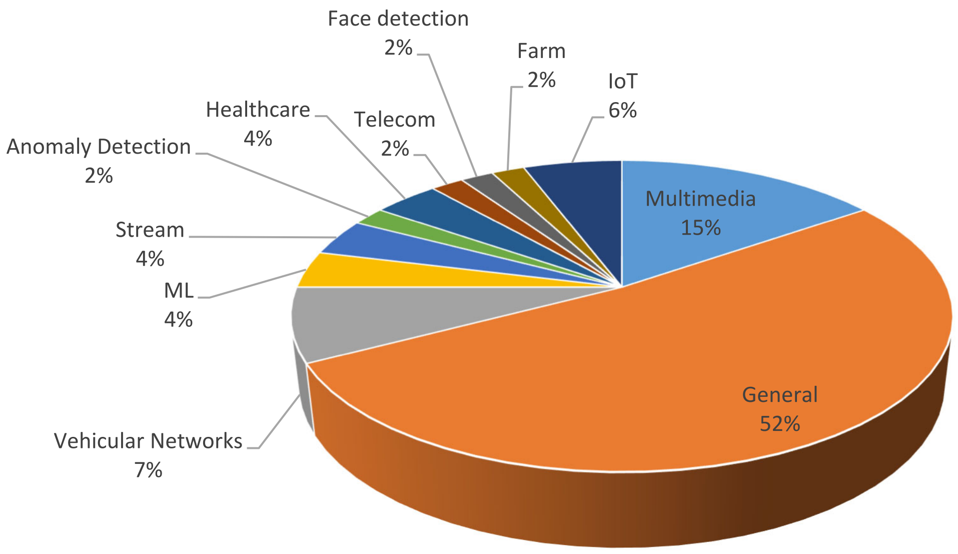

Fig. 9 Some case studies are presented related to the scheduling strategies in serverless computing

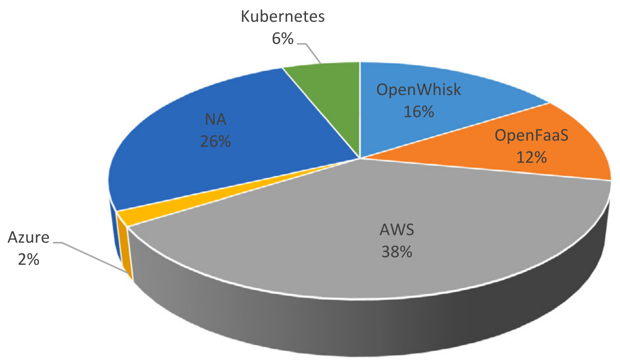

Fig. 10 Evaluation tools the scheduling strategies in serverless computing

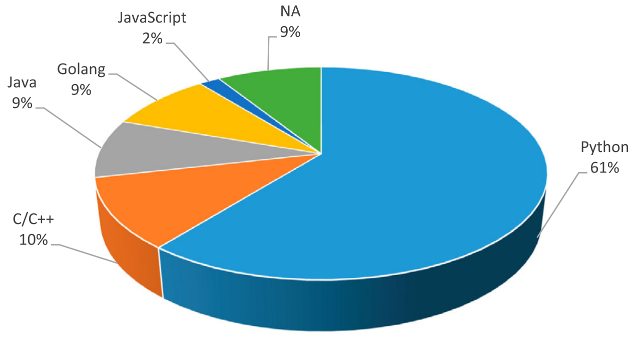

Fig. 11 Using programming languages to implement scheduling strategies in serverless computing

Python had the most significant utilization for implementation tools on scheduling strategies in serverless computing, with 61% of the utilization. Further, 9 % of the research articles do not specify the programming language for the proposed model in their papers. Finally, C/C++, Golang, Java, and JavaScript programming languages with 10%, 9%, 9%, and 2% ranks, respectively, are in the following places in the chart.

7 Open issues

This section discusses some open issues related to scheduling and implementing challenges in serverless computing systems. Scheduling is a crucial issue in resource allocation on serverless computing that various approaches, such as energy-aware, resource-aware, etc., can effectively handle. In serverless computing systems, scheduling-based methods primarily address the allocation of resources by providers, considering the client's dynamism in the cloud environment. In the implementation of serverless computing, the old resource allocation approach has been transformed into an on-demand execution approach where resources are allocated on a demand basis. In the serverless computing approach, providers and customers can decrease their service costs according to consumer demand by switching from a rental approach to a pay-per-use approach. Even though reducing task completion time remains the most critical objective. However, cloud service providers are now primarily responsible for resource utilization and isolation measures. Several general challenges are discussed in the following sections that should be addressed during the development of scheduling algorithms and may cause problems in the future.

7.1 Future research opportunities

This subsection aims to provide a technical answer to the following question:

Hence, this section presents a new perspective and emphasizes the need to motivate experts and researchers to dig deeper into the subject to present relevant challenges that have not been addressed previously. Hence, a research direction challenge is presented based on the currently available assessment criteria. According to TQ6, we suggest new informative challenges and open issues. Figure 12 depicts current open issues through their main challenges of scheduling algorithms in serverless computing systems based on further directions in this issue. Based on Fig. 12, there is a new open issue in the scheduling algorithm for serverless computing as follow below:

TQ6: What are the future research directions and open perspectives for scheduling in the serverless computing approach?

Based on the discussions mentioned above, we have categorized this technical question into nine categories: Compatibility costs, Component coordination, Differences in capability, Heterogeneity characteristics, Differences in goals, Awareness of the surroundings, Accessibility to resources, Storage of data and Access time to data. Due to the inherent non-deterministic nature of scheduling algorithms in resource allocation optimization problems, they can be considered hybrid with other event-triggering algorithms.

- *Compatibility costs* By utilizing the serverless approach, providers can determine the best method for performing each task according to the cumulative task duration they provided due to shifting goals and choose the most efficient approach. Thus, providers are facing a new scheduling challenge at the task level. However, it is crucial to understand that this change and adaptability also come with a cost and are not without their costs. As scheduling algorithms are primarily developed to reduce costs and maximize resources, this case can create a challenge for scheduling algorithms in optimizing resource allocation [77, 79, 101].
- *Component coordination* Orchestrating applications on serverless computing required load balancing and application-independent engineering to find the right balance between proactive resource allocation, data locality, and suitable response time guarantees that satisfy both parties. It will be difficult for serverless computing providers to overcome scheduling challenges if they ignore the correct configuration for orchestration. In other words, not orchestrating between all the provider components in every system in a

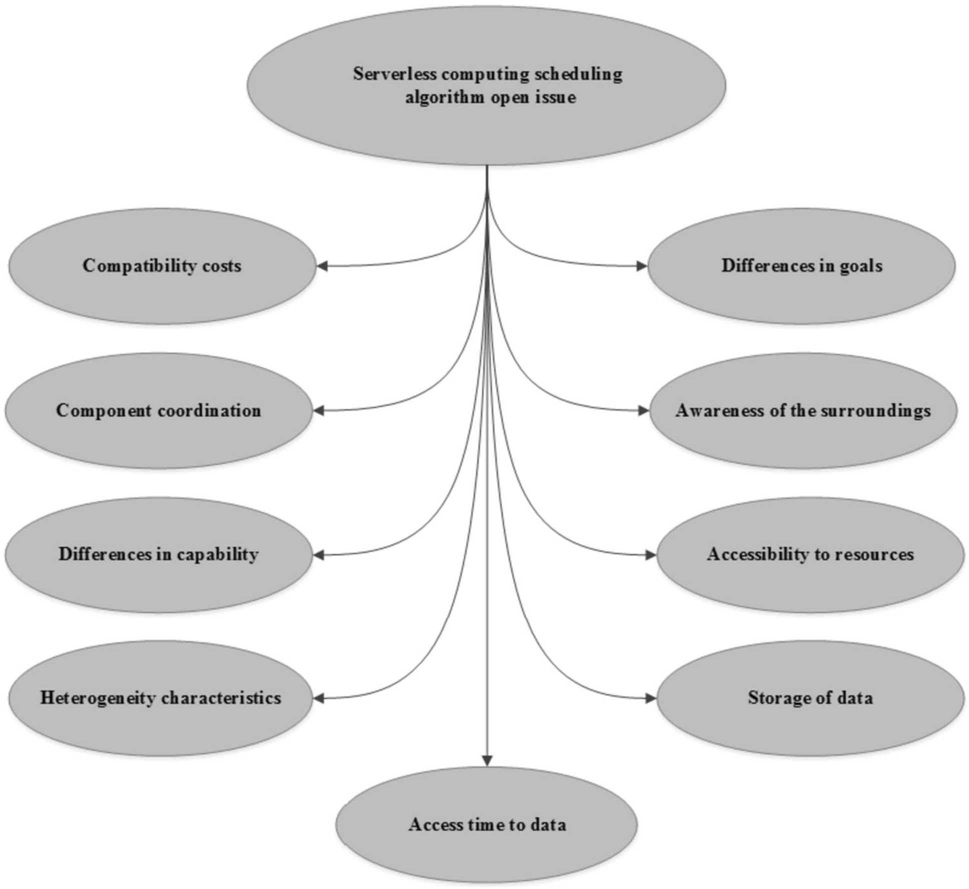
Serverless computing scheduling algorithm open issue
Compatibility costs
Component coordination
Differences in capability
Heterogeneity characteristics
Differences in goals
Awareness of the surroundings
Accessibility to resources
Storage of data
Access time to data

Fig. 12 Serverless computing scheduling algorithm open issues

serverless computing approach can create problems for scheduling processes in resource allocation, leading to a decrease in efficiency, increasing costs, and eventually a reduction in the quality of service provided to clients. Hence, it is the provider's responsibility to resolve these scheduling challenges according to the appropriate approach they use [83, 94, 101, 116].

- *Differences in capability* As a result of serverless providers' performance and capability differences, awareness-based scheduling algorithms, when faced with different situations, can deviate from the schedule-considered goal that must then be achieved, resulting in the execution deviating from the schedule. Consequently, this execution may violate any of the specified restrictions. For instance, when the maximum number of concurrent clients has been reached, this event may be even more harmful. The workflow might suffer when multiple tasks need to coincide. The reason confronting this type of challenge is a significant difference between the prevailing infrastructure and the optimal system, which causes an essential challenge. To resolve this challenge, function-as-a-service providers that provide this type of service in various ways can be evaluated based on their structure, performance, and mechanisms. A practical method of facilitating this process is continuously monitoring the invoked and used functions [74, 83, 86, 90].
- *Characteristics of heterogeneity* The best feature of serverless computing is edge-device implementation. Edge devices have the feature of being heterogeneous, which means they have a variety of processing capacities and different types of processing resources. Serverless providers must,therefore, be able to adapt to this heterogeneous property, respond appropriately to varying requests, and manage all client requests in various ways. Hence, this issue causes providers to select different scheduling approaches based on environment, applications, and demand. Consequently, this causes them to face various challenges when attempting to improve the quality of service presented to clients. For instance, in the case of stateless, short-lived processes, it is necessary to adjust the scheduling process to accommodate the reduced latency and time required to access resources. Resource allocation efficiency can be maximized, reducing energy consumption, response times, deadlines, and costs. It can increase the level of user satisfaction with the services provided. In contrast, the service provider should ensure that requests are answered as efficiently and quickly as possible for data users who work with time-sensitive data. Providers always need to be required to develop a comprehensive, intelligent, and detailed scheduling algorithm that fits and is compatible with all applications' performance specifications. The issue is a significant challenge for providers [90, 104, 107].
- *Differences in goals* A provider's goal is to make the maximum use of their resources. In contrast, the customer's goal is to get the most accurate responses possible as soon as possible. The amount of computing power or memory required will vary depending on the requirements of the function in many different circumstances. As such, this can be a very challenging task regarding local conditions. For instance, serverless providers might charge a fee based on the time it takes to perform a particular function or impose an execution time limit based on how long the function must run before it can be executed. Users must consider parallelism constraints, overheads, and how the data is obtained when scheduling the processing of created workloads, employing functions, and allocating resources for those workloads. However, the scheduling approach for cloud functions remains a significant challenge [80, 94].
- *Surrounding awareness* Once an idealized infrastructure is in place, it should be possible to execute requested functions immediately after they are submitted instead of delaying them for some time. Implementing this event can be very challenging in an actual implementation environment. Scheduling different resources according to the optimization criteria constraints, costs, and time can be extremely time-consuming and challenging. Therefore, it is possible to consider dynamic or static schedules based on the environment, application, and objective of the schedule event schedule can be implemented in several different ways depending on the application, the goal, and the environment. In a static approach, tasks are allocated to computing resources before execution based on information acquired from previous operations, for example. As far as using resources is concerned, this method is inarguably inefficient on the serverless platform. On the other hand, dynamic approaches monitor the execution schedule during execution to adjust it more efficiently and effectively. When deadlines are at risk, delegating tasks to quicker functions may be an option. Dynamic approaches differ from static approaches in that they have to be set based on awareness of various environmental events; otherwise, there will be an increase in the implementation costs if the scheduler is not set based on awareness of these events [73, 100, 118, 124].
- *Access to resources* Sometimes, specific processes may require more additional resources than the logical limit allocated, which can result in difficulties in scheduling algorithms executing those processes and allocating resources. Thus, this issue reduces resources, which prevents other processes from completing their tasks.

The specific limitations imposed by the service providers on the execution time for cloud functions will delay the execution of serverless computing functions. There are particular limitations on the execution time of serverless computing functions imposed by the providers. It is critical to use a hybrid execution strategy to execute tasks that may exceed the time limitation. It is necessary to use standard VMs to perform these tasks. However, there is still an issue regarding virtual machine deployment, which requires determining the size of VMs at the outset and the duration of their startup and shutdown [85, 91, 125].

- *Data storage* Because serverless computing functions are generally stateless, it is usually necessary to employ an external storage device to store temporary data that the serverless computing function needs to access. As a result, the existing structure of serverless computing can be used to store objects. The data is not transmitted directly among resources in serverless computing scheduling, as opposed to traditional scheduling of heterogeneous sources, in which the data is transmitted directly among resources. Due to this issue, scheduling algorithms in awareness-based schedules may face challenges in maximizing resource allocation efficiency and performance that need additional data by increasing awareness-based scheduling processes [90, 108, 122].
- *Access time to data* Furthermore, the serverless approach employs a stateless worker pattern, meaning that every function must externalize the information required to maintain two consecutive calls. Although considered environments implemented by providers may remain operational after the event has been completed, it's not guaranteed that this environment will remain active. In other words, it means the implementation of a specific

environment for a specific event. Hence, the time spent accessing data and initializing code during the execution of a function also has a time-related aspect to billed execution time. Even though this amount is less compared to the time expended on VMs, it remains significant compared to the duration of the entire function. Failure to pay attention to this case could result in additional costs. Since scheduling algorithms are primarily designed to optimize resource allocation and reduce costs, this case can pose a challenge to scheduling algorithms [81, 90, 110, 115].

8 Conclusion

Serverless computing has allowed developers to concentrate solely on developing their applications since it leaves users to put aside infrastructure settings. In addition, it has introduced a new payment model and a microservice architecture that provides many advantages, such as converting an application to functions. Serverless computing is being considered as a mechanism to implement and use functions in devices at the edge of the network, such as IoT devices with limited resources, leading to a particular focus on optimal resource utilization. The scheduling approach is an essential step in optimizing the process of utilizing resources most efficiently, providing providers of serverless services with a mechanism to maximize their resource allocation and utilization efficiency. In this article, an attempt has been made to examine new approaches and algorithms used in the scheduling algorithms in serverless computing. Next, an attempt has been made to provide the taxonomy process presented in this research based on the approaches and algorithms used in serverless computing. The proposed taxonomy is classified into six main fields: Energy-aware, Data-aware, Deadline-aware, Package-aware, Resource-aware, and Hybrid. Based on these factors, which have been derived from the review articles, the technical questions listed in Sect. 4.1 in the tabular form (i.e., Table 3) have been answered using comparative charts to help explain the results. The research challenges were discussed as important issues and open issues. It is important to note that these challenges are presented explicitly for serverless environments with their dynamic behaviors and resource scheduling problems. In this regard, these investigated challenges may have an important role in the future directions of the field of studies in which motivated researchers work.

Author contributions MG, MG-A, LE conducted this research. MG: Methodology, Software, Validation, Writing original draft. MG-A: Investigation, Resources, Data curation, Visualization. LE: Investigation, Visualization.

Funding This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data availability The datasets used or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Competing interests We certify that there is no actual or potential conflict of interest in relation to this article.

Ethical approval All procedures performed in studies involving human participants were in accordance with the ethical standards of the institutional and/or national research committee and with the 1964 Helsinki declaration and its later amendments or comparable ethical standards. This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Li, Y., Lin, Y., Wang, Y., Ye, K., Xu, C.: Serverless computing: state-of-the-art, challenges and opportunities. IEEE Trans. Serv. Comput. **16**(2), 1522–1539 (2022)
2. Barcelona-Pons, D., Sutra, P., Sánchez-Artigas, M., París, G., García-López, P.: Stateful serverless computing with crucial. ACM Trans. Softw. Eng. Methodol. **31**(3), 1–38 (2022)
3. Sharma, P.: Challenges and opportunities in sustainable serverless computing. ACM SIGENERGY Energy Inform. Rev. **3**(3), 53–58 (2023)
4. Cao, Y., Niu, B., Wang, H., Zhao, X.: Event-based adaptive resilient control for networked nonlinear systems against unknown deception attacks and actuator saturation. Int. J. Robust Nonlinear Control (2024). https://doi.org/10.1002/rnc.7231
5. Lee, H., Satyam, K., Fox, G.: Evaluation of production serverless computing environments. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pp. 442–450. IEEE (2018)
6. Wu, W., Zhang, L., Wu, Y., Zhao, H.: Adaptive saturated two-bit-triggered bipartite consensus control for networked MASs with periodic disturbances: a low-computation method. IMA J. Math. Control. Inf. (2024). https://doi.org/10.1093/imamci/dnae002
7. Le, D.N., Pal, S., Pattnaik, P.K., OpenFaaS. Cloud computing solutions: architecture, data storage, implementation and security. 287–303 (2022)
8. Marin, E., Perino, D., Di Pietro, R.: Serverless computing: a security perspective. J. Cloud Comput. **11**(1), 1–12 (2022)
9. Huang, S., Zong, G., Zhao, N., Zhao, X., Ahmad, A.M.: Performance recovery-based fuzzy robust control of networked nonlinear systems against actuator fault: a deferred actuator-switching method. Fuzzy Sets Syst. **480**, 108858 (2024). https://doi.org/10.1016/j.fss.2024.108858
10. Tarahomi, M., Izadi, M., Ghobaei-Arani, M.: An efficient power-aware VM allocation mechanism in cloud data centers: a micro genetic-based approach. Cluster Comput. **24**, 919–934 (2021). https://doi.org/10.1007/s10586-020-03152-9
11. Mampage, A., Karunasekera, S., Buyya, R.: A holistic view on resource management in serverless computing environments: taxonomy and future directions. ACM Comput. Surv. **54**(11s), 1–36 (2022)
12. Benedetti, P., Femminella, M., Reali, G., Steenhaut, K.: Experimental analysis of the application of serverless computing to IoT platforms. Sensors **21**(3), 928 (2021)
13. Sarkar, S., Wankar, R., Srirama, S.N., Suryadevara, N.K.: Serverless management of sensing systems for fog computing framework. IEEE Sens. J. **20**(3), 1564–1572 (2019)
14. Xue, B., Yang, Q., Jin, Y., Zhu, Q., Lan, J., Lin, Y., Tan, J., et al.: Genotoxicity assessment of haloacetaldehyde disinfection byproducts via a simplified yeast-based toxicogenomics assay. Environ. Sci. Technol. **57**(44), 16823–16833 (2023). https://doi.org/10.1021/acs.est.3c04956
15. Zhang, C., Zhu, D., Luo, Q., Liu, L., Liu, D., Yan, L., Zhang, Y.: Major factors controlling fracture development in the Middle Permian Lucaogou Formation tight oil reservoir, Junggar Basin, NW China. J. Asian Earth Sci. **146**, 279–295 (2017). https://doi.org/10.1016/j.jseaes.2017.04.032
16. Rajan, A.P.: A review on serverless architectures-function as a service (FaaS) in cloud computing. TELKOMNIKA (Telecommun. Comput. Electron. Control) **18**(1), 530–537 (2020)
17. Hellerstein, J.M., Faleiro, J., Gonzalez, J.E., Schleier-Smith, J., Sreekanti, V., Tumanov, A., Wu, C.: Serverless computing: one step forward, two steps back. arXiv preprint arXiv:1812.03651 (2018)
18. Naranjo, D.M., Risco, S., de Alfonso, C., Pérez, A., Blanquer, I., Moltó, G.: Accelerated serverless computing based on GPU virtualization. J. Parallel Distrib. Comput. **139**, 32–42 (2020)
19. Bebortta, S., Das, S.K., Kandpal, M., Barik, R.K., Dubey, H.: Geospatial serverless computing: architectures, tools and future directions. ISPRS Int. J. Geo Inf. **9**(5), 311 (2020)
20. Patros, P., Spillner, J., Papadopoulos, A.V., Varghese, B., Rana, O., Dustdar, S.: Toward sustainable serverless computing. IEEE Internet Comput. **25**(6), 42–50 (2021)
21. Hassan, H.B., Barakat, S.A., Sarhan, Q.I.: Survey on serverless computing. J. Cloud Comput. **10**(1), 1–29 (2021)
22. Jia, Z., Witchel, E.: 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, pp. 152–166 (2021)
23. Grafberger, A., Chadha, M., Jindal, A., Gu, J., Gerndt, M.: FedLess: secure and scalable federated learning using serverless computing. In: 2021 IEEE International Conference on Big Data (Big Data), pp. 164–173. IEEE (2021)
24. Kelly, D., Glavin, F., Barrett, E.: Serverless computing: Behind the scenes of major platforms. In: 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), pp. 304–312. IEEE (2020)
25. Khatri, D., Khatri, S.K., Mishra, D.: Potential bottleneck and measuring performance of serverless computing: a literature study. In: 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), pp. 161–164. IEEE (2020)
26. Kjorveziroski, V., Bernad Canto, C., Juan Roig, P., Gilly, K., Mishev, A., Trajkovik, V., Filiposka, S.: IoT serverless computing at the edge: open issues and research direction. Trans. Netw. Commun. (2021)
27. Lenarduzzi, V., Daly, J., Martini, A., Panichella, S., Tamburri, D.A.: Toward a technical debt conceptualization for serverless computing. IEEE Softw. **38**(1), 40–47 (2020)
28. Golec, M., Ozturac, R., Pooranian, Z., Gill, S.S., Buyya, R.: iFaaSBus: a security-and privacy-based lightweight framework for serverless computing using IoT and machine learning. IEEE Trans. Ind. Inf. **18**(5), 3522–3529 (2021)
29. Mondal, S.K., Pan, R., Kabir, H.M., Tian, T., Dai, H.N.: Kubernetes in IT administration and serverless computing: an empirical study and research challenges. J. Supercomput. **78**(2), 2937–2987 (2022)
30. Prakash, A.A., Kumar, K.S.: Cloud serverless security and services: a survey. In: Applications of Computational Methods in Manufacturing and Product Design, pp. 453–462. Springer, Singapore (2022)
31. Kumari, A., Behera, R.K., Sahoo, B., Misra, S.: Role of serverless computing in healthcare systems: case studies. In: International Conference on Computational Science and Its Applications, pp. 123–134. Springer, Cham (2022)
32. Zhang, Y., Goiri, Í., Chaudhry, G.I., Fonseca, R., Elnikety, S., Delimitrou, C., Bianchini, R.: Faster and cheaper serverless computing on harvested resources. In: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, pp. 724–739 (2021)
33. Yan, M., Castro, P., Cheng, P., Ishakian, V.: Building a chatbot with serverless computing. In: Proceedings of the 1st International Workshop on Mashups of Things and APIs, pp. 1–4 (2016)
34. Sewak, M., Singh, S.: Winning in the era of serverless computing and function as a service. In: 2018 3rd International

Conference for Convergence in Technology (I2CT), pp. 1–5. IEEE (2018)
35. Li, Z., Guo, L., Cheng, J., Chen, Q., He, B., Guo, M.: The serverless computing survey: a technical primer for design architecture. ACM Comput. Surv. **54**(10s), 1–34 (2022)
36. Sankaran, A., Datta, P. and Bates, A.: Workflow integration alleviates identity and access management in serverless computing. In: Annual Computer Security Applications Conference, pp. 496–509 (2020)
37. Stigler, M.: Understanding serverless computing. In: Beginning Serverless Computing, pp. 1–14. Apress, Berkeley (2018)
38. Ginzburg, S., Freedman, M.J.: Serverless isn't server-less: measuring and exploiting resource variability on cloud FaaS platforms. In: Proceedings of the 2020 Sixth International Workshop on Serverless Computing, pp. 43–48 (2020)
39. Taibi, D., Spillner, J., Wawruch, K.: Serverless computing-where are we now, and where are we heading? IEEE Softw. **38**(1), 25–31 (2020)
40. Ghorbian, M., Ghobaei-Arani, M.: A Blockchain-enabled serverless approach for IoT healthcare applications. In: Serverless Computing: Principles and Paradigms, pp. 193–218. Springer, Cham (2023)
41. Casale, G., Artač, M., Van Den Heuvel, W.J., van Hoorn, A., Jakovits, P., Leymann, F., Long, M., Papanikolaou, V., Presenza, D., Russo, A., Srirama, S.N.: Radon: rational decomposition and orchestration for serverless computing. SICS Softw.-Intensive Cyber-Phys. Syst. **35**(1), 77–87 (2020)
42. Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L., Pallickara, S.: Serverless computing: an investigation of factors influencing microservice performance. In: 2018 IEEE International Conference on Cloud Engineering (IC2E), pp. 159–169. IEEE (2018)
43. Xu, Z., Zhang, H., Geng, X., Wu, Q., Ma, H.: Adaptive function launching acceleration in serverless computing platforms. In: 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), pp. 9–16. IEEE (2019)
44. Adzic, G., Chatley, R.: Serverless computing: economic and architectural impact. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 884–889 (2017)
45. Mohanty, S.K., Premsankar, G., Di Francesco, M.: An evaluation of open source serverless computing frameworks. CloudCom **2018**, 115–120 (2018)
46. Aske, A., Zhao, X.: Supporting multi-provider serverless computing on the edge. In: Proceedings of the 47th International Conference on Parallel Processing Companion, pp. 1–6 (2018)
47. Kaffes, K., Yadwadkar, N.J., Kozyrakis, C.: Centralized core-granular scheduling for serverless functions. In: Proceedings of the ACM Symposium on Cloud Computing, pp. 158–164 (2019)
48. Mahmoudi, N., Khazaei, H.: Performance modeling of serverless computing platforms. IEEE Trans. Cloud Comput. **10**(4), 2834–2847 (2020)
49. Kaffes, K., Yadwadkar, N.J., Kozyrakis, C.: Practical scheduling for real-world serverless computing. arXiv preprint arXiv:2111.07226 (2021)
50. Zuk, P., Rzadca, K.: Scheduling methods to reduce response latency of function as a service. In: 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp. 132–140. IEEE (2020)
51. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N., Gonzalez, J.E.: Cloud programming simplified: a berkeley view on serverless computing. arXiv preprint arXiv:1902.03383 (2019)
52. Bisht, J., Vampugani, V.S.: Load and cost-aware min-min workflow scheduling algorithm for heterogeneous resources in fog, cloud, and edge scenarios. Int. J. Cloud Appl. Comput. **12**(1), 1–20 (2022)
53. Majewski, M., Pawlik, M., Malawski, M.: Algorithms for scheduling scientific workflows on serverless architecture. In: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 782–789. IEEE (2021)
54. Mahmoudi, N., Khazaei, H.: MLProxy: SLA-aware reverse proxy for machine learning inference serving on serverless computing platforms. arXiv preprint arXiv:2202.11243 (2022)
55. Nezafat Tabalvandani, M.A., Hosseini Shirvani, M., Motameni, H.: Reliability-aware web service composition with cost minimization perspective: a multi-objective particle swarm optimization model in multi-cloud scenarios. Soft Comput. 1–24 (2023)
56. Suresh, A., Somashekar, G., Varadarajan, A., Kakarla, V.R., Upadhyay, H., Gandhi, A.: Ensure: efficient scheduling and autonomous resource management in serverless environments. In: 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), pp. 1–10. IEEE (August)
57. Pathak, P., Singh, P.: Kubernetes and Docker the Star Duo of container culture. In: Proceedings of 3rd International Conference on Machine Learning, Advances in Computing, Renewable Energy and Communication: MARC 2021, pp. 79–90. Springer, Singapore (2022)
58. Balaji, K., Sai Kiran, P., Sunil Kumar, M.: Power aware virtual machine placement in IaaS cloud using discrete firefly algorithm. Appl. Nanosci. **13**(3), 2003–2011 (2023)
59. Jiang, J., Gan, S., Du, B., Alonso, G., Klimovic, A., Singla, A., Wu, W., Wang, S., Zhang, C.: A systematic evaluation of machine learning on serverless infrastructure. VLDB J. 1–25 (2023)
60. Wang, H., Niu, D., Li, B.: Distributed machine learning with a serverless architecture. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 1288–1296. IEEE (2019)
61. Mampage, A., Karunasekera, S., Buyya, R.: Deep reinforcement learning for application scheduling in resource-constrained, multi-tenant serverless computing environments. Future Gener. Comput. Syst. **143**, 277–292 (2023)
62. Alqaryouti, O., Siyam, N.: Serverless computing and scheduling tasks on cloud: a review. Am. Acad. Sci. Res. J. Eng. Technol. Sci. **40**(1), 235–247 (2018)
63. Kjorveziroski, V., Filiposka, S., Trajkovik, V.: IoT serverless computing at the edge: a systematic mapping review. Computers **10**(10), 130 (2021)
64. Saurav, S.K., Benedict, S.: A taxonomy and survey on energy-aware scientific workflows scheduling in large-scale heterogeneous architecture. In: 2021 6th International Conference on Inventive Computation Technologies (ICICT), pp. 820–826. IEEE (2021)
65. Shafiei, H., Khonsari, A., Mousavi, P.: Serverless computing: a survey of opportunities, challenges, and applications. ACM Comput. Surv. **54**(11s), 1–32 (2022)
66. Xie, R., Tang, Q., Qiao, S., Zhu, H., Yu, F.R., Huang, T.: When serverless computing meets edge computing: architecture, challenges, and open issues. IEEE Wirel. Commun. **28**(5), 126–133 (2021)
67. Cassel, G.A.S., Rodrigues, V.F., da Rosa Righi, R., Bez, M.R., Nepomuceno, A.C., da Costa, C.A.: Serverless computing for Internet of Things: a systematic literature review. Future Gener. Comput. Syst. **128**, 299–316 (2022)
68. Ghobaei-Arani, M. and Ghorbian, M.: Scheduling mechanisms in serverless computing. In: Serverless Computing: Principles and Paradigms, pp. 243–273. Springer, Cham (2023)

69. Pérez, A., Risco, S., Naranjo, D.M., Caballer, M., Moltó, G.: On-premises serverless computing for event-driven data processing applications. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), pp. 414–421. IEEE (2019)
70. Jarachanthan, J., Chen, L., Xu, F., Li, B.: AMPS-Inf: automatic model partitioning for serverless inference with cost efficiency. In: 50th International Conference on Parallel Processing, pp. 1–12 (2021)
71. Hosseini Shirvani, M., Noorian Talouki, R.: Bi-objective scheduling algorithm for scientific workflows on cloud computing platform with makespan and monetary cost minimization approach. Complex Intell. Syst. **8**(2), 1085–1114 (2022)
72. Wu, S., Tao, Z., Fan, H., Huang, Z., Zhang, X., Jin, H., Yu, C., Cao, C.: Container lifecycle-aware scheduling for serverless computing. Software **52**(2), 337–352 (2022)
73. Kallam, S., Patan, R., Ramana, T.V., Gandomi, A.H.: Linear weighted regression and energy-aware greedy scheduling for heterogeneous big data. Electronics **10**(5), 554 (2021)
74. Aslanpour, M.S., Toosi, A.N., Cheema, M.A., Gaire, R.: Energy-aware resource scheduling for serverless edge computing. In: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 190–199. IEEE (2022)
75. Gunasekaran, J.R., Thinakaran, P., Chidambaram, N., Kandemir, M.T., Das, C.R.: Fifer: tackling underutilization in the serverless era. arXiv preprint arXiv:2008.12819 (2020)
76. Aslanpour, M.S., Toosi, A.N., Gaire, R. and Cheema, M.A.: WattEdge: a holistic approach for empirical energy measurements in edge computing. In: International Conference on Service-Oriented Computing, pp. 531–547. Springer, Cham (2021)
77. Rausch, T., Rashed, A., Dustdar, S.: Optimized container scheduling for data-intensive serverless edge computing. Future Gener. Comput. Syst. **114**, 259–271 (2021)
78. Wu, J., Wu, M., Li, H., Li, L., Li, L.: A serverless-based, on-the-fly computing framework for remote sensing image collection. Remote Sens. **14**(7), 1728 (2022)
79. Yu, M., Cao, T., Wang, W., Chen, R.: Restructuring serverless computing with data-centric function orchestration. arXiv preprint arXiv:2109.13492 (2021)
80. Das, S.: Ant Colony Optimization for MapReduce Application to Optimise Task Scheduling in Serverless Platform (Doctoral dissertation, Dublin, National College of Ireland) (2021)
81. Seubring, W., Lazovik, A., Blaauw, F.: Data Locality Aware Scheduling on a Serverless Edge Platform (Doctoral dissertation) (2021)
82. Jindal, A., Gerndt, M., Chadha, M., Podolskiy, V., Chen, P.: Function delivery network: extending serverless computing for heterogeneous platforms. Software **51**(9), 1936–1963 (2021)
83. Nestorov, A.M., Polo, J., Misale, C., Carrera, D., Youssef, A.S.: Performance evaluation of data-centric workloads in serverless environments. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), pp. 491–496. IEEE (2021)
84. Przybylski, B., Żuk, P., Rzadca, K.: Data-driven scheduling in serverless computing to reduce response time. In: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 206–216. IEEE (2021)
85. García-López, P., Sánchez-Artigas, M., Shillaker, S., Pietzuch, P., Breitgand, D., Vernik, G., Sutra, P., Tarrant, T., Ferrer, A.J.: Servermix: tradeoffs and challenges of serverless data analytics. arXiv preprint arXiv:1907.11465 (2019)
86. HoseinyFarahabady, M.R., Taheri, J., Zomaya, A.Y. and Tari, Z.: Data-intensive workload consolidation in serverless (Lambda/FaaS) platforms. In: 2021 IEEE 20th International Symposium on Network Computing and Applications (NCA), pp. 1–8. IEEE (2021)
87. Tang, Y. and Yang, J.: Lambdata: optimizing serverless computing by making data intents explicit. In: 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), pp. 294–303. IEEE (2020)
88. Singhvi, A., Houck, K., Balasubramanian, A., Shaikh, M.D., Venkataraman, S., Akella, A.: Archipelago: a scalable low-latency serverless platform. arXiv preprint arXiv:1911.09849 (2019)
89. Asghari Alaie, Y., Hosseini Shirvani, M., Rahmani, A.M.: A hybrid bi-objective scheduling algorithm for execution of scientific workflows on cloud platforms with execution time and reliability approach. J. Supercomput. **79**(2), 1451–1503 (2023)
90. Mampage, A., Karunasekera, S., Buyya, R.: Deadline-aware dynamic resource management in serverless computing environments. In: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 483–492. IEEE (2021)
91. Wang, B., Ali-Eldin, A., Shenoy, P.: Lass: running latency sensitive serverless computations at the edge. In: Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing, pp. 239–251 (2021)
92. Krishna, S.R., Majji, S., Kishore, S.K., Jaiswal, S., Kostka, J.A.L., Chouhan, A.S.: Optimization of time-driven scheduling technique for serverless cloud computing. Turk. J. Comput. Math. Educ. **12**(10), 1–8 (2021)
93. Zuk, P., Rzadca, K.: Reducing response latency of composite functions-as-a-service through scheduling. J. Parallel Distrib. Comput. **167**, 18–30 (2022)
94. Fan, D. and He, D.: A scheduler for serverless framework base on kubernetes. In: Proceedings of the 2020 4th High Performance Computing and Cluster Technologies Conference & 2020 3rd International Conference on Big Data and Artificial Intelligence, pp. 229–232 (2020)
95. Totoy, G., Boza, E.F., Abad, C.L.: An Extensible Scheduler for the OpenLambda FaaS Platform. Min-Move'18 (2018)
96. Aumala, G., Boza, E., Ortiz-Avilés, L., Totoy, G., Abad, C.: Beyond load balancing: package-aware scheduling for serverless platforms. In: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 282–291. IEEE (2019)
97. Bai, T., Nie, J.Y., Zhao, W.X., Zhu, Y., Du, P., Wen, J.R.: An attribute-aware neural attentive model for next basket recommendation. In: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, pp. 1201–1204 (2018)
98. Chetabi, F.A., Ashtiani, M., Saeedizade, E.: A package-aware approach for function scheduling in serverless computing environments. J.f Grid Comput. **21**(2), 23 (2023)
99. Ebrahimpour, H., Ashtiani, M., Bakhshi, F., Bakhtiariazad, G.: A heuristic-based package-aware function scheduling approach for creating a trade-off between cold start time and cost in FaaS computing environments. J. Supercomput. 1–49 (2023)
100. Suresh, A., Gandhi, A.: Fnsched: an efficient scheduler for serverless functions. In: Proceedings of the 5th international workshop on serverless computing, pp. 19–24 (2019)
101. Yuvaraj, N., Karthikeyan, T., Praghash, K.: An improved task allocation scheme in serverless computing using gray wolf Optimization (GWO) based reinforcement learning (RIL) approach. Wirel. Pers. Commun. **117**(3), 2403–2421 (2021)
102. Cheng, Y. and Zhou, Z.: Autonomous resource scheduling for real-time and stream processing. In: 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp. 1181–1184. IEEE (2018)

103. Lakhan, A., Mohammed, M.A., Rashid, A.N., Kadry, S., Panityakul, T., Abdulkareem, K.H., Thinnukool, O.: Smart-contract aware ethereum and client-fog-cloud healthcare system. Sensors **21**(12), 4093 (2021)
104. Kim, Y.K., HoseinyFarahabady, M.R., Lee, Y.C., Zomaya, A.Y.: Automated fine-grained cpu cap control in serverless computing platform. IEEE Trans. Parallel Distrib. Syst. **31**(10), 2289–2301 (2020)
105. Patterson, L., Pigorovsky, D., Dempsey, B., Lazarev, N., Shah, A., Steinhoff, C., Bruno, A., Hu, J., Delimitrou, C.: A hardware-software stack for serverless edge swarms. arXiv preprint arXiv:2112.14831 (2021)
106. Soltani, B., Ghenai, A. and Zeghib, N.: A migration-based approach to execute long-duration multi-cloud serverless functions. In: ICAASE, pp. 42–50 (2018)
107. Zhang, H., Shen, M., Huang, Y., Wen, Y., Luo, Y., Gao, G., Guan, K.: A serverless cloud-fog platform for dnn-based video analytics with incremental learning. arXiv preprint arXiv:2102.03012 (2021)
108. Gadepalli, P.K., Peach, G., Cherkasova, L., Aitken, R., Parmer, G.: Challenges and opportunities for efficient serverless computing at the edge. In: 2019 38th Symposium on Reliable Distributed Systems (SRDS), pp. 261–2615. IEEE (2019)
109. Fard, H.M., Prodan, R., Wolf, F.: Dynamic multi-objective scheduling of microservices in the cloud. In: 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), pp. 386–393. IEEE (2020)
110. Zhang, M., Krintz, C., Wolski, R.: Edge-adaptable serverless acceleration for machine learning Internet of Things applications. Software **51**(9), 1852–1867 (2021)
111. Aytekin, A., Johansson, M.: Exploiting serverless runtimes for large-scale optimization. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), pp. 499–501. IEEE (2019)
112. Huang, Z., Mi, Z., Hua, Z.: HCloud: a trusted JointCloud serverless platform for IoT systems with blockchain. China Commun. **17**(9), 1–10 (2020)
113. Zhang, J., Wang, A., Li, M., Chen, Y., Cheng, Y., HyperFaaS: a truly elastic serverless computing framework
114. Denninnart, C., Gentry, J., Salehi, M.A.: Improving robustness of heterogeneous serverless computing systems via probabilistic task pruning. In: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 6–15. IEEE (2019)
115. Ling W, Tian C, Ma L, Hu Z.: Lite-Service: a framework to build and schedule telecom applications in device, edge and cloud. In: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS) 2018 Jun 28, pp. 708–717. IEEE (2018)
116. Silab, M.V., Hassanpour, S.B., Khonsari, A., Dadlani, A.: On skipping redundant computation via smart task deployment for faster serverless. In: ICC 2022-IEEE International Conference on Communications (pp. 5475–5480). IEEE (2022)
117. Tychalas, D., Karatza, H.: SaMW: a probabilistic meta-heuristic algorithm for job scheduling in heterogeneous distributed systems powered by microservices. Clust. Comput. **24**(3), 1735–1759 (2021)
118. Mujezinović, A., Ljubović, V.: Serverless architecture for workflow scheduling with unconstrained execution environment. In: 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 242–246. IEEE (2019)
119. Denninnart, C. and Salehi, M.A.: SMSE: a serverless platform for multimedia cloud systems. arXiv preprint arXiv:2201.01940 (2022)
120. Ao, L., Izhikevich, L., Voelker, G.M., Porter, G.: Sprocket: a serverless video processing framework. In: Proceedings of the ACM Symposium on Cloud Computing, pp. 263–274 (2018)
121. Wen, Z., Wang, Y. and Liu, F.: StepConf: SLO-aware dynamic resource configuration for serverless function workflows. In: IEEE INFOCOM 2022-IEEE Conference on Computer Communications, pp. 1868–1877. IEEE (2022)
122. Nesen, A., Bhargava, B.: Towards situational awareness with multimodal streaming data fusion: serverless computing approach. In: Proceedings of the International Workshop on Big Data in Emergent Distributed Environments, pp. 1–6 (2021)
123. Wu, C., Sreekanti, V., Hellerstein, J.M.: Transactional causal consistency for serverless computing. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 83–97 (2020)
124. Carver, B., Zhang, J., Wang, A., Anwar, A., Wu, P., Cheng, Y.: Wukong: a scalable and locality-enhanced framework for serverless parallel computing. In: Proceedings of the 11th ACM Symposium on Cloud Computing, pp. 1–15 (2020)
125. Tang, Q., Xie, R., Yu, F.R., Chen, T., Zhang, R., Huang, T., Liu, Y.: Distributed task scheduling in serverless edge computing networks for the internet of things: a learning approach. IEEE Internet Things J. **9**(20), 19634–19648 (2022)
126. De Palma, G., Giallorenzo, S., Mauro, J., Trentin, M., Zavattaro, G.: Topology-aware serverless function-execution scheduling. arXiv preprint arXiv:2205.10176 (2022)
127. Lakhan, A., Mohammed, M.A., Rashid, A.N., Kadry, S., Abdulkareem, K.H., Nedoma, J., Martinek, R., Razzak, I.: Restricted Boltzmann machine assisted secure serverless edge system for internet of medical things. IEEE J. Biomed. Health Inform. **27**(2), 673–683 (2022)

Mohsen Ghorbian is a Ph.D. candidate in the computer science department of the Islamic Azad University of Qom. He received the B.Sc and M.Sc degree in Software Engineering from Islamic Azad University. His research interests are cloud computing, serverless computing, blockchain, and machine learning techniques.

Mostafa Ghobaei-Arani received the Ph.D. Degree in Software Engineering from Islamic Azad University, Science and Research Branch, Tehran, Iran. He is Assistant Professor of Computer Engineering Department, Qom Branch, Islamic Azad University, Qom, Iran. He has published more than 80 journal papers in the area of distributed systems. His research interests include distributed computing, cloud computing, autonomic computing, edge/fog computing, serverless computing, soft computing, big data, and the IoT.

Leila Esmaeili is a faculty member of Computer Engineering at the Islamic Azad University of Qom. Moreover, she is a Ph.D. Candidate at Amirkabir University of Technology (Tehran Polytechnic), Department of Computer Engineering and Information Technology. Her research interests revolve around process mining, organizational mining, and recommender systems in the context of interactive systems such as social commerce and IoT.