# Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions

ZHIHENG ZHONG, The University of Melbourne
MINXIAN XU, Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences
MARIA ALEJANDRA RODRIGUEZ, The University of Melbourne
CHENGZHONG XU, University of Macau
RAJKUMAR BUYYA, The University of Melbourne

Containerization is a lightweight application virtualization technology, providing high environmental consistency, operating system distribution portability, and resource isolation. Existing mainstream cloud service providers have prevalently adopted container technologies in their distributed system infrastructures for automated application management. To handle the automation of deployment, maintenance, autoscaling, and networking of containerized applications, container orchestration is proposed as an essential research problem. However, the highly dynamic and diverse feature of cloud workloads and environments considerably raises the complexity of orchestration mechanisms. Machine learning algorithms are accordingly employed by container orchestration systems for behavior modeling and prediction of multi-dimensional performance metrics. Such insights could further improve the quality of resource provisioning decisions in response to the changing workloads under complex environments. In this article, we present a comprehensive literature review of existing machine learning-based container orchestration approaches. Detailed taxonomies are proposed to classify the current researches by their common features. Moreover, the evolution of machine learning-based container orchestration technologies from the year 2016 to 2021 has been designed based on objectives and metrics. A comparative analysis of the reviewed techniques is conducted according to the proposed taxonomies, with emphasis on their key characteristics. Finally, various open research challenges and potential future directions are highlighted.

CCS Concepts: • **General and reference → General literature**; • **Computer systems → Cloud computing;**

Additional Key Words and Phrases: Container orchestration, machine learning, cloud computing, resource provisioning, systematic review

This work is supported by the Key-Area Research and Development Program of Guangdong Province (NO. 2020B010164003), National Natural Science Foundation of China (No. 62102408), SIAT Innovation Program for Excellent Young Researchers, and Australian Research Council (ARC) Discovery Project.

Authors' addresses: Z. Zhong, M. A. Rodriguez, and R. Buyya, The Cloud Computing and Distributed Systems Laboratory, School of Computing and Information Systems, The University of Melbourne, Grattan Street, Parkville, Melbourne, Victoria VIC 3010, Australia; emails: zhiheng@student.unimelb.edu.au, {maria.read, rbuyya}@unimelb.edu.au; M. Xu (corresponding author), Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, 1068 Xueyuan Avenue, Shenzhen University Town, Shenzhen, Guangdong 518055, China; email: mx.xu@siat.ac.cn; C. Xu, State Key Lab of IOTSC, University of Macau, Avenida da Universidade, Taipa, Macau 999078, China; email: czxu@um.edu.mo.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Association for Computing Machinery.

0360-0300/2022/09-ART217 $15.00
https://doi.org/10.1145/3510415

**ACM Reference format:**
Zhiheng Zhong, Minxian Xu, Maria Alejandra Rodriguez, Chengzhong Xu, and Rajkumar Buyya. 2022. Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions. *ACM Comput. Surv.* 54, 10s, Article 217 (September 2022), 35 pages.
https://doi.org/10.1145/3510415

## 1 INTRODUCTION

Our era has witnessed the trend of cloud computing becoming the mainstream industrial computing paradigm, providing stable service delivery with high cost-efficiency, scalability, availability, and accessibility [1]. Existing mainstream cloud service providers, including **Amazon Web Services** (**AWS**) [2], Google [3], and Alibaba [4], prevalently adopt virtualization technologies including **virtual machines** (**VM**) and containers in their distributed system infrastructures for automated application deployment. In recent years, their infrastructures are evolving from VM centric to container centric [5].

Containers employ a logical packing mechanism that binds both software and dependencies together for application abstraction [6]. Unlike VMs that support hardware-level resource virtualization where each VM has to maintain its own **operating system** (**OS**), containers virtualize resources at the OS level where all the containers share the same OS with less overhead. Therefore, containers are more lightweight in nature with high application portability, resource efficiency, and environmental consistency. They define a standardized unit for application deployment under an isolated runtime system. Thanks to these features, we have observed the prevalence of container technologies for automatic application deployment under diverse cloud environments.

Following this trend of containerization, container orchestration techniques are proposed for the management of containerized applications. Different from the management of VM lifecycle via VM orchestration at large-grained control, container orchestration is the fine-grained and automated management process of a container lifecycle, including resource allocation, deployment, autoscaling, health monitoring, migration, load balance, security, and network configuration. For cloud service providers, which have to handle hundreds or thousands of containers simultaneously, a refined and robust container orchestration system is the key factor in controlling overall resource utilization, energy efficiency, and application performance. Under the surge of cloud workloads in terms of resource demands, bandwidth consumption, and **quality of service** (**QoS**) requirements, the traditional cloud computing environment is extended to fog and edge infrastructures that are close to end users with extra computational power [7]. Consequently, this also requires current container orchestration systems to be further enhanced in response to the rising resource heterogeneity, application distribution, workload diversity, and security requirements across hybrid cloud infrastructures.

### 1.1 Needs for Machine Learning-based Container Orchestration

Considering the increasingly diverse and dynamic cloud workloads processed by existing cloud service providers, it is still unclear how to automate the orchestration process for complex heterogeneous workloads under large-scale cloud computing systems [8, 9]. Note that containers bear a resemblance to VMs from the perspective of orchestration. **Machine-learning** (**ML**) has been applied for orchestration of VMs. For example, early efforts in the use of **reinforcement learning** (**RL**) algorithms for auto-configuration of VMs can be seen in [10–12]. In traditional cloud computing platforms, Container Orchestrators are usually designed with heuristic policies without consideration of the diversity of workload scenarios and QoS requirements [13]. Their main drawbacks are listed as follows:

(1) Most of these policies are static heuristic methods configured offline according to certain workload scenarios at a limited scale. For instance, threshold-based autoscaling strategies can only be suitable for managing a set of pre-defined workloads [6, 14, 15]. Such policies can not handle highly dynamic workloads where applications need to be scaled in/out at runtime according to specific behavior patterns (details and solutions are discussed in Sections 4.2.1 and 4.3.2).
(2) The performance of heuristic methods could dramatically degrade when the system scales up. For example, bin-packing algorithms, such as best fit or least fit algorithms, are widely utilized for solving task scheduling and resource allocation [16–18]. However, such algorithms could perform poorly with high task scheduling delays in large-scale compute clusters (details and solutions are discussed in Sections 4.3.1 and 4.3.3).
(3) Resource contention and performance interference between co-located applications are usually ignored. Co-located applications could compete for shared resources, which may cause application performance downgrade, extra maintenance costs, and **service-level agreement** (**SLA**) violations [3, 4, 8] (details and solutions are discussed in Section 4.2.3).
(4) The dependency structures between containerized application components are not considered during resource provisioning. Although some existing studies [19–22] address this issue to a certain degree by leveraging machine learning methods, their ML models are only feasible for traditional cloud-based applications and relatively simple for containerized workload scenarios. For instance, containerized microservice-based applications are more lightweight and decentralized in nature, compared with traditional monolithic applications. There are internal connections between different microservice units within an application. Critical components in a microservice architecture are the dependencies of most other microservices and more likely to suffer from **service level objectives** (**SLO**) violations due to higher resource demands and communication costs [23, 24]. These factors should all be taken into account during resource provisioning (details and solutions are discussed in Section 4.2.4).
(5) Current container orchestration methodologies mostly emphasize the evaluation of infrastructure-level metrics, while application-level metrics and specific QoS requirements are not receiving sufficient consideration. For example, containerized workloads may be attached with stricter time constraints compared with traditional cloud workloads, such as task deployment delays, task completion time, and communication delays [25, 26] (details and solutions are discussed in Section 4.2.2).

With such fast-growing complexity of application management in cloud platforms, cloud service providers have a strong motivation to optimize their container orchestration policies by leveraging ML techniques [27]. Through applying mathematical methods to training data, ML algorithms manage to build sophisticated analytic models that can precisely understand the behavior patterns in data flows or impacts of system operations. Thus, ML approaches could be adopted for modeling and prediction of both infrastructure-level or application-level metrics, such as application performance, workload characteristics, and system resource utilization. These insights could further assist the Container Orchestrators to improve the quality of resource provisioning decisions. On the other hand, ML algorithms could directly produce resource management decisions instead of heuristic methods, offering higher accuracy and lower time overhead in large-scale systems [28–31].

## 1.2 Motivation of Research

ML-based container orchestration technologies have been leveraged in cloud computing environments for various purposes, such as resource efficiency, load balance, energy efficiency, and SLA assurance. Therefore, we aim at investigating them in depth in this article:

Table 1. A Comparison of Our Work with Existing Surveys Based on Key Parameters

| Ref. | Application Deployment Unit | | Infrastructure | | | Classfication Schemes | | | | Investigated Problem |
|---|---|---|---|---|---|---|---|---|---|---|
| | VM | Container | Cloud | Fog | Edge | Application Architecure | Behavior Modeling and Prediction | Resource Provisioning | ML-based Orchestration Strategies | |
| [32] | ✓ | ✓ | ✓ | | | | | ✓ | | How to design effective cloud orchestration techniques to cope with large-scale heterogeneous cloud? |
| [33] | ✓ | ✓ | ✓ | | | | | ✓ | | How to apply brownout to support adaptive management of resources and applications in cloud? |
| [34] | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | How to employ machine learning techniques to achieve reliable resource provisioning in distributed computing environment? |
| [35] | ✓ | | ✓ | | | | | ✓ | | How to develop an autonomic resource scheduling technique for cloud resources based on users' QoS requirements? |
| [5] | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | How containers are used in high performance computing, big data analytics and geo-distributed applications? |
| [36] | | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | | How are container-based approaches applied in cloud activities? |
| [13] | | ✓ | ✓ | | | | | ✓ | | How different container orchestration systems can offer resources and achieve specific requirements of applications? |
| This review | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | How machine learning algorithms can be applied to optimize resource usage from container orchestration perspective? |

(1) The ML-based container orchestration technologies have shown promise in application deployment and resource management in cloud computing environments. Hence, we aim at outlining the evolution and principles of ML-based container orchestration technologies in existing studies.
(2) We recognize the need for a literature review to address the status of ML-based container orchestration researches in such fast-evolving and challenging scenarios. Furthermore, we investigate and classify the relevant articles by their key features. Our goal is to identify potential future directions and encourage more efforts in advanced research.
(3) Although an innovative review is proposed on container orchestration in Reference [13], the research on this field is growing continually by leveraging ML models. Therefore, there is a need for fresh reviews of machine learning-based container orchestration approaches to find out the advanced research challenges and potential future directions.

## 1.3 Our Contributions

The main contributions of this work are:
(1) We introduce a taxonomy of the most common ML algorithms used in the field of container orchestration.
(2) We present a comprehensive literature review of the state-of-the-art ML-based container orchestration approaches, and demonstrate the evolution of ML-based approaches used in container orchestration in recent years.
(3) We classify the reviewed orchestration methods by their key characteristics and conditions.
(4) We identify the future directions of ML-based container orchestration technologies.

## 1.4 Related Surveys

As summarized in Table 1, some previous surveys have already explored the field of application management in cloud computing environments while focusing on different research problems. Weerasiri et al. [32] introduce a comprehensive classification framework for analysis and evaluation of cloud resource orchestration techniques, while this work mainly focuses on resource provisioning perspectives without discussing ML-based approaches. Xu and Buyya [33] survey brownout technologies for adaptive application maintenance in cloud computing systems without discussing orchestration techniques. Duc et al. [34] look into the research challenge of resource management and performance optimization under edge-cloud platforms, with emphasis on workload modeling and resource management through ML techniques, where container orchestration is not their focus. Singh and Chana [35] depict a classification of QoS-aware autonomic resource provisioning under cloud computing environments through the methodical analysis of related research. However, this work is designed for managing general cloud workloads without adequate analysis of the key characteristics of containerized applications.

Furthermore, some recent studies also investigate different scopes of container orchestration techniques. Rodriguez and Buyya [13] propose a systematic review and taxonomy of the mainstream container orchestration systems by classifying their features, system architectures, and management strategies. This work pays more attention to system modeling and design perspectives rather than detailed orchestration strategies. Casalicchio and Iannucci [5] conduct an extensive literature review of the state-of-the-art container technologies, focusing on three main issues, including performance evaluation, orchestration, and cyber-security. Pahl et al. [36] present a survey and taxonomy of cloud container technologies with a systematic classification of the existing researches. Nonetheless, the research direction of machine learning-based orchestration for containerized applications has not been explored with systematic categories in any existing survey and taxonomy.

Compared to them, our article extends the previous surveys by focusing on how machine learning algorithms can be applied to solve complex research problems from a container orchestration perspective, such as multi-dimensional workload characterization or autoscaling in hybrid clouds. We emphasize the diversity and complexity of orchestration schemes under specific application architectures and cloud infrastructures. Furthermore, it also specifies the extensive research questions and potential future directions of machine learning-based container orchestration techniques.

### 1.5 Article Structure

The rest of this article is organized as follows: Section 2 introduces the background of machine-learning and container orchestration. Section 3 describes an overview of machine learning-based container orchestration technologies, followed by the categories and taxonomy of the existing approaches. A description of the reviewed approaches and their mappings to the categories are presented in Section 4. Section 5 summarizes a discussion the future directions and open challenges. Finally, the conclusion of this work is given in Section 6.

## 2 BACKGROUND

In this section, we present a comprehensive taxonomy of the existing ML models utilized in the area of container orchestration and a brief introduction of the high-level container orchestration framework.

### 2.1 Machine-Learning

Machine-learning is a general concept describing a set of computing algorithms that can learn from data and automatically build analytical models through data analysis [37]. One of its fundamental objectives is to build mathematical models that can emulate and predict the behavior patterns of various applications through training data. Machine-learning has gained immense popularity through the past decades, widely accepted in many research areas such as image recognition, speech recognition, medical diagnose, and smart building [38, 39]. With the continuously growing

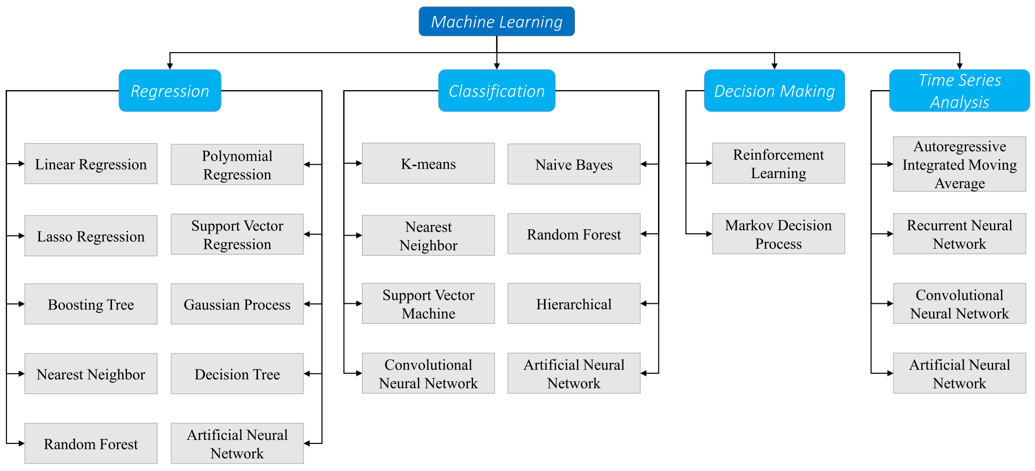


Fig. 1. Machine-learning taxonomy in the context of container orchestration by optimization objectives.

computational power and adoption of GPUs in warehouse-scale datacenters, the capability and data scale of machine learning technologies are still soaring [40].

As depicted in Figure 1, we design a machine learning taxonomy through classifying some of the most popular ML models in the field of container orchestration by their optimization objectives:

(1) Regression algorithms predict a continuous output variable through analysis of the relationship between the output variable and one or multiple input variables. Popular regression algorithms (e.g., **support vector regression** (**SVR**), random forest, and polynomial regression) are usually used to explore and understand the relation between different performance metrics.
(2) Classification methods categorize training data into a series of classes. The use cases of classification mainly include anomaly detection and dependency analysis. For example, K-means is broadly adopted for the identification of abnormal system behaviors or components [41, 42], while **support vector machine** (**SVM**) can be leveraged for decomposing the internal structure of containerized applications and identifying key application components [24].
(3) Decision making models generate resolutions by simulating the decision making process and identifying the choices with the maximized cumulative rewards [43]. As the most common algorithms in this category, RL models, including model-free (e.g., Q-Learning and Actor-Critic) and model-based RL that belong to the reactive mechanism used to react to the current environment, have been widely employed for decision making in resource provisioning.
(4) Time series analysis is the predictive approach that achieves pattern recognition of time series data and forecasts of future time series values or trends from past values. Ranging from **autoregressive integrated moving average** (**ARIMA**) to more advanced **recurrent neural network** (**RNN**) models, such algorithms are useful for behavior modeling of sequential data, including workload arrival rates or resource utilization.

To be noted, some ML models can be utilized under multiple optimization scenarios. For example, **artificial neural network** (**ANN**) models can be applied for time series analysis of resource utilization or regression of application performance metrics [28, 44].

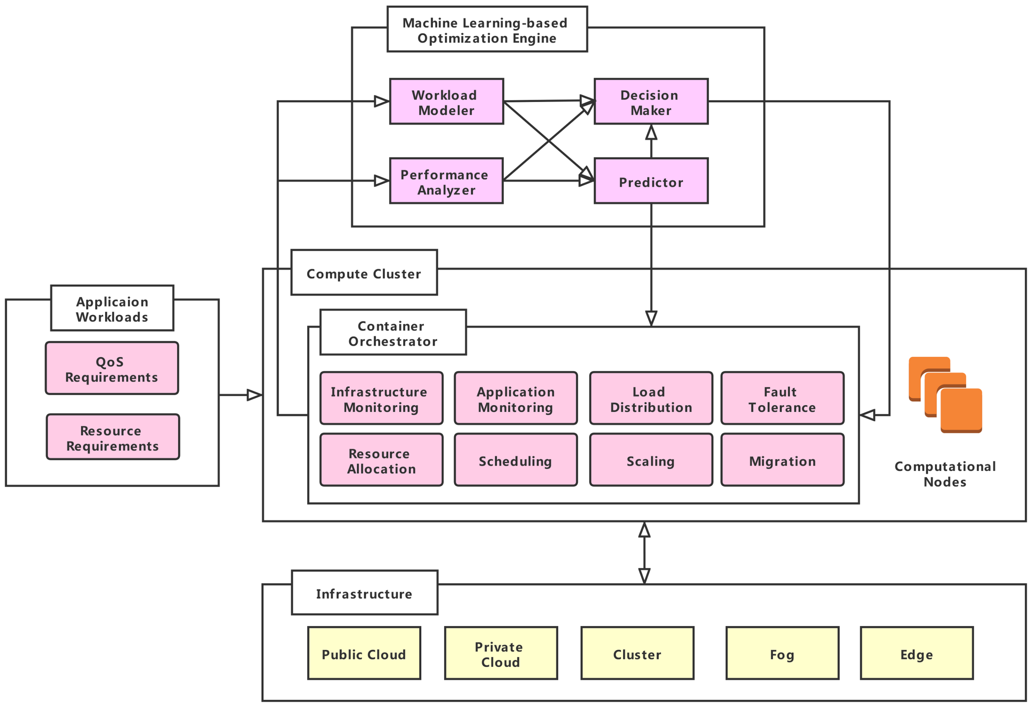


Fig. 2. A high-level machine learning-based container orchestration framework reference architecture.

## 2.2 Container Orchestration

Container orchestration empowers cloud service providers to decide how containerized applications are configured, deployed, and maintained under cloud computing environments [5]. It is targeted at automatic application deployment and dynamic configuration adjustment at runtime for a diverse range of workloads. Figure 2 demonstrates a reference architecture of machine learning-based container orchestration frameworks, where the components included are common to most existing systems. A detailed description of each layer is given as below:

*2.2.1 Application Workloads.* Workloads are generally defined as the requests submitted by users to an application or the Orchestrator. Workloads could be classified into many categories from different perspectives according to their unique features, such as long-running services or short-living batch jobs (classified by execution time); computation-intensive, data-intensive, or memory-sensitive workloads (classified by resource usage pattern) [3, 4, 45]. Each workload is also defined with multi-dimensional resource requirements (e.g., CPU, memory, disk, network) and QoS requirements (e.g., time constraints, priorities, throughput). The extremely dynamic and unpredictable feature of heterogeneous workloads greatly increases the complexity of orchestration mechanisms.

*2.2.2 Compute Cluster.* A compute cluster is a group of virtual or physical computational nodes that run in a shared network. As the core component in a containerized cluster, the Container Orchestrator is responsible for assigning containerized application components to worker nodes where containers are actually hosted and executed. Its major functionalities involve:

(1) Resource Allocation assigns a specific amount of resources to a container. Such configurations and limitations are basic metrics in managing container placement and resource isolation control between containers.
(2) Scheduling defines the policy for the initial placement for one or a group of containers, by considering conditions such as resource constraints, communication costs, and QoS requirements.
(3) Scaling is the resource configuration adjustment of containerized applications or computational nodes in response to any potential workload fluctuations.
(4) Migration is designed as a complementary mechanism to prevent severe infrastructure-level resource overloading or resource contention between co-located applications by relocating one or a group of containers from one node to another.
(5) Infrastructure Monitoring keeps the track of infrastructure-level metrics of the cluster, such as the number of nodes, node states, resource usage of each node, and network throughput. Such metrics are the critical information for evaluating the health conditions of the cluster and making precise decisions in the above resource source provisioning layers.
(6) Application Monitoring does not only periodically check the application states but also records their resource consumption and performance metrics (e.g., response time, completion time, throughput, and error rates). This information serves as the crucial reference data in anomaly detection and SLA violation measurement.
(7) Load Distribution, as the core mechanism for load balancing under containerized environments, distributes network traffic between containers evenly with policies such as Round-Robin [46]. It plays an important role in improving system scalability, availability, and network performance.
(8) Fault Tolerance ensures the high availability of the system through replica control. Each container is maintained with a configurable number of replicas across multiple nodes in case of a single point of failure. It is also possible to have multiple Orchestrators to deal with unexpected node crashes or system overloading.

*2.2.3 Infrastructure.* Thanks to the high portability, flexibility, and lightweight nature of containers, it allows containers to be deployed across a multitude of infrastructures, including private clouds, public clouds, fog, and edge devices. Similar to traditional server farms, private clouds provide exclusive resource sharing within a single organization based on its internal datacenter [1]. By contrast, public cloud service providers support on-demand resource renting from their warehouse-scale cloud datacenters.

Since applications and workloads are all deployed and processed at cloud data centers in traditional cloud computing, this requires a massive volume of data transferred to cloud servers. As a consequence, it leads to significant propagation delays, communication costs, bandwidth, and energy consumption. Through moving computation and storage facilities to the edge of a network, fog and edge infrastructures can achieve higher performance in a delay-sensitive, QoS-aware, and cost-saving manner [47, 48]. With the requests or data directly received from users, fog or edge nodes (e.g., IoT-enabled devices, routers, gateways, switches, and access points) can decide whether to host them locally or send them to the cloud.

*2.2.4 Machine Learning-based Optimization Engine.* An ML-based optimization engine builds certain ML models for workload characterization and performance analysis, based on analysis of monitoring data and system logs received from the Orchestrator. Furthermore, it can produce future resource provisioning decisions relying on the generated behavior models and prediction results. The engine can be entirely integrated or independent from the Orchestrator. Its major components are listed as follows:

(1) Workload Modeler is designed for ML-based workload characterization through analyzing the input application workloads and identifying their key characteristics.
(2) Performance Analyzer generates application and system behavior models through applying ML algorithms to application and infrastructure-level monitoring data acquired from the Orchestrator.
(3) Predictor forms forecasts of workload volumes or application/system behaviors, relying on the models obtained from Workload Modeler and Performance Analyzer. The prediction results could be sent either to the Orchestrator or Decision Maker.
(4) Decision Maker combines the behavior models and prediction results received from the above components with certain ML-based optimization methods/schemes to further generate precise resource provisioning decisions that are fed back to the Orchestrator.

These components are common to most container orchestration systems; however, not all of them have to be implemented to make the whole system functional. According to different design principles, the engine can alternatively produce multi-level prediction results to assist the original Orchestrator in making high-quality decisions in resource provisioning, or directly generate such decisions instead of the Orchestrator.

## 3 TAXONOMY OF MACHINE LEARNING-BASED CONTAINER ORCHESTRATION TECHNOLOGIES

Figure 3 presents a taxonomic classification of the literature reviewed in our work. Application Architecture describes the behaviors and internal structures of containerized application components that together perform as a whole unit. Infrastructure indicates the environments or platforms where the applications operate. Objectives are the improvements that the proposed ML-based approaches attempt to achieve. Behavior Modeling and Prediction leverage ML models for pattern recognition and simulation of system and application behaviors, as well as forecasting future trends according to previously collected data. Resource Provisioning specifies the ML-based or heuristic policies for resource management of containerized applications at different phases in a container lifecycle under various scenarios.

### 3.1 Application Architecture

This section presents the most common application architectures that define the composition of containerized applications and how they are deployed, executed, and maintained. The application architecture also decides the minimum orchestration unit that the ML-based approaches will operate.

*3.1.1 Monolithic.* Monolithic applications follow an all-in-one architecture where all the functional modules are developed and configured into exactly one deployment unit, namely, one container. Such applications could be initially easy to develop and maintain at a small scale. For example, Microsoft Azure [49] still supports automated single-container-based application deployment for enterprize solutions where the business logic is not feasible for building complex multi-component models. However, the consistent development and enrichment of monolithic applications would inevitably lead to incremental application sizes and complexity [50]. Consequently, the maintenance costs can dramatically grow in continuous deployment. Even modification of a single module requires retesting and redeployment of the whole application. Furthermore, scaling of monolithic applications means replication of the entire deployment unit containing all the

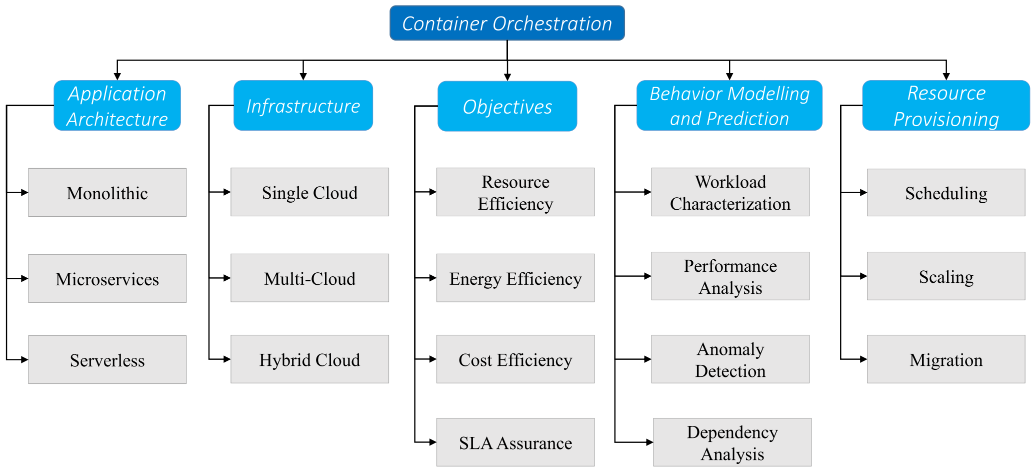
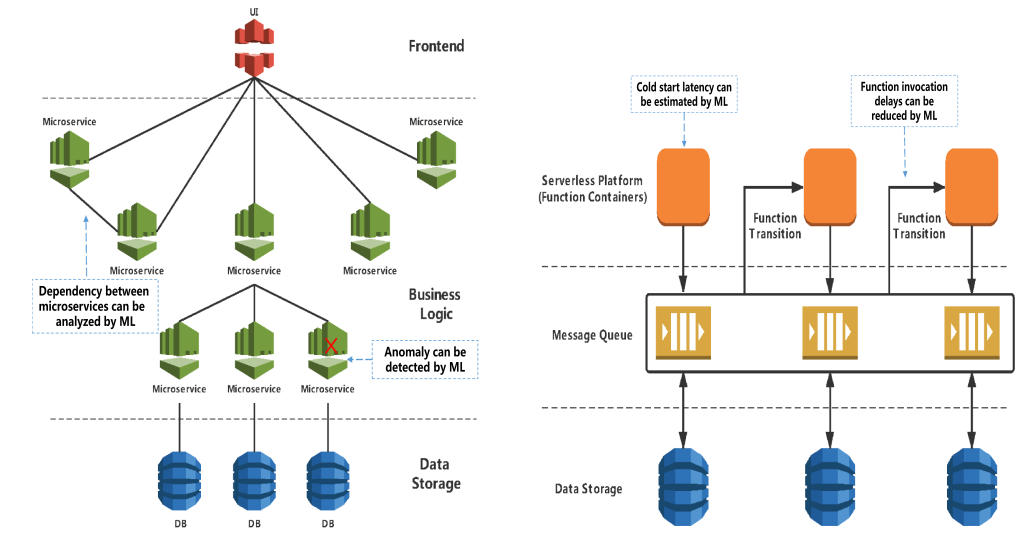


Fig. 3. Machine learning-based container orchestration taxonomy.


UI
Frontend
Microservice
Microservice
Microservice
Microservice
Microservice
Dependency between microservices can be analyzed by ML
Business Logic
Microservice
Microservice
Microservice
Anomaly can be detected by ML
Data Storage
DB
DB
DB
Cold start latency can be estimated by ML
Function invocation delays can be reduced by ML
Serverless Platform (Function Containers)
Function Transition
Function Transition
Message Queue
Data Storage

(a) Microservice Architecture and Potential Usage of ML (b) Serverless Architecture and Potential Usage of ML

Fig. 4. Examples of the architectures.

modules [51]. In most scenarios, only a proportion of the modules need to be scaled due to resource shortage. Therefore, scaling the whole application would lead to poor overall resource efficiency and reliability. Thus, the monolithic architecture is only suitable for small-scale applications with simple internal structures.

*3.1.2 Microservice.* To address the problem of high development and maintenance costs caused by colossal application sizes, the **microservice architecture** (**MSA**) is proposed to split single-component applications into multiple loosely coupled and self-contained microservice components [52]. An example of MSA is given in Figure 4(a). Each microservice unit can be deployed and operated independently for different functionalities and business objectives. Furthermore, they can interact with each other and collaborate as a whole application through lightweight communication methods such as **representational state transfer** (**REST**). Through decomposing applications into a group of lightweight and independent microservice units, MSA has significantly reduced the costs and complexity of application development and deployment. Nonetheless, the growing number of components and dynamic inter-dependencies between microservices in MSA raises the problem of load distribution and resource management at the infrastructure level. A well-structured orchestration framework for MSA should be able to maintain multiple parts of an application with SLA assurance, granting more control over individual components. These challenges can be addressed by ML-based approaches, for example, utilizing the ML-based approach to analyze dependencies between different microservices and resource usage of isolated microservices.

*3.1.3 Serverless.* The serverless architecture defines an event-driven application paradigm that undertakes stateless computational tasks, namely, serverless functions. Designed to perform certain user-defined functionalities, functions are usually short pieces of code hosted in function containers with specific configurations and limited execution time. To ensure the successful completion and performance requirements of functions, serverless platforms are responsible for managing the function execution environments in terms of resource provisioning, energy consumption, SLA assurance, and security [53, 54]. Compared with the microservice that tends to support continuous requests with long-running lifecycles, the serverless is the event-driven handler for ephemeral tasks with finer granularity. As depicted in Figure 4(b), a typical serverless application is represented in the form of a function chain consisting of a workflow of individual functions. Within a function chain, interactions and transitions between functions are conducted through a centralized messaging service, such as a message queue or event bus [55]. The orchestration problem under such context translates into managing the invocation of function chains in terms of function initialization, execution, transition, and data flow control. Several previous studies have proved that the significant time overhead of function initialization is currently the major performance bottleneck in function invocation [56, 57]. Therefore, the current research focus of ML-based solutions in this field is on minimization of function invocation delays and resource wastage [58–60]. Currently, as there is no prevailing technology (e.g., containers for MSA) to support the handlers in the serverless platform, the serverless utilizes containers to realize their platform, which may experience performance slowdown [61].

## 3.2 Infrastructure

A cloud infrastructure consists of a set of hardware resources and virtualization software to deliver virtualized resources to users for application deployment. The execution of ML-based approaches also needs to consume the resources provisioned by infrastructure. In general, we have identified three types of cloud infrastructures in the context of container orchestration:

(1) A single cloud environment is built on resources from only one cloud service provider (either a private or public cloud) to host and serve all the applications.
(2) A multi-cloud environment includes multiple cloud services (e.g., private clouds, public clouds, or a mixture of both). As different cloud service providers may differ in many aspects, such as resource configurations, price, network latency, and geographic locations, this allows more choices for optimization of application deployment.
(3) A hybrid cloud environment is composed of a mixture of private clouds, public clouds, fog, or edge devices. It is not always efficient to deploy all the applications and data to public/private clouds, considering the data transmission time and network latency from end users to cloud

servers. To solve this issue, the hybrid cloud enables applications and data to be deployed and processed at fog or edge devices that are close to end users.

## 3.3 Optimization Objectives

In light of the diversity of application architectures and cloud infrastructures, many types of metrics have been considered as optimization objectives during the behavior modeling and resource provisioning of containerized applications. As shown in Figure 3, we group the existing objective metrics into four major categories. Since an orchestration solution usually needs to achieve multiple objectives according to specific user requirements, balancing the tradeoff between different optimization objectives remains a key concern in automated application deployment and maintenance.

(1) **Resource Efficiency.** Infrastructure-level resource usage metrics are usually treated as key application performance indicators for energy and cost efficiency. They are the fundamental data source of most behavior modeling schemes, such as prediction of the resource demands of coming workloads, or discovery of the relationship between resource usage patterns and other performance metrics. Such insights could be further applied in decision making of resource provisioning to improve the overall resource efficiency and application performance.
(2) **Energy Efficiency.** Under the continuously growing scale of cloud data centers, the tremendous electricity usage consumed by cloud infrastructures has emerged as a critical concern in the field of cloud computing [62]. Therefore, various approaches have been proposed to minimize energy consumption and optimize energy efficiency [33, 41, 58, 63]. As the overall electricity usage of a system is estimated as the summation of the consumption by each **physical machine (PM)** where its energy usage is directly related to its resource utilization, an essential way to control energy efficiency is to adjust and balance the resource utilization of physical machines during resource provisioning.
(3) **Cost Efficiency.** Following the pay-as-you-go payment model of mainstream cloud service providers, market-based solutions regard cost efficiency as one of their principal targets [6, 15, 29, 64]. Through evaluation and selection of diverse cloud services according to their pricing models and computing capability, an optimized orchestration solution aims at minimizing the overall financial costs while satisfying the QoS requirements defined by users.
(4) **SLA Assurance.** Containerized applications are mostly configured with specific performance requirements, such as response time, initialization time, completion time, and throughput. These constraints are mostly expressed as SLA contracts, while their violations could lead to certain penalties. Because of the dynamic and unpredictable feature of cloud workloads, autoscaling is usually leveraged to automate application maintenance with SLA assurance [8, 28, 29, 60, 65–67], in response to the frequently changing workloads.

## 3.4 Behavior Modeling and Prediction

Behavior modeling is a fundamental step in understanding the overall application or system behavior patterns through analysis of application/infrastructure-level metrics. The variety of multi-layer metrics related to workloads, application performance, and system states significantly complicates the modeling process. Nonetheless, a well-structured behavior model that can produce precise prediction results based on ML approaches, which is also apparently useful for achieving certain optimization objectives during orchestration. The behavior modeling and prediction can also provide important information for the decision making in the resource provisioning phase.

(1) Workload characterization captures the key features of application workloads. Because of the dynamic and decentralized nature of containerized applications like microservices, the

received workloads may differ in many ways, such as task structures, resource demands, arrival rates, and distributed locations. These factors make it hard to define a robust method for characterization and categorization of all the different workloads within an orchestration system. However, the knowledge of workload behaviors is necessary to improve the quality of resource provisioning decisions by making precise resource assignments in response to any incoming workloads.

(2) Performance analysis discovers the relation among infrastructure (e.g., resource utilization and energy consumption) or application-level (e.g., response time, execution time, and throughput) metrics to depict the system states and application performance. These insights are important in managing the tradeoff between different optimization objectives.

(3) Anomaly detection classifies and identifies abnormal system behaviors, including security threats, instance failure, workload spikes, performance downgrade, and resource overloading. Such anomalies could severely harm the system availability and reliability. Therefore, fast and accurate localization of their root causes could prevent SLA violations or system crashes.

(4) Dependency analysis looks into the graph-based internal dependencies between containerized application components. It helps to understand the workload distribution/pressure among application components and make precise resource configurations. Since the dependencies may be dynamically updated at runtime, it requires an incremental model that can consistently adjust itself and address the chain reactions of individual components to the overall application performance.

## 3.5 Resource Provisioning

Considering the various application architectures, infrastructures, and optimization objectives discussed in the above sections, resource provisioning for containerized applications has become much more challenging. The diversity of cloud workloads, resource heterogeneity within hybrid cloud environments, and complex application internal structures should be all assessed during resource provisioning. Therefore, the state-of-the-art resource provisioning strategies are commonly relying on the decisions on the amount of provisioned resources, which can exploit ML-based approaches to achieve higher accuracy and shorter computation delays. Based on our investigation, the Resource Provisioning category in Figure 3 can be classified into more detailed categories, we provide Figure 5 to complement Figure 3, where the resource provisioning operations can be further classified into the following categories:

(1) Scheduling decides the initial placement of a containerized task unit, which could consist of a single task, a group of independent tasks, or a group of dependent tasks in graph-based structures. Due to the variety of application architectures and task structures, application deployment policies should consider a wide range of placement schemes. The quality of scheduling decisions has a direct impact on the overall application performance and resource efficiency [9].

(2) Scaling is the size adjustment of containerized applications or computational nodes in response to any potential workload fluctuations, which ensures the applications supported with enough resources to minimize SLA violations. Horizontal scaling adjusts the number of container replicas belonging to the applications or the number of nodes. By contrast, vertical scaling only updates the amount of resources assigned to existing containers or nodes. Moreover, hybrid scaling combines both horizontal and vertical scaling to produce an optimized solution.

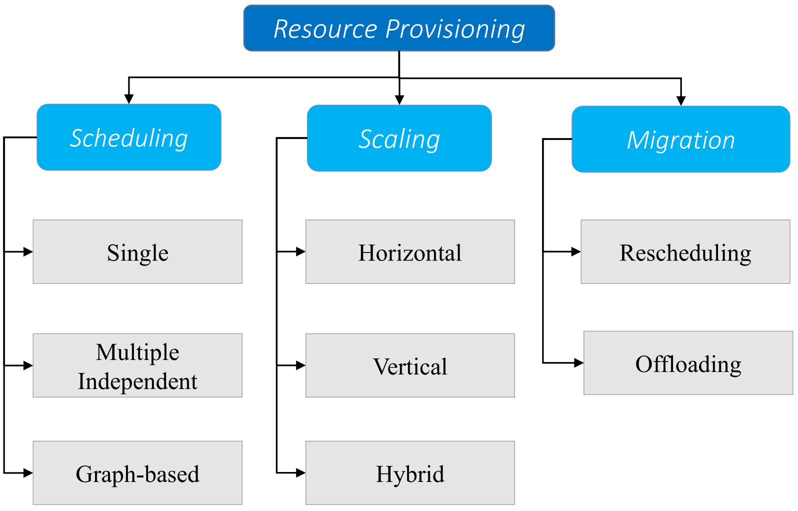


Fig. 5. Resource provisioning taxonomy.

(3) Migration is the relocation of one or a group of tasks from one node to another. When resource contention, overloading, or underloading occur between co-located applications due to poor scheduling decisions, rescheduling is triggered for load balancing through task migration within a single cloud. On the other hand, computation offloading manages the migration of computation-intensive tasks that are experiencing resource bottlenecks to devices with enough demanded resources across hybrid cloud environments. Targeted to improve the application performance, a refined offloading policy should be able to pick a suitable relocation destination that minimizes the migration costs in terms of execution time, bandwidth, and energy consumption.

## 3.6 Evolution of Machine Learning-based Container Orchestration Technologies

The optimization of the objectives and metrics of ML-based approaches for container orchestration has been investigated and multiple methods have been proposed over the years. To show the evolution and development of ML-based approaches for container orchestration in recent years. Figure 6 demonstrates the evolution of ML-based models since 2016, with emphasis on their objectives and metrics. As the research related to machine-learning for container orchestration starts from 2016, our examination for the evolution falls between 2016 and 2021.

In 2016, the ARIMA [68] and **nearest neighbor** (**NN**) [18] algorithms were already leveraged for resource utilization prediction of containerized applications. ARIMA is a dynamic stochastic process proposed in the 1970s, which has been used for forecasting time series showing non-stationarity by identifying the seasonal differences. NN is a proximity search approach that can find a candidate that is closest to a given point. As ARIMA and NN have been used widely in time series prediction, they were firstly applied in container orchestration to predict resource utilization, such as CPU, memory, and I/O. However, at this stage, the application models were relatively simple, which only considered the time series pattern of infrastructure-level resource metrics.

In 2017, Shah et al. [69] first adopted the **long short-term memory** (**LSTM**) model for dependency analysis of microservices. The LSTM is an approach based on neural network and well suited to classifying, processing, and forecasting based on time series data. Compared with traditional feedforward neural network, LSTM also has feedback connections to enhance its performance,

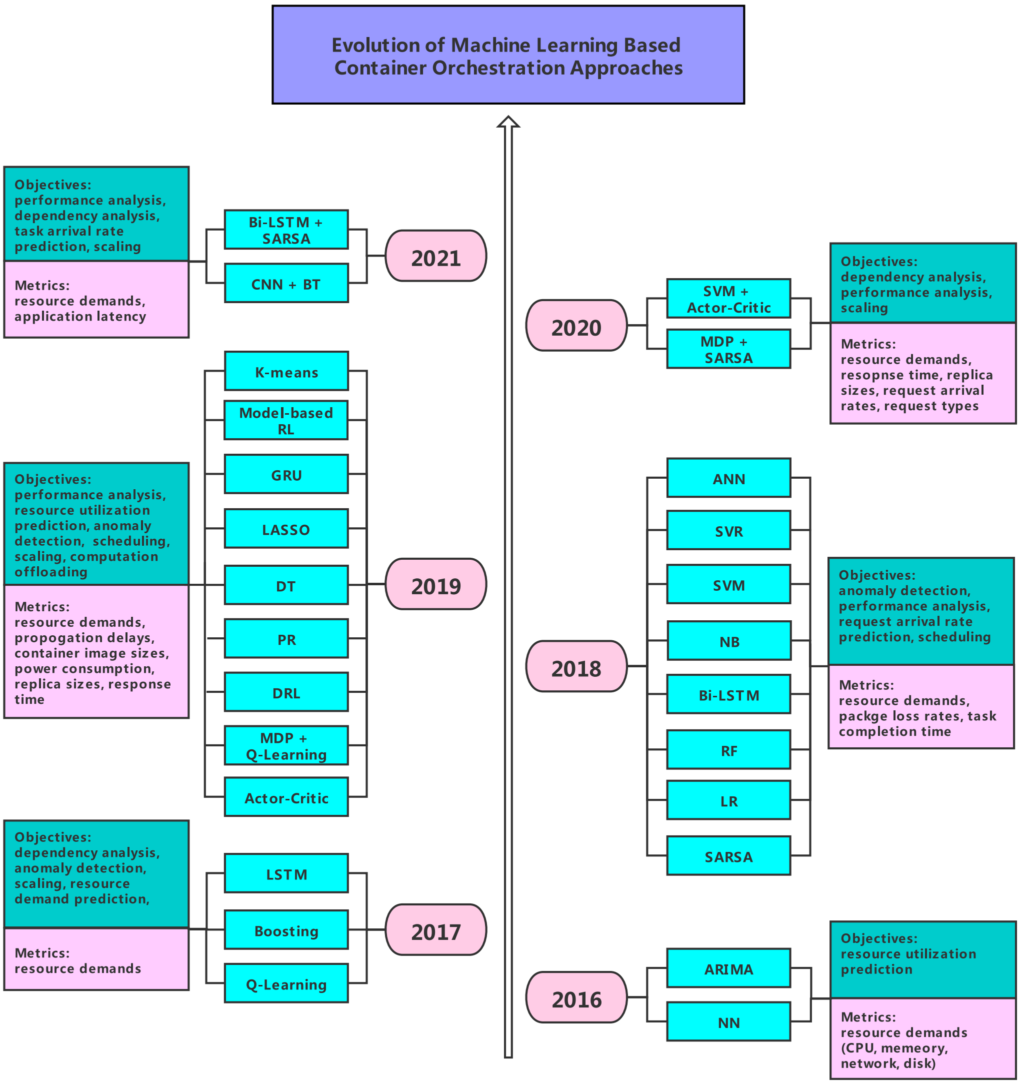

Evolution of Machine Learning Based Container Orchestration Approaches
2021
Bi-LSTM + SARSA
CNN + BT
Objectives: performance analysis, dependency analysis, task arrival rate prediction, scaling
Metrics: resource demands, application latency
2020
SVM + Actor-Critic
MDP + SARSA
Objectives: dependency analysis, performance analysis, scaling
Metrics: resource demands, resopnse time, replica sizes, request arrival rates, request types
2019
K-means
Model-based RL
GRU
LASSO
DT
PR
DRL
MDP + Q-Learning
Actor-Critic
Objectives: performance analysis, resource utilization prediction, anomaly detection, scheduling, scaling, computation offloading
Metrics: resource demands, propogation delays, container image sizes, power consumption, replica sizes, response time
2018
ANN
SVR
SVM
NB
Bi-LSTM
RF
LR
SARSA
Objectives: anomaly detection, performance analysis, request arrival rate prediction, scheduling
Metrics: resource demands, packge loss rates, task completion time
2017
LSTM
Boosting
Q-Learning
Objectives: dependency analysis, anomaly detection, scaling, resource demand prediction,
Metrics: resource demands
2016
ARIMA
NN
Objectives: resource utilization prediction
Metrics: resource demands (CPU, memeory, network, disk)

Fig. 6. Evolution of machine learning-based container orchestration technologies.

which has functioned well in handwriting recognition and speech recognition. The model in [69] evaluated both the internal connections between microservice units and the time series pattern of resource metrics. Furthermore, anomaly detection was built on top of the LSTM model for the identification of abnormal behaviors in resource utilization or application performance. Besides, Cheng et al. [70] used **Gradient Boosting Regression** (**GBR**) that can ensemble multiple weak prediction models (e.g., regression trees) to form a more powerful model, and applied GBR for resource demand prediction in workload characterization.

A model-free RL method, namely, Q-Learning, is also used for scaling microservices. Q-learning works by learning the action-value function to evaluate the reward of taking an action in a particular state. The benefit of Q-learning is that it can achieve the expected reward without the model of the environment. Xu et al. [29] leveraged Q-Learning to produce vertical scaling plans. An optimal scaling decision was targeted to minimize resource wasting and computation costs under the assumption of SLA assurance.

In 2018, Tang et al. [71] employed the **bidirectional LSTM** (**Bi-LSTM**) model for the prediction of workload arrival rates and application throughput. Their training module had demonstrated significant accuracy improvement over ARIMA and LSTM models in terms of time series prediction. Ye et al. [44] applied a series of traditional regression methods based on the statistical process for relationship estimation, including SVR, **linear regression** (**LR**), and modern ANN based on deep learning, to conduct performance analysis of relevant resource metrics. They attempted to evaluate the relationship between resource allocation and application performance. However, only single-component applications with limited performance benchmarks were considered within the scope of their work.

Du et al. [42] designed an anomaly detection engine composed of traditional machine learning methods including **k-nearest neighbors** (**KNN**), SVM, **Naive Bayes** (**NB**), and **random forest** (**RF**), to classify and diagnose abnormal resource usage patterns in containerized applications. Orhean et al. [25] utilized **state–action–reward–state–action** (**SARSA**), a model-free RL algorithm similar to Q-learning, to manage the graph-based task scheduling problem in **directed acyclic graph** (**DAG**) structures, aiming at minimizing the overall DAG execution time.

In 2019, Cheng et al. [72] proposed a hybrid **gated recurrent unit** (**GRU**) model to further reduce the computational costs and error rates of resource usage prediction of cloud workloads. GRU is considered as an optimized model of LSTM [73], as LSTM models are relatively complex with high computational costs and data processing time. An LSTM cell structure consists of three gates, including the input gate, the forget gate, and the output gate. GRU simplifies this structure and achieves higher computational efficiency by integrating the input gate and the forget gate into one update gate.

Performance analysis of containerized applications was also explored in depth. Venkateswaran and Sarkar [16] leveraged the K-means clustering and **polynomial regression** (**PR**) to classify the multi-layer container execution structures under multi-cloud environments by their feasibility to the application performance requirements. Both K-means and polynomial regression can find groups that have not been explicitly labeled in the data, which are originally used in signal processing and applied to identify the execution structures of containers. According to workload arrival rates and resource metrics, Podolskiy et al. [74] applied Lasso regression (LASSO) to forecast **service level indicators** (**SLI**) in terms of application response time and throughput for Kubernetes private cloud. LASSO can generate a linear model via variable selection and regularization to improve prediction accuracy. Dartois et al. [75] used the **decision tree** (**DT**) regression algorithm to analyze the **solid state drive** (**SSD**) I/O performance under interference between applications. The DT can break down the dataset into small subsets and an associated decision tree can be incrementally generated, which is easy to interpret, understand and virtualize. In addition, DT is not very sensitive to outliers or missing data, and it can handle both categorical and numerical variables.

As for resource provisioning, **deep reinforcement learning** (**DRL**) was first applied in the context of task scheduling [26, 31]. Bao et al. [26] designed a DRL framework for the placement of batch processing jobs, where an ANN model represented the mapping relationship between workload features, system states, and corresponding job placement decisions. The Actor-Critic RL algorithm was selected to train the ANN model and generate optimal scheduling decisions that minimized the performance interference between co-located batch jobs. Compared with traditional heuristic scheduling policies like bin packing, their solution demonstrated remarkable performance improvement on the Kubernetes cluster regarding overall job execution time. Moreover, DRL was also employed to solve the problem of computation offloading under fog-cloud environments in Reference [31]. On top of a **Markov decision process** (**MDP**) model that simulated the interactions of the offloading process at a large scale, the deep Q-Learning method optimized the migration decisions by minimization of the time overhead, energy usage, and computational costs. To explore the efficiency of hybrid scaling mechanisms, Rossi et al. [76, 77] leveraged model-based RL models to compose a mixture of horizontal and vertical scaling operations for monolithic applications, aiming at minimizing the resource usage, performance degradation, and adaption costs.

In 2020, several RL-based scaling approaches were proposed in the form of hybrid ML models [24, 28, 30]. Qiu et al. [24] adopted the SVM model for dependency analysis of microservices and recognition of the key components that are highly likely to experience resource bottlenecks and performance downgrade. To prevent severe **service level objectives** (**SLO**) violations, the Actor-Critic method was utilized to generate the appropriate resource assignment decisions for these components through horizontal scaling. The approach has been validated and executed on the Kubernetes cluster with significant performance improvement compared with Kubernetes's autoscaling approach. Besides, Sami et al. [30] combined MDP and SARSA models to build a horizontal scaling solution for monolithic applications under fog-cloud environments. SARAS produced the optimized scaling decisions through model training relying on the MDP model that simulated the scaling scenarios with the fluctuating workloads and resource availability in fog taken into account.

In 2021, Zhang et al. [23] proposed a novel approach composed of a **convolutional neural network** (**CNN**) and **boosted trees** (**BT**) for dependency and performance analysis of microservices. Their CNN model did not only analyze the inter-dependencies between microservice units for system complexity navigation, but also the time series metrics related to application performance. Furthermore, the BT model is responsible for the prediction of long-term QoS violations. To further improve the speed and efficiency of RL-based scaling approaches for microservices under hybrid cloud environments, Yan et al. [78] developed a multi-agent parallel training module based on SARSA and improved the horizontal scaling policy of Kubernetes, supported by the microservice workload prediction results generated by Bi-LSTM.

Overall, diverse ML algorithms have been utilized in the context of container orchestration, ranging from workload modeling to decision making through RL. However, there are not many new ML models adopted in the area of container orchestration in recent years. To further improve prediction accuracy and computational efficiency, the emerging trend of hybrid ML-based solutions targets to combine multiple existing ML methods to form a complete orchestration pipeline, including multi-dimensional behavior modeling and resource provisioning. The evolution of ML models also contributes to the extension of various application architectures and cloud infrastructures.

## 4 STATE-OF-THE-ART IN MACHINE LEARNING FOR ORCHESTRATION OF CONTAINERS

In this section, we introduce a literature review of machine learning-based container orchestration approaches. To stress the key features in the evaluated studies, we use the taxonomy in Section 3 to outline the key characteristics of the approaches designed for behavior modeling and prediction, as well as resource provisioning.

### 4.1 Article Selection Methodology

In this subsection, we introduce the approach we followed to reach the articles. The relevant articles have been broadly searched in the mainstream academic sources, including ACM Digital Library, IEEEXplore, Springer, Elsevier, ScienceDirect, Wiley Interscience, and Google Scholar. We

Table 2. Summary of Workload Characterization Approaches

| Ref. | Mechanism | Infrastructure | Application Architecture | Methods | Objectives | Advantages | Limitations |
|---|---|---|---|---|---|---|---|
| [58] | Time series analysis | Single cloud | Serverless | LSTM | Request arrival rate prediction | High prediction accuracy | Simplicity of application models |
| [8] | Classification | Single cloud | Monolithic | K-means++ | Resource demand prediction | High scalability | Limited accuracy under high load variance |
| [28] | Time series analysis | Single cloud | Monolithic | ARIMA | Request arrival rate prediction | Capability of large data scales | Inaccuracy under trend turning |
| [79] | Time series analysis | Single cloud | Monolithic | ARIMA-TES | Multi-dimensional workload prediction | High robustness, accuracy, and anti-interference | Higher time overhead |
| [80] | Time series analysis | Single cloud | Monolithic | LSTM, Bi-LSTM | Resource demand prediction | High prediction accuracy | Lack of consideration for anomaly detection |
| [72] | Time series analysis | Single cloud | Monolithic | GRU-ES | Resource demand prediction | Low error rates | Ignorance of potential resource allocation strategies based on the prediction model |
| [18] | Regression | Single cloud | Microservice | TSNNR | Resource demand prediction | Improved prediction accuracy | Limited workload scenarios |
| [70] | Regression | Single cloud | Microservice | GBR | Resource demand prediction | High prediction accuracy | Limited workload scenarios |
| [81] | Time series analysis | Single cloud | Microservice | LSTM | Request arrival rate prediction | Low time overhead | High computational expense |
| [82] | Time series analysis | Hybrid cloud | Microservice | IGRU-SD | Resource demand prediction | Low error rates | Unclear demonstration of the relationship between resource allocation and energy efficiency |
| [67] | Time series analysis | Single cloud | Microservice | AR, LSTM, HTM | Request arrival rate prediction | Multi-model optimization | High computational costs |
| [71] | Time series analysis | Single cloud | Microservice | Bi-LSTM | Prediction of request arrival rate and application throughput | Improved prediction accuracy | Implicit time overhead analysis |
| [78] | Time series analysis | Hybrid cloud | Microservice | Bi-LSTM | Task arrival rate prediction | Performance improvement | Inaccuracy in long-term forecasts |

focus the search criteria on titles and abstracts, starting with the following keywords: container orchestration, application component, microservice, serverless, machine-learning, deep learning, characterization, dependency analysis, anomaly detection, classification, prediction, monitoring, scheduling, scaling, migration, and load distribution. As there are many irrelevant articles in the search results, we further refine the results through secondary filtering and reviewing based on relevance and quality.

Eventually, 44 research articles are selected in the field of machine learning-based container orchestration technologies. The study distribution by publication sources includes 64% conference papers, 24% journal articles, and 12% symposium articles. The articles discussed in each class are key representative works selected from literature by several evaluation criteria, including robustness of application models, coverage of critical challenges in the field, accurate depiction of real-world orchestration scenarios, high scalability under complex cloud environments, novelty of orchestration approaches, refined combinations of ML-based models, and extensibility to hybrid clouds.

## 4.2 Behavior Modeling and Prediction

This section presents the approaches related to behavior modeling and prediction.

*4.2.1 Workload Characterization.* The knowledge of workload characteristics and behavior patterns offers important reference data for the estimation of resource provisioning and load distribution. Table 2 shows the existing studies that leverage ML techniques in workload characterization.

Many previous studies have tried to model the time series pattern of request arrival rates in containerized applications through various algorithms. ARIMA, as a classic algorithm for analyzing time series data, was utilized in Reference [28]. Compared with other linear models that are mainly suitable for linear datasets such as **autoregressive** (**AR**), **moving average** (**MA**), **autoregressive moving average** (**ARMA**), and **exponentially weighted moving average** (**EWMA**), ARIMA enjoys high accuracy for large-scale datasets and even under unstable time series by using a set of lagged observations of time series. However, as the workload scenario of ARIMA is limited to linear models, it is usually referenced as a baseline approach by many studies included in our literature review.

To further speed up the data training process, LSTM models are adopted in References [58, 67, 81] to predict the request arrival rates under large dynamic variations and avoid unnecessary resource provisioning operations. Unlike general feedforward neural networks, LSTM has a more complicated structure with feedback connections that can improve prediction accuracy. It produces prediction results by analyzing the whole data sequence and is more accurate at identifying new patterns. However, LSTM models only train the data in one direction. Bi-LSTM models can overcome this limitation by processing the data sequence from both forward and backward directions. Therefore, Bi-LSTM models are proposed in References [71, 78] to capture more key metrics and improve the prediction accuracy.

Another direction in ML-based workload characterization is resource demand modeling and prediction. Zhong et al. [8] leverage the K-means++ algorithm for task classification and identification based on the resource usage (e.g., CPU and memory) patterns of different workloads. This advantage of K-means based approach is scalable for multiple types of tasks; however, the accuracy is undermined under loads with high variance. Zhang et al. [18] introduce the **Time Series Nearest Neighbor Regression** (**TSNNR**) algorithm for prediction of future workload resource requirements by matching the recent time series data trend to similar historical data, which can improve the prediction accuracy compared with simple NN approach due to the consideration of time series. However, it is constrained to limited workload scenarios. To enhance the ARIMA model in the analysis of the nonlinear relationship and trend turning points within data sequences, Xie et al. [79] apply **ARIMA with triple exponential smoothing** (**ARIMA-TES**) in the prediction of container workload resource usage with multi-dimension, which can achieve high robustness and accuracy but bring higher time overhead compared with ARIMA.

Besides, A hybrid association learning architecture is designed in Reference [80] through combining the LSTM and Bi-LSTM models. A multi-layer structure is built to find the interdependencies and relationship between various resource metrics, which are generally classified into three distinct groups, including CPU, memory, and I/O. To further reduce the error rates of resource usage prediction and data training time, GRU-based models are utilized due to their high computational efficiency and prediction accuracy. Lu et al. [82] develop a hybrid prediction framework consisting of a GRU and a straggler detection model (IGRU-SD) to predict the periodical resource demand patterns of cloud workloads on a long-term basis. Likewise, Cheng et al. [72] propose a GRU-ES model where the exponential smoothing method is used to update the resource usage prediction results generated by GRU and reduce prediction errors. Generally, these hybrid solutions can improve the performance of a single technique; however, due to the complex architecture, it is more difficult to analyze the contributions of different variables.

In summary, the majority of the reviewed articles in workload characterization have focused on the analysis and prediction of request arrival rates and resource usage patterns through time series analysis or regression models. The researchers can benefit from the investigation in this section by selecting specific techniques according to their objectives and techniques' advantages. For example, given one research work aims at predicting resource demand with low error rates, it can utilize GRU-based approach in [58], and it can improve the existing work by using more comprehensive application models. If another work plans to forecast requests arrival rate with high accuracy, the Bi-LSTM approaches in [71] and [78] can be the candidates. However, the researchers need to overcome the inaccuracy in long-term prediction.

*4.2.2 Performance Analysis.* Performance analysis captures the key infrastructure and application-level metrics for evaluation of the overall system status or application performance. A summary of the ML-based performance analysis approaches is given in Table 3.

Table 3. Summary of Performance Analysis Approaches

| Ref. | Mechanism | Infrastructure | Application Architecture | Methods | Objectives | Advantages | Limitations |
|---|---|---|---|---|---|---|---|
| [59] | Time series analysis, regression | Single cloud | Serverless | LSTM, LR | Prediction of function invoking time | Online prediction of function chains | Additional resource consumption for model training |
| [83] | Regression | Hybrid cloud | Serverless | GBR | Prediction of costs and end-to-end latency | High prediction accuracy | High computational expenses |
| [84] | Regression | Hybrid cloud | Serverless | BO | Prediction of costs and execution time based on function configurations | High prediction accuracy | High computational complexity |
| [16] | Classification, regression | Hybrid cloud | Monolithic | K-means, PR | Classification of cloud service providers and container clusters | High prediction accuracy and effectiveness | Simplicity of application models |
| [28] | Time series analysis | Single cloud | Monolithic | ANN | Resource utilization and response time prediction | Incremental modeling | Long model training time |
| [63] | Time series analysis | Single cloud | Monolithic | ARIMA | GPU resource utilization prediction | Discovery of the peak of resource consumption | Implicit accuracy and time overhead evaluation |
| [44] | Regression | Single cloud | Monolithic | SVR, LR, and ANN | Prediction of application performance based on resource metrics | Low error rates | High computational expenses |
| [85] | Regression | Single cloud | Microservice | ANN | Performance modeling of microservice workflow systems | Sample complexity reduction | Randomness due to boundary effects |
| [86] | Regression | Multi-cloud | Microservice | SVR, ANN | Performance modeling of microservice response time | Time saving for route searching | Dependency on offline training of historical data |
| [87] | Regression | Hybrid cloud | Microservice | LR, PR, RF, and SVR | Prediction of offloading execution time | High prediction accuracy | High computational costs |
| [66] | Regression | Single cloud | Microservice | GP | Prediction of end-to-end latency | High prediction accuracy | High computational complexity |
| [74] | Regression | Single cloud | Microservice | LR, RF, and LASSO | Prediction of SLI | High prediction accuracy by removing anomalies | High computational complexity |
| [23] | Time series analysis, regression | Single cloud | Microservice | CNN, BT | End-to-end latency and QoS violations prediction | Online prediction with high resource efficiency | Overfitting and misprediction |
| [75] | Regression | Single cloud | Microservice | DT, MARS, boosting, and RF | Modeling and prediction of SSD I/O performance | Short data training time | Simplicity of application models |

Das et al. [83] design a performance modeler based on GBR for predicting the costs and end-to-end latency of input serverless functions based on their configurations under edge-cloud environments. Such metrics are the key factors in the estimation of the efficiency of function scheduling decisions. Similarly, Akhtar et al. [84] leverage the **Bayesian Optimization** (**BO**) function to achieve the same purpose. The prediction results will be used to estimate the optimal function configurations that meet the time constraints in function deployment with the lowest costs. Both the GBR and BO can provide high prediction accuracy, where the GBR is through the optimization of different loss functions and BO is via the direct search based on Bayes Theorem to find the best results of the objective function. However, both of these approaches suffer from high computational expenses.

To estimate the cold start latency in serverless computing platforms, Xu et al. [59] propose a two-phase approach. LSTM is used in the first phase to predict the invoking time of the first function in a function chain, while the rest of the functions is processed through LR. Although this two-phase approach consumes additional resources for model training, it achieves a significant reduction of prediction error rates and resource wasting in the container pool.

Venkateswaran and Sarkar [16] try to classify the cloud service providers and container cluster configurations under multi-cloud environments through a two-level approach. K-means is employed in the first level to precisely classify the comparable container cluster compositions by their performance data, while PR is applied in the second level for analyzing the relationship between container strength and container system performance. The disadvantage of this work is that the evaluated application model only contains a limited number of microservices.

Some research works have investigated the issue of infrastructure-level resource utilization prediction [28, 63, 75]. Zhang et al. [28] leverage the ANN model to predict the CPU utilization and request response time by modeling a series of metrics, including CPU and memory usage, response

Table 4. Summary of Anomaly Detection Approaches

| Ref. | Mechanism | Infrastructure | Application Architecture | Methods | Objectives | Advantages | Limitations |
|---|---|---|---|---|---|---|---|
| [41] | Classification | Hybrid cloud | Monolithic | K-means, ensemble, and hierarchical | Identification of overloaded or underloaded nodes | Short data training time | Limited workload scenarios, high space complexity |
| [88] | Classification | Single cloud | Monolithic | K-means, KNN, and self-organizing map | Container vulnerability detection | High detection accuracy | Insufficient anomaly cases |
| [42] | Classification | Single cloud | Microservice | KNN, SVM, NB, and RF | Anomaly detection of microservices according to real-time performance metrics | Various monitoring metrics | Insufficient evaluation of application-level anomalies |
| [69] | Time series analysis | Single cloud | Microservice | LSTM | Anomaly detection and prediction on application or infrastructure-level metrics | Improved prediction accuracy | Static models |
| [89] | Classification | Single cloud | Monolithic | Isolation forest | Identification of abnormal resource metrics | Improved detection accuracy | Unstable monitoring delays |

time, and the request arrival rates mentioned in Section 4.2.1. Besides, Dartois et al. [75] look into the research topic of SSD I/O performance modeling and interference prevention with a series of regression techniques, including boosting, RF, DT, and **Multivariate adaptive regression splines** (**MARS**). As SSDs are prevalently used by large-scale data center for data storage due to their high performance and energy efficiency, their internal mechanisms could directly impact application-level behaviors and cause potential SLO violations. Compared with the ANN approach, these regression-based approaches can obtain results within a shorter data training time, while the prediction accuracy is not as good as ANN.

On the other hand, some previous studies focus on behavior modeling of application-level metrics [23, 66, 74, 85]. RScale [66] is implemented as a robust scaling system with high computational complexity leveraging **Gaussian process** (**GP**) regression for investigation of the interconnections between end-to-end tail latency of microservice workloads and internal performance dependencies. Likewise, Sinan [23], as an ML-based and QoS-aware cluster manager for containerized microservices, combines the CNN and BT model for the prediction of end-to-end latency and QoS violations. Taking both the microservice inter-dependencies and the time series pattern of application-level metrics into account, Sinan evaluates the efficiency of short-term resource allocation decisions as well as long-term application QoS performance. In most cases, the approach functions well except overfitting and misprediction may happen occasionally.

Overall, most studies in this section are concerned with prediction of time constraints or resource usage patterns through regression or time series analysis techniques.

### 4.2.3 *Anomaly Detection.*

Anomaly detection is a critical mechanism for identifying abnormal behaviors in system states or application performance. Table 4 describes the reviewed approaches regarding anomaly detection.

Due to the application-centric and decentralized features of containers, they are more likely to experience a wide range of security threats, which may lead to potential propagation delays in container image dependency management or security attacks [90]. Tunde-Onadele et al. [88] classify 28 container vulnerability scenarios into six categories and develop a detection model, including both dynamic and static anomaly detection schemes with KNN, K-means, self-organization map algorithms, which could reach detection coverage up to 86%. High detection accuracy has been achieved by this approach; however, the investigated anomaly cases are not comprehensive.

To balance the energy efficiency and resource utilization of a containerized computing system under hybrid cloud environments, Chhikara et al. [41] employ K-means, hierarchical clustering algorithms, and ensemble learning for identification and classification of underloaded and overloaded hosts. Further container migration operations will be conducted between these two groups for load balancing and energy consumption reduction. The limitations are that this hybrid solution leads to a large solution space and only considers limited workload scenarios.

Table 5. Summary of Dependency Analysis Approaches

| Ref. | Mechanism | Infrastructure | Application Architecture | Methods | Objectives | Advantages | Limitations |
|---|---|---|---|---|---|---|---|
| [86] | Regression | Multi-cloud | Microservice | BO, GP | Discovery of optimal route of individual microservice | Load balance | Poor performance under highly dynamic environment |
| [24] | Classification | Single cloud | Microservice | SVM | Identification of potential heavy-loaded microservice units | High accuracy | Implicit explanation of the time overhead and computational costs |
| [23] | Classification | Single cloud | Microservice | CNN | Dependency analysis of microservices between pipelined-tiers | Navigation of system complexity | Overfitting and misprediction |
| [69] | Time series analysis | Single cloud | Microservice | LSTM | Dependency analysis among microservice units | Improved prediction accuracy | Ignorance of dependency updates |

Shah et al. [69] extend the LSTM model to analyze the long-term dependencies of real-time performance metrics among microservice units. Relying on the static models they constructed, they manage to identify critical performance indicators that support anomaly detection of various application and infrastructure-level metrics, including network throughput and CPU utilization.

*4.2.4 Dependency Analysis.* Table 5 summarizes the recent studies in service dependency analysis of containerized applications. As serverless function chains or workflows are usually predefined by users [91], current ML-based dependency analysis solutions only focus on decomposing the internal structures of microservice units and monitoring any dynamic structural updates.

SVM is utilized by Qiu et al. [24] to find the microservice units with higher risks causing SLO violations through analysis of the performance metrics related to the **critical path** (**CP**) of each individual microservice. CP is defined as the longest path between the client request and the underlying microservice in the execution history graph. Since CP can change dynamically at runtime in response to potential resource contention or performance interference, the SVM classifier is implemented with incremental learning for dependency analysis in a dynamic and consistent manner. However, the relationship between the time overhead and computational costs is not detailed discussed.

Load balancing between microservices under the multi-cloud environment could be rather complex, because of the unstable network latency, dynamic service configuration, and fluctuating application workloads. To address this challenge, Cui et al. [86] leverage the BO search algorithm with GP to produce the optimal load-balanced request chain for each microservice unit. Then all the individual request chains are consolidated into a tree structure dependency model that can be dynamically updated in case of potential environmental changes. This solution can achieve good load balancing effects, while performance can be degraded under a highly dynamic environment as the updates are quite time-consuming.

## 4.3 Resource Provisioning

In this section, we discuss various resource provisioning techniques, including scheduling, scaling, and migration.

*4.3.1 Scheduling.* As shown in Table 6, various algorithms have been proposed to solve the scheduling issue for improving system and application performance.

Most of the reviewed approaches follow a design pattern of combining ML-based Workload Modelers or Performance Analyzers with a heuristic scheduling Decision Maker, as discussed in Section 2.2.4. As the system complexity has been navigated by prediction models, bin packing and approximation algorithms such as best fit or least fit are commonly adopted to make scheduling decisions with improved resource utilization and energy efficiency [8, 16, 18, 58, 83, 84]. For example, Venkateswaran and Sarkar [16] manage to significantly reduce the complexity of selecting the best-fit container system and cluster compositions under multi-cloud environments, standing

Table 6. Summary of Scheduling Approaches

| Ref. | Infrastructure | Application Architecture | Methods | Task Structure | Objectives | Advantages | Limitations |
|---|---|---|---|---|---|---|---|
| [58] | Single cloud | Serverless | Heuristic | Multiple independent | Resource utilization improvement and energy saving | Reduction of cold start and response latency | Poor efficiency for tasks with long lifetimes |
| [83] | Hybrid cloud | Serverless | Heuristic | Single | Cost and latency minimization | Multi-objective task placement | Limited accuracy under high load variance |
| [84] | Hybrid cloud | Serverless | Heuristic | Graph-based | Cost minimization | SLA assurance | Simplicity of application workloads |
| [8] | Single cloud | Monolithic | Heuristic | Single | Resource utilization optimization | Load balance | High scheduling delays |
| [16] | Hybrid cloud | Monolithic | Heuristic | Single | Automated task deployment | Optimized container build time and provisioning time | High computational expenses and time overhead |
| [63] | Single cloud | Monolithic | Heuristic | Single | Resource utilization and energy efficiency optimization | QoS improvement | Insufficient analysis of computational costs and time complexity |
| [85] | Single cloud | Microservice | Actor-Critic | Graph-based | Cost saving | Training time reduction and accuracy improvement | Limitation caused by scarce data |
| [18] | Single cloud | Microservice | Heuristic | Graph-based | Resource utilization optimization | Cost saving | High scheduling delays |
| [25] | Single cloud | Microservice | Q-Learning, SARSA | Graph-based | Minimization of task execution time | SLA assurance | Limited scalability |
| [26] | Single cloud | Microservice | Actor-Critic, ANN | Single | Minimization of task completion time | Performance interference awareness | Implicit description of the space and time complexity |

on their prediction model described in Section 4.2.2. To mitigate the resource contention between co-located tasks deployed at the same host, Thinakaran et al. [63] implement a correlation-based scheduler for handling task co-location by measuring GPU consumption correlation metrics, especially consecutive peak resource demand patterns recognized by its ARIMA prediction model. Generally, although these heuristic-based approaches can achieve acceptable results within a short time, the results are not the optimal ones.

The rest of the studies choose RL models as the core component in their scheduling engine. For instance, Orhean et al. [25] choose two classic model-free RL models, namely, Q-Learning and SARSA, to schedule a group of tasks in a DAG structure. To reduce the overall DAG execution time, the internal tasks in a DAG categorized by their key features and priorities are scheduled under consideration of the dynamic cluster state and machine performance. To address the limitation of high sample complexity of model-free RL approaches, Zhang et al. [85] attempt to handle scientific workflows under microservice architectures through model-based RL. An ANN model is trained to emulate the system behavior by identification of key performance metrics collected from the microservice infrastructure, so that the synthetic interactions generated by ANN could directly be involved in the policy training process with Actor-Critical to generate scheduling decisions. In such a way, it simplifies the system model and avoids the time consuming and computationally expensive interactions within the real microservice environment. In RL-based scheduling algorithms, the states and actions require to be carefully designed, otherwise, the scalability can be significantly limited due to the large solution space.

In conclusion, heuristic or RL models are applied in most of the investigated works for decision making in task scheduling, to achieve resource utilization improvement and task completion time minimization.

*4.3.2 Scaling.* Scaling can dynamically adjust the system states in response to the changing workloads and cloud environments. Different from the scheduling introduced in Section 4.3.1, the target of scaling is for containerized applications rather than tasks. Currently, the dominant scaling mechanisms include horizontal scaling via increasing or decreasing instances of containers, vertical scaling via adding or removing hardware resources for a single container, and hybrid

Table 7. Summary of Scaling Approaches

| Ref. | Mechanism | Infrastructure | Application Architecture | Methods | Objectives | Advantages | Limitations |
|---|---|---|---|---|---|---|---|
| [59] | Horizontal | Single cloud | Serverless | Heuristic | Reduction of execution latency and resource consumption | Alleviation of cold starts | Instability under changing workloads |
| [60] | Horizontal | Single cloud | Serverless | Q-Learning | SLA assurance | Reduction of time overhead of cold starts | Simplicity of training model |
| [92] | Hybrid | Single cloud | Monolithic | Ensemble | Resource saving | High resource efficiency and reliability | Limited flexibility |
| [8] | Horizontal | Single cloud | Monolithic | Heuristic | Resource utilization and SLA assurance | High resource efficiency | Frequent cluster resizing |
| [28] | Horizontal | Single cloud | Monolithic | SARSA, Q-Learning | Resource utilization optimization | SLA violation reduction | Inaccuracy due to cold starts |
| [76] | Hybrid | Single cloud | Monolithic | Model-based RL | Minimization of application performance penalty, adaption costs, and resource usage | Improved training speed | Simplicity of application models |
| [77] | Hybrid | Multi-cloud | Monolithic | Model-based RL, heuristic | Optimal application performance and adaption time | Cost saving | Simplicity of application structures and QoS requirements |
| [30] | Horizontal | Hybrid Cloud | Monolithic | MDP, SARSA | Minimization of resource consumption and response time | High scalability | Limited dimensionality |
| [65] | Horizontal | Single cloud | Microservice | BO, GP | SLA assurance | High precision and short training time | Poor performance and sub-optimal decisions under workload spikes |
| [93] | Horizontal | Single cloud | Microservice | RF | SLA assurance | Container expansion time reduction | Limited QoS and workload scenarios |
| [29] | Vertical | Single cloud | Microservice | Q-Learning | Optimization of resource configurations and costs | SLA assurance | Limited workload scenarios |
| [70] | Horizontal | Single cloud | Microservice | Heuristic | Cost minimization | resource utilization improvement | High time complexity |
| [81] | Horizontal | Single cloud | Microservice | Heuristic | Resource efficiency | Avoidance of oscillations under unexpected workload spikes | Rigid scaling mechanisms |
| [66] | Horizontal | Single cloud | Microservice | Heuristic | SLA assurance | Resource utilization optimization | High time complexity |
| [67] | Horizontal | Single cloud | Microservice | Heuristic | SLA assurance | Lower request loss | High resource usage |
| [74] | Vertical | Single cloud | Microservice | Heuristic | SLO assurance | Improved resource utilization | Simplicity of application datasets |
| [78] | Horizontal | Hybrid cloud | Microservice | SARSA | Resource utilization optimization | SLA assurance | Capability limitation of edge devices |
| [23] | Horizontal | Single cloud | Microservice | Heuristic | QoS assurance | Resource utilization optimization | Implicit evaluation of computational costs and time complexity |
| [24] | Hybrid | Single cloud | Microservice | Actor-critic | Resource utilization optimization | SLO violation mitigation | Limited scalability and anomaly detection |

scaling by combining horizontal scaling and vertical scaling. The research works related to scaling are given in Table 7, which summarizes the adopted scaling mechanism, optimization objective, advantages, and limitations of investigated approaches. The researcher can select the appropriate approach based on their research goals.

To improve the decision quality and accuracy of RL-based autoscalers for monolithic applications with SLA assurance, Zhang et al. [28] build a horizontal scaling approach through the SARSA algorithm, based on analysis of the application workloads and CPU usage predicted by the ARIMA and ANN models. For addressing the same research questions under fog-cloud environments, Sami et al. [30] simulate the horizontal scaling of containers as an MDP model that considers both the changing workloads and free resource capacity in fogs. Then, SARSA is chosen for finding the optimal scaling strategy on top of the MDP model through online training at a small data scale. Further considering the possibility of hybrid scaling of monolithic applications, Rossi et al. [76] propose a model-based RL approach, targeting to find a combination of horizontal and vertical scaling decisions that meet the QoS requirements with the lowest adaption costs and resource wastage. Furthermore, the authors extend this model with a network-aware heuristic method for container placement in geographically distributed clouds [77]. Although the RL-based approaches mentioned above can improve resource utilization and QoS to a certain degree, their application workloads and QoS scenarios are too simple, without enough consideration of the diversity and complexity of cloud workloads.

Plenty of previous studies [66, 67, 70, 74, 81] have tried to leverage heuristic methods for microservice scaling, assisted by ML-based workload modeling and performance analysis. However, such approaches underestimate the inter-dependencies between microservices that are updated dynamically. On the other hand, model-based RL algorithms are usually unsuitable for microservice-based applications for the same reason. As microservice dependencies could potentially change at runtime, the simulation of state transitions could then be invalid.

Therefore, model-free RL algorithms are more common in the application scaling of microservices, as they do not rely on transition models. Qiu et al. [24] implement an SLO violation alleviation mechanism using Actor-Critic to scale the critical microservices detected by SVM as mentioned in Section 4.2.4. The Actor-Critic model produces a horizontal scaling decision to minimize the SLO violations through evaluating three crucial features, including SLO maintenance ratio, workload changes, and request composition. To speed up the model training process of RL methods, Yan et al. [78] design a multi-agent parallel training model based on SARSA for horizontal scaling of microservices under hybrid clouds. Assisted with the workload prediction results generated by Bi-LSTM , their elastic scaling approach could make a more accurate scaling decision of when, where, and how many microservice instances should be scaled up/down. In such a way, it achieves significant resource utilization improvement and cost reduction under SLA assurance. However, due to the high computation complexity, this approach is not suitable for edge devices given their limited capability.

Cold starts during the invocation of serverless functions are a serious performance bottleneck of serverless computing platforms. Cold starts are defined as the time overhead of environment setup and function initialization. Xu et al. [59] present a container pool scaling strategy, where function containers are pre-initialized by evaluating the first function invocation time (predicted by LSTM as discussed in Section 4.2.2) and the number of containers in each function category. Similarly, Agarwal et al. [60] introduce a Q-Learning agent to summarize the function invocation patterns and make the optimal scaling decisions of function containers in advance. A series of metrics, including the number of available function containers, per-container CPU utilization, and success/failure rates, are selected to represent the system states in the Q-Learning model. Due to the nature of model-free RL methods, prior information of the input function is not necessary. However, the performance is not guaranteed under changing workloads as only simple types of workloads are considered in these solutions.

In summary, there has been a large body of literature in the area of scaling using diverse solutions, covering all types of application architectures and cloud infrastructures. The majority of the reviewed works are related to autoscaling of microservice-based applications, with the model-free RL models as the latest resolution to process the dynamically changing workloads and microservice dependencies [24, 29, 78]. Their common targets mainly focus on SLA assurance and resource efficiency optimization.

*4.3.3 Migration.* As depicted in Table 8, Migration is a complementary mechanism for resource optimization across cloud environments.

Some reviewed approaches manage to bind ML-based behavior models with heuristic migration algorithms. Zhong et al. [8] develop a least-fit rescheduling algorithm to evict and relocate a set of lower-priority containers with the least QoS impact when unexpected resource contention occurs between co-located containers. The rescheduling algorithm readjusts the container configuration based on runtime performance metrics and selects the node with the most available resources evaluated by K-means for relocation. Besides, Chhikara et al. [41] introduce an energy-efficient

Table 8. Summary of Migration Approaches

| Ref. | Mechanism | Infrastructure | Application Architecture | Methods | Objectives | Advantages | Limitations |
|---|---|---|---|---|---|---|---|
| [8] | Rescheduling | Single cloud | Monolithic | Heuristic | SLA assurance | Resource contention alleviation | Long execution delays |
| [31] | Offloading | Hybrid cloud | Monolithic | MDP, DNN, and Q-Learning | Reduction of communication delays, power consumption | Low migration costs | High space complexity |
| [41] | Offloading | Hybrid cloud | Monolithic | Heuristic | Energy efficiency improvement | Load balance | High time and space complexity |
| [18] | Rescheduling | Single cloud | Microservice | Heuristic | Resource utilization optimization | Cost saving | SLA violations |
| [94] | Offloading | Hybrid cloud | Microservice | Q-Learning, MDP | Reduction of service delays and migration cost | Optimized performance | Lack of consideration on load balancing |

offloading model with a set of heuristic methods, including random placement, first-fit, best-fit, and correlation threshold-based placement algorithms. It is aimed at resource load balancing under hybrid cloud environments by migrating containers from overloaded nodes to underloaded nodes that are identified through classification as described in Section 4.2.3. However, since the performance metrics are obtained by third-party APIs, these approaches may come with high execution delays in large-scale computing systems.

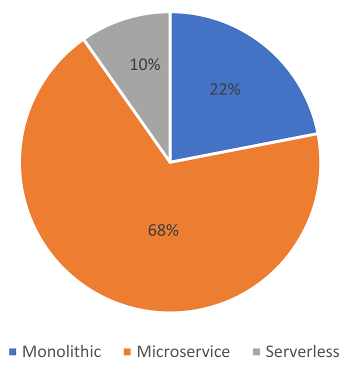
A fog-cloud container offloading prototype system is presented by Tang et al. [31]. The container offloading process is considered as a multi-dimensional MDP model. To reduce the network delays and computation costs under potentially unstable environments, the deep Q-Learning algorithm combines the **deep neural network** (**DNN**) and Q-Learning model to quickly produce an efficient offloading plan. Wang et al. [94] further extends the combination of MDP and Q-Learning in the context of microservice coordination under edge-cloud environments. The process of microservice coordination is assumed as a sequential decision scheme and formulated as an MDP model. On top of the MDP model, Q-Learning is used to find the optimal solution for service migration or offloading, in light of long-term performance metrics, including overall migration delays and costs. However, the large solution space of these approaches limits the scalability of a large-scale container-based cluster.

## 4.4 Observations

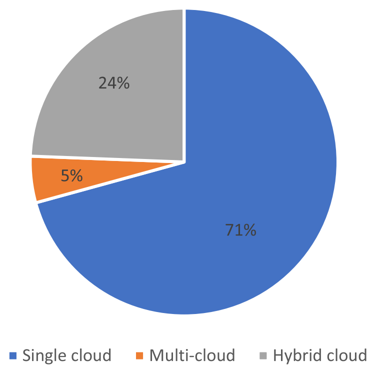
In this section, we summarize the observations of the discussed researches based on three important categories, namely, application architecture, infrastructure, and hybrid machine learning model.

*4.4.1 Application Architecture.* As described in Figure 7(a), most of the reviewed articles (68%) are concerned with modeling and management of microservice-based applications. In light of the dynamic and decentralized nature of microservices, diverse ML algorithms have been investigated for capturing the key characteristic of microservice workloads, performance, and inter-dependencies, such as BGR, LSTM, GRU, SVM, GP, ANN, CNN, and DT [23, 69, 70, 81, 82, 85, 86]. As the inter-dependencies of microservice units could dynamically update at runtime, the key concern of microservice resource provisioning is the chain reactions caused by microservice unit scaling operations. By combining ML-based behavior models with heuristic or model-free RL methods for scaling, the most recent studies manage to achieve high resource utilization optimization, cost reduction, and SLA assurance [24, 66, 67, 70, 74, 78, 81].

The research works related to the orchestration of single-component containerized applications hold a study distribution of 22%. As the workload scenarios of these applications are relatively simple, their core drive of workload/behavior modeling is to predict their resource demands under certain QoS requirements. Assisted with such prediction results, heuristic and RL methods (both model-free and model-based) are adopted for optimization of the resource allocation process regarding cost, energy, and resource efficiency [8, 16, 28, 44, 63, 79, 80].



(a) Application Architectures



(b) Serverless Architecture

Fig. 7. Study distribution of (a) application architectures and (b) cloud infrastructures.

Although the serverless architecture is currently having the lowest study distribution (10%), it is enjoying a growing popularity and becoming a prevalent application architecture in cloud computing. Most of the existing ML-based researches in this field are trying to alleviate the performance downgrade caused by function cold starts. LSTM and GBR models have been employed to estimate the function invocation time in serverless function chains, while the allocation of functions and scaling of function containers are mainly solved by heuristic methods and Q-Learning [58–60, 83].

*4.4.2 Infrastructure.* The majority of the included researches (71%) only consider application deployment under single cloud environments as demonstrated in Figure 7(b). Since single cloud environments are easier to manage, it is not necessary to raise the system complexity for most applications. Only 5% of studies have attempted behavior modeling and scaling under geographic-distributed multi-cloud environments [77, 86], because the interactions under such context are usually time-consuming and computation-intensive.

As traditional cloud computing commonly causes significant propagation delays, bandwidth and energy consumption by hosting all the applications and data in cloud servers, edge and fog computing are emerging as mainstream computing paradigms. Therefore, the latest studies have investigated the possibility of container orchestration under hybrid cloud environments, where the crucial research question is to decide where to host the containerized applications among cloud/edge devices and the cloud. The proposed solutions for scheduling and offloading of containerized applications under hybrid clouds, including MDP, RL, DRL, and heuristic methods [16, 30, 31, 41, 78, 83], aim to reduce end-to-end latency and optimize resource/energy efficiency.

*4.4.3 Hybrid Machine Learning Model.* Figure 8 shows the study distribution of ML models between 2016 and 2021, where we can observe a rising trend of hybrid ML models. A single ML model consisting of merely one ML algorithm is designed to solve a specific container orchestration problem, either data analysis or resource provisioning. By 2018, only single ML models had been adopted in this field. As the internal structures of containerized applications like microservices are becoming rather complex and dynamic with continuously growing demands of higher modeling/prediction accuracy and lower response time, this requires ML models to be more robust and efficient with even lower error rates, computation costs, and data training time. Therefore, most studies have been investigating hybrid ML models composed of a mixture of multiple ML algorithms to solve one or more orchestration problems from 2019 [31, 65, 72, 79, 80, 82, 94].

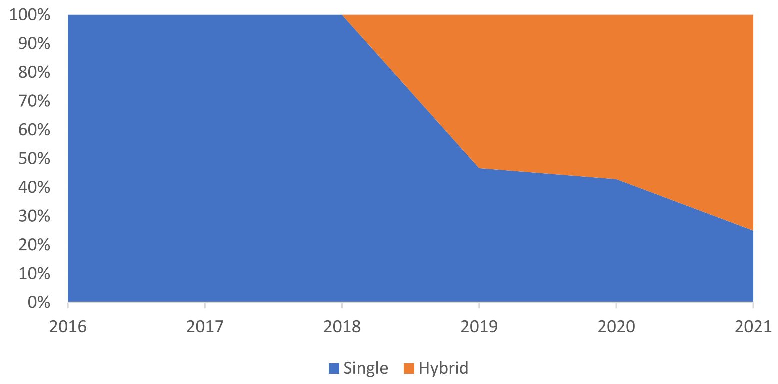


Fig. 8. Study distribution of machine learning models between 2016 and 2021.

Table 9. Problems in Containers Orchestration and Potential ML-based Solutions

| **Problems in container orchestration** | **Potential ML-based solutions** |
|---|---|
| How to characterize workloads by modeling and predicting requests behavior and resource usage pattern? | LSTM, K-means++, ARIMA, Bi-LSTM, GRU, TSNNR, and GBR |
| How to analyze inter-dependency between microservices? | BO, GP, CNN, and LSTM |
| How to detect anomalies in container orchestration systems? | K-means, KNN, SVM, NB, RF, LSTM, isolation forest, and LASSO |
| How to analyze dependency of tasks in containerized applications? | SVM |
| How to achieve energy efficient resource scheduling for containers? | MDP, Q-learning, and SARSA |
| How to balance the trade-offs between different metrics during container orchestration? | Actor-critic, ANN, and RF |
| How to reduce the communication delay when moving microservices to edge/fog devices? | DNN, Q-learning |
| How to alleviate function cold starts in serverless computing? | Q-learning, LSTM, and LR |
| How to make scaling/migration decisions for containers? | Model-based RL, and MDP |

## 5 DISCUSSIONS AND FUTURE DIRECTIONS

Tackling the problem of container orchestration based on ML in cloud computing requires addressing a set of sub-problems including workloads characterization, microservice inter-dependency analysis, container anomalies detection, and task dependency analysis. Various ML-based solutions have been proposed and applied to solve these problems. To support the readers to pick up the solutions to their target problems, Table 9 summarizes the techniques utilized in the surveyed research work. From the present survey, we can conclude that significant efforts have been made to address the workloads characterization problem and the dominant ML mechanisms, such as LSTM, Bi-LSTM, K-means, and ARIMA have been applied. However, for some other problems such as dependency analysis in tasks and communication delays in edge devices, they are worth considering the adoption of more ML solutions when realizing more efficient solutions to the problems.

Although the existing studies have covered diverse orchestration schemes, application architectures, and cloud infrastructures, some research gaps and challenges are only partially addressed. In this section, we discuss a series of open research opportunities and potential future directions:

(1) **Workload Distribution in Microservices.** The current workload characterization methods are mainly focusing on modeling and prediction of the request arrival rates and resource usage patterns. Very few works try to address the issue of workload distribution across microservice units. The changing workload distribution on individual microservices could potentially cause a chain reaction and further impact the overall application performance. Therefore, how to simulate and standardize workload distribution between microservices for load balance and performance optimization remains an unsolved research question. This

question can be addressed by using efficient and accurate workload distribution prediction methods based on ML, and the efficiency of the tasks scheduling discussed in Section 4.3.1 can be further improved.

(2) **Microservice Dependency Analysis.** The dynamic inter-dependencies between microservices is a crucial part of application complexity navigation. Though some works have attempted to predict application performance or identify critical microservice units through dependency analysis, there is no existing solution to explicitly address the relationship between the status of individual microservices and overall application performance metrics. Such analysis models are necessary for scheduling and scaling microservice-based applications, which can significantly improve performance optimization and SLA assurance. One promising approach is taking advantage of advanced ML-based approaches to establish comprehensive analysis models for microservices.

(3) **Systematic Anomaly Detection and Recovery.** The current ML-based anomaly detection methods for containerized applications are mostly based on resource/performance metrics or security threats. There is a need for a systematic approach for anomaly detection, root cause categorization, and recovery in a timely and accurate fashion, under different application architectures and cloud infrastructures. For example, RL-based approach can be applied for making decision on recovery.

(4) **Graph-based Task Scheduling.** Batch processing jobs consisting of a group of dependent tasks in DAG structures are common in containerized applications, but the literature related to the scheduling problem of such applications is very limited. Some previous studies manage to resolve this problem under a simplified condition of homogeneous job structures where each task in a DAG is configured similarly with the same execution time. This kind of assumption is rather unrealistic considering the complex and heterogeneous nature of DAG jobs. A sophisticated scheduling strategy should not only consider the overall DAG structure and different task configurations (e.g., resource demands, execution time, and replica sizes), but also the runtime performance metrics and fault tolerance. One promising way is to model these configurations as the states in the RL-based model and the tasks can be scheduled by the actions in the model to improve performance.

(5) **Management of Function Chains in Serverless Architectures.** The latest ML-based studies on serverless architectures are all related to the alleviation of function cold starts, with many other research directions left to be discovered. As functions are submitted without any prior application-level knowledge, it is tricky to implement a robust and efficient solution for workload classification, resource demand estimation, and autoscaling. How to optimize the invocation of function chains in an SLO and cost-aware manner is also a crucial research question, especially under hybrid cloud environments. As an initial exploration, we suggest using SVM or KNN-based approaches to conduct some analysis for the invocation of function chains.

(6) **Microservices in Hybrid Clouds.** Under the emerging trend of edge and fog computing, the most recent studies have made efforts to move microservices to edge/fog devices for communication delays reduction and cost saving. However, the extremely heterogeneous and geographically distributed computation resources within hybrid clouds significantly raise the complexity of application and resource management. A resource provisioning solution under such environments must consider the long-term impacts of each microservice placement or adjustment decision, regarding multi-dimensional optimization objectives, such as costs, energy, and end-to-end latency. Therefore, there is great potential for ML-based solutions to simulate the process of scheduling, scaling, and offloading for microservices under hybrid clouds.

(7) **Energy-aware Resource Management.** Through combing ML-based workload characterization and performance analysis models, the overall energy consumption of a system could be precisely predicted based on the resource utilization of each PM. Accordingly, these insights grant us more options for developing energy-aware orchestration frameworks. For instance, brownout technologies could be utilized to activate/deactivate optional microservices for energy efficiency optimization, according to the predicted trends in resource utilization and SLA violations [33], where the selection of deactivated microservices can be made by ML-based approaches.

(8) **Multi-dimensional Performance Optimization.** One of the essential optimization problems in container orchestration is to manage the tradeoff between different performance metrics, especially SLA violations and financial costs. Though some previous studies [58, 63, 83] address this issue to a certain degree by balancing several optimization objectives during resource provisioning, a standard performance analysis benchmark should be designed to accurately decompose the relation between a set of pre-defined key performance metrics. The relationship of various performance metrics can be analyzed via ML-based solutions, such as nearest neighbour or decision tree to represent co-relationship. Such knowledge could be further applied in resource provisioning to produce optimal orchestration decisions with multiple performance requirements taken into account.

(9) **Fully Integrated ML-based Optimization Engine.** Considering the long data training time and large data volumes requested by ML models, a partially integrated ML engine enjoys high popularity, where existing ML models are only utilized for behavior modeling to assist the original Container Orchestrator in online resource provisioning. Following the reference architecture of ML-based optimization engine in Section 2.2.4, a fully integrated ML engine should be capable of combining multiple ML models, such as integrating both offline training of behavior models and fast online decision making of resource provisioning.

(10) **Edge Intelligence.** Due to the extreme data scales and geographical distribution of edge devices, data transition to a centralized ML-based Optimization Engine across edge networks is potentially time-consuming and expensive. Hence, the next-generation edge intelligence agents should follow a decentralized architecture where each independent agent is deployed at a distributed location close to end users for data collection, analytical model construction, and maintenance of serverless or microservice-based applications under low propagation delays. However, the current researches are mainly conducted with small-scale experiments and the scalability problem with multiple agents is not well addressed. A promising solution is to apply the approximate solution based on RL to manage agents that can scale for a large-scale environment.

(11) **Costs Analysis of Applying ML.** Based on the current survey, ML solutions have demonstrated the powerful capacity to address some problems in container orchestration. However, in some cases, such as the online decision scenario or energy-efficient scenario, the computational costs of ML-based methods can undermine the algorithm performance or consume extra energy. Although some researches have aimed at reducing the costs of the online decision based on ML [24], the current research still lacks a comprehensive analysis of the costs incurred by ML-based solutions of container orchestration. One promising approach to address this problem is balancing the tradeoffs between offline and online training, for instance, in the RL-based approach, the more trained decisions based on historical data can be stored in the knowledge pool, and the online decisions can rely more on it rather than online training. In addition, the state space should be carefully designed.

## 6 CONCLUSION

In this work, we have conducted an extensive literature review on how container orchestration is managed using diverse machine learning-based approaches in the state-of-the-art studies, with emphasis on the specific application architectures and cloud infrastructures. The last few years have witnessed a continuously growing trend of the adoption of machine learning methods in containerized application management systems for behavior modeling, as well as resource provisioning of complicated decentralized applications. Compared with most traditional heuristic methods, ML-based models could produce more accurate orchestration decisions with shorter computation delays, under complex and dynamic cloud environments consisting of highly heterogeneous and geographically distributed computation resources.

We have observed machine learning-based methods applied under a wide range of scenarios, but there is no systematic approach to build a complete machine learning-based optimization framework in charge of the whole orchestration process. Under the emerging trend of moving microservice and serverless applications to hybrid clouds, such frameworks are necessary to handle the sustainably growing system complexity and balance multi-dimensional optimization objectives. This research work will help researchers find the important characteristics of ML-based orchestration approaches and will also help to select the most suitable techniques for efficient container orchestration with the specific requirement under different application architectures. For instance, researchers can utilize the GRU-based approach to conduct time series analysis of workloads, and exploit the RL-based approach to make resource provisioning decisions.

## ACKNOWLEDGMENTS

We thank Dr. Sukhpal Singh Gill and anonymous reviewers for their valuable suggestions on improving this paper.

## REFERENCES

[1] Qi Zhang, Lu Cheng, and Raouf Boutaba. 2010. Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications* 1, 1 (2010), 7–18.
[2] Saakshi Narula, Arushi Jain, and Prachi. 2015. Cloud computing security: Amazon web service. In *Proceedings of the 2015 International Conference on Advanced Computing Communication Technologies*. 501–505.
[3] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the 2015 European Conference on Computer Systems*. 1–18.
[4] Qixiao Liu and Zhibin Yu. 2018. The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from alibaba trace. In *Proceedings of the 2018 ACM Symposium on Cloud Computing*. 347–360.
[5] Emiliano Casalicchio and Stefano Iannucci. 2020. The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience* 32, 17 (2020), 1–21.
[6] Zhiheng Zhong and Rajkumar Buyya. 2020. A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources. *ACM Transactions on Internet Technology* 20, 2 (2020), 1–24.
[7] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.
[8] Zhiheng Zhong, Jiabo He, Maria A. Rodriguez, Sarah Erfani, Ramamohanarao Kotagiri, and Rajkumar Buyya. 2020. Heterogeneous task co-location in containerized cloud computing environments. In *Proceedings of the 2020 IEEE International Symposium on Real-Time Distributed Computing*. 79–88.
[9] Christina Terese Joseph and K. Chandrasekaran. 2019. Straddling the cStraddling the crevasse: A review of microservice software architecture foundations and recent advancementsrevasse: A review of microservice software architecture foundations and recent advancements. *Software: Practice and Experience* 49, 10 (2019), 1448–1484.
[10] Xiangping Bu, Jia Rao, and Cheng-Zhong Xu. 2009. A reinforcement learning approach to online web systems auto-configuration. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE, 2–11.

[11] Cheng-Zhong Xu, Jia Rao, and Xiangping Bu. 2012. URL: A unified reinforcement learning approach for autonomic cloud management. *Journal of Parallel and Distributed Computing* 72, 2 (2012), 95–105.

[12] Xiangping Bu, Jia Rao, and Cheng-Zhong Xu. 2012. Coordinated self-configuration of virtual machines and appliances using a model-free learning approach. *IEEE Transactions on Parallel and Distributed Systems* 24, 4 (2012), 681–690.

[13] Maria A. Rodriguez and Rajkumar Buyya. 2019. Container-based cluster orchestration systems: A taxonomy and future directions. *Software: Practice and Experience* 49, 5 (2019), 698–719.

[14] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the 2012 ACM Symposium on Cloud Computing*. 1–13.

[15] Andrew Chung, Jun Woo Park, and Gregory R. Ganger. 2018. Stratus: Cost-aware container scheduling in the public cloud. In *Proceedings of the 2018 ACM Symposium on Cloud Computing*. 121–134.

[16] Sreekrishnan Venkateswaran and Santonu Sarkar. 2019. Fitness-aware containerization service leveraging machine learning. *IEEE Transactions on Services Computing* 1, 1 (2019), 1–14.

[17] Tarek Menouer, Otman Manad, Christophe Cérin, and Patrice Darmon. 2019. Power efficiency containers scheduling approach based on machine learning technique for cloud computing environment. In *Proceedings of the 2019 International Symposium on Pervasive Systems, Algorithms and Networks*. 193–206.

[18] Haitao Zhang, Huadong Ma, Guangping Fu, Xianda Yang, Zhe Jiang, and Yangyang Gao. 2016. Container based video surveillance cloud service with fine-grained resource provisioning. In *Proceedings of the 2016 IEEE International Conference on Cloud Computing*. 758–765.

[19] Maryam Barshan, Hendrik Moens, Steven Latre, Bruno Volckaert, and Filip De Turck. 2017. Algorithms for network-aware application component placement for cloud resource allocation. *Journal of Communications and Networks* 19, 5 (2017), 493–508.

[20] Josep Ll. Berral, Ricard Gavalda, and Jordi Torres. 2011. Adaptive scheduling on power-aware managed data-centers using machine learning. In *Proceedings of the 2011 IEEE/ACM International Conference on Grid Computing*. 66–73.

[21] William Trneberg, Amardeep Mehta, Eddie Wadbro, Johan Tordsson, Johan Eker, Maria Kihl, and Erik Elmroth. 2017. Dynamic application placement in the Mobile cloud network. *Future Generation Computer Systems* 70, 1 (2017), 163–177.

[22] Shiqiang Wang, Murtaza Zafer, and Kin K. Leung. 2017. Online placement of multi-component applications in edge computing environments. *IEEE Access* 5, 1 (2017), 2514–2533.

[23] Yanqi Zhang, Weizhe Hua, Zhuangzhuang Zhou, G. Edward Suh, and Christina Delimitrou. 2021. Sinan: ML-based and QoS-aware resource management for cloud microservices. In *Proceedings of the 2021 ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 167–181.

[24] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2020. FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices. In *Proceedings of the 2020 USENIX Symposium on Operating Systems Design and Implementation*. 805–825.

[25] Alexandru Iulian Orhean, Florin Pop, and Ioan Raicu. 2018. New scheduling approach using reinforcement learning for heterogeneous distributed systems. *Journal of Parallel and Distributed Computing* 117, 1 (2018), 292–302.

[26] Yixin Bao, Yanghua Peng, and Chuan Wu. 2019. Deep learning-based job placement in distributed machine learning clusters. In *Proceedings of the 2019 IEEE INFOCOM - IEEE Conference on Computer Communications*. 505–513.

[27] Ricardo Bianchini, Marcus Fontoura, Eli Cortez, Anand Bonde, Alexandre Muzio, Ana-Maria Constantin, Thomas Moscibroda, Gabriel Magalhaes, Girish Bablani, and Mark Russinovich. 2020. Toward ML-centric cloud platforms. *Communications of the ACM* 63, 2 (2020), 50–59.

[28] Shubo Zhang, Tianyang Wu, Maolin Pan, Chaomeng Zhang, and Yang Yu. 2020. A-SARSA: A predictive container auto-scaling algorithm based on reinforcement learning. In *Proceedings of the 2020 IEEE International Conference on Web Services*. 489–497.

[29] Yu Xu, Jianguo Yao, Hans-Arno Jacobsen, and Haibing Guan. 2017. Cost-efficient negotiation over multiple resources with reinforcement learning. In *Proceedings of the 2017 IEEE/ACM International Symposium on Quality of Service*. 1–6.

[30] Hani Sami, Azzam Mourad, Hadi Otrok, and Jamal Bentahar. 2020. FScaler: Automatic resource scaling of containers in fog clusters using reinforcement learning. In *Proceedings of the 2020 International Wireless Communications and Mobile Computing*. 1824–1829.

[31] Zhiqing Tang, Xiaojie Zhou, Fuming Zhang, Weijia Jia, and Wei Zhao. 2019. Migration modeling and learning algorithms for containers in fog computing. *IEEE Transactions on Services Computing* 12, 5 (2019), 712–725.

[32] Denis Weerasiri, Moshe Chai Barukh, Boualem Benatallah, Quan Z. Sheng, and Rajiv Ranjan. 2017. A taxonomy and survey of cloud resource orchestration techniques. *Computing Surveys* 50, 2 (2017), 1–41.

[33] Minxian Xu and Rajkumar Buyya. 2019. Brownout approach for adaptive management of resources and applications in cloud computing systems: A taxonomy and future directions. *Computing Surveys* 52, 1 (2019), 1–27.

[34] Thang Le Duc, Rafael García Leiva, Paolo Casari, and Per-Olov Östberg. 2019. Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. *Computing Surveys* 52, 5 (2019), 1–39.

[35] Sukhpal Singh and Inderveer Chana. 2015. QoS-aware autonomic resource management in cloud computing: A systematic review. *Computing Surveys* 48, 3, Article 42 (2015), 46 pages.

[36] Claus Pahl, Antonio Brogi, Jacopo Soldani, and Pooyan Jamshidi. 2019. Cloud container technologies: A state-of-the-art review. *IEEE Transactions on Cloud Computing* 7, 3 (2019), 677–692.

[37] A. Brnabic and L. M. Hess. 2021. Systematic literature review of machine learning methods used in the analysis of real-world data for patient-provider decision making. *BMC Medical Informatics and Decision Making* 21, 1 (2021), 1–19.

[38] Djamel Djenouri, Roufaida Laidi, Youcef Djenouri, and Ilangko Balasingham. 2019. Machine learning for smart building applications: Review and taxonomy. *Computing Surveys* 52, 2 (2019), 1–36.

[39] Pedro Henriques Abreu, Miriam Seoane Santos, Miguel Henriques Abreu, Bruno Andrade, and Daniel Castro Silva. 2016. Predicting breast cancer recurrence using machine learning techniques: A systematic review. *Computing Surveys* 49, 3 (2016), 1–40.

[40] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. 2018. A survey on deep learning: Algorithms, techniques, and applications. *Computing Surveys* 51, 5 (2018), 1–36.

[41] Prateek Chhikara, Rajkumar Tekchandani, Neeraj Kumar, and Mohammad S. Obaidat. 2020. An efficient container management scheme for resource constrained intelligent IoT devices. *IEEE Internet of Things Journal* 1, 1 (2020), 1–13.

[42] Qingfeng Du, Tiandi Xie, and Yu He. 2018. Anomaly detection and diagnosis for container-based microservices with performance monitoring. In *Proceedings of the 2018 International Conference on Algorithms and Architectures for Parallel Processing*. 560–572.

[43] Kok-Lim Alvin Yau, Junaid Qadir, Hooi Ling Khoo, Mee Hong Ling, and Peter Komisarczuk. 2017. A survey on reinforcement learning models and algorithms for traffic signal control. *Computing Surveys* 50, 3 (2017), 1–38.

[44] Kejiang Ye, Yanmin Kou, Chengzhi Lu, Yang Wang, and Cheng-Zhong Xu. 2018. Modeling application performance in docker containers using machine learning techniques. In *Proceedings of the 2018 IEEE International Conference on Parallel and Distributed Systems*. 1–6.

[45] David Carrera, Malgorzata Steinder, Ian Whalley, Jordi Torres, and Eduard Ayguade. 2012. Autonomic placement of mixed batch and transactional workloads. *IEEE Transactions on Parallel and Distributed Systems* 23, 2 (2012), 219–231.

[46] Rasmus V. Rasmussen and Michael A. Trick. 2008. Round robin scheduling–a survey. *European Journal of Operational Research* 188, 3 (2008), 617–636.

[47] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. 2018. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems* 78, 1 (2018), 680–698.

[48] Mithun Mukherjee, Lei Shu, and Di Wang. 2018. Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Communications Surveys Tutorials* 20, 3 (2018), 1826–1857.

[49] Ricardo Bianchini, Marcus Fontoura, Eli Cortez, Anand Bonde, Alexandre Muzio, Ana-Maria Constantin, Thomas Moscibroda, Gabriel Magalhaes, Girish Bablani, and Mark Russinovich. 2020. Toward ML-Centric cloud platforms. *Communications of the ACM* 63, 2 (2020), 50–59.

[50] Mohsen Mosleh, Kia Dalili, and Babak Heydari. 2018. Distributed or monolithic? A computational architecture decision framework. *IEEE Systems Journal* 12, 1 (2018), 125–136.

[51] Zhongshan Ren, Wei Wang, Guoquan Wu, Chushu Gao, Wei Chen, Jun Wei, and Tao Huang. 2018. Migrating web applications from monolithic structure to microservices architecture. In *Proceedings of the 2018 Asia-Pacific Symposium on Internetware*. 1–10.

[52] Gabor Kecskemeti, Attila Csaba Marosi, and Attila Kertesz. 2016. The ENTICE approach to decompose monolithic services into microservices. In *Proceedings of the 2016 International Conference on High Performance Computing Simulation*. 591–596.

[53] Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, and Sartaj Singh Baveja. 2015. Lambda architecture for cost-effective batch and speed big data processing. In *Proceedings of the 2015 IEEE International Conference on Big Data*. 2785–2792.

[54] Theo Lynn, Pierangelo Rosati, Arnaud Lejeune, and Vincent Emeakaroha. 2017. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In *Proceedings of the 2017 IEEE International Conference on Cloud Computing Technology and Science*. 162–169.

[55] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. 2020. Xanadu: Mitigating cascading cold starts in serverless function chain deployments. In *Proceedings of the 2020 International Middleware Conference*. 356–370.

[56] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 2020. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. In *Proceedings of the 2020 International Conference on Architectural Support for Programming Languages and Operating Systems*. 467–481.

[57] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight virtualization for serverless applications. In *Proceedings of the 17th {usenix} Symposium on Networked Systems Design and Implementation ({nsdi} 20)*. 419–434.

[58] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Nachiappan C. Nachiappan, Mahmut Taylan Kandemir, and Chita R. Das. 2020. Fifer: Tackling resource underutilization in the serverless era. In *Proceedings of the 2020 International Middleware Conference*. 280–295.

[59] Zhengjun Xu, Haitao Zhang, Xin Geng, Qiong Wu, and Huadong Ma. 2019. Adaptive function launching acceleration in serverless computing platforms. In *Proceedings of the 2019 IEEE International Conference on Parallel and Distributed Systems*. 9–16.

[60] Siddharth Agarwal, Maria Alejandra Rodriguez, and Rajkumar Buyya. 2021. A reinforcement learning approach to reduce serverless function cold start frequency. In *Proceedings of the 2021 IEEE/ACM International Symposium on Cluster, Cloud, and Internet Computing*. 797–803.

[61] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2018. SOCK: Rapid task provisioning with serverless-optimized containers. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference* . USENIX Association, 57–69.

[62] Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, and Athanasios V. Vasilakos. 2014. Cloud computing: Survey on energy efficiency. *Computing Surveys* 47, 2 (2014), 1–36.

[63] Prashanth Thinakaran, Jashwant Raj Gunasekaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. 2019. Kube-Knots: Resource harvesting through dynamic container orchestration in GPU-based datacenters. In *Proceedings of the 2019 IEEE International Conference on Cluster Computing*. 1–13.

[64] Minxian Xu, Adel N. Toosi, and Rajkumar Buyya. 2020. A self-adaptive approach for managing applications and harnessing renewable energy for sustainable cloud computing. *IEEE Transactions on Sustainable Computing* 1, 1 (2020), 1–15.

[65] Guangba Yu, Pengfei Chen, and Zibin Zheng. 2019. Microscaler: Automatic scaling for microservices with an online learning approach. In *Proceedings of the 2019 IEEE International Conference on Web Services*. 68–75.

[66] Peng Kang and Palden Lama. 2020. Robust resource scaling of containerized microservices with probabilistic machine learning. In *Proceedings of the 2020 IEEE/ACM International Conference on Utility and Cloud Computing*. 122–131.

[67] Laszlo Toka, Gergely Dobreff, Balazs Fodor, and Balazs Sonkoly. 2020. Adaptive AI-based auto-scaling for Kubernetes. In *Proceedings of the 2020 IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*. 599–608.

[68] Yang Meng, Ruonan Rao, Xin Zhang, and Pei Hong. 2016. CRUPA: A container resource utilization prediction algorithm for auto-scaling based on time series analysis. In *Proceedings of the 2016 International Conference on Progress in Informatics and Computing*. 468–472.

[69] Syed Yousaf Shah, Zengwen Yuan, Songwu Lu, and Petros Zerfos. 2017. Dependency analysis of cloud applications for performance monitoring using recurrent neural networks. In *Proceedings of the 2017 IEEE International Conference on Big Data*. 1534–1543.

[70] Yi-Lin Cheng, Ching-Chi Lin, Pangfeng Liu, and Jan-Jan Wu. 2017. High resource utilization auto-scaling algorithms for heterogeneous container configurations. In *Proceedings of the 2017 IEEE International Conference on Parallel and Distributed Systems*. 143–150.

[71] Xuehai Tang, Qiuyang Liu, Yangchen Dong, Jizhong Han, and Zhiyuan Zhang. 2018. Fisher: An efficient container load prediction model with deep neural network in clouds. In *Proceedings of the 2018 IEEE International Conference on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications*. 199–206.

[72] Yuming Cheng, Chao Wang, Huihuang Yu, Yahui Hu, and Xuehai Zhou. 2019. GRU-ES: Resource usage prediction of cloud workloads using a novel hybrid method. In *Proceedings of the 2019 IEEE International Conference on High Performance Computing and Communications; the 2019 IEEE International Conference on Smart City; the 2019 IEEE International Conference on Data Science and Systems*. 1249–1256.

[73] Rui Fu, Zuo Zhang, and Li Li. 2016. Using LSTM and GRU neural network methods for traffic flow prediction. In *Proceedings of the 2016 Youth Academic Annual Conference of Chinese Association of Automation*. 324–328.

[74] Vladimir Podolskiy, Michael Mayo, Abigail Koay, Michael Gerndt, and Panos Patros. 2019. Maintaining SLOs of cloud-native applications via self-adaptive resource sharing. In *Proceedings of the 2019 IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. 72–81.

[75] Jean-Emile Dartois, Jalil Boukhobza, Anas Knefati, and Olivier Barais. 2019. Investigating machine learning algorithms for modeling SSD I/O performance for container-based virtualization. *IEEE Transactions on Cloud Computing* 1, 1 (2019), 1–14.

[76] Fabiana Rossi, Matteo Nardelli, and Valeria Cardellini. 2019. Horizontal and vertical scaling of container-based applications using reinforcement learning. In *Proceedings of the 2019 IEEE International Conference on Cloud Computing*. 329–338.

[77] Fabiana Rossi, Valeria Cardellini, and Francesco Lo Presti. 2019. Elastic deployment of software containers in geo-distributed computing environments. In *Proceedings of the 2019 IEEE Symposium on Computers and Communications*. 1–7.

[78] Ming Yan, XiaoMeng Liang, ZhiHui Lu, Jie Wu, and Wei Zhang. 2021. HANSEL: Adaptive horizontal scaling of microservices using Bi-LSTM. *Applied Soft Computing* 105, 1 (2021), 107216–107230.

[79] Yulai Xie, Minpeng Jin, Zhuping Zou, Gongming Xu, Dan Feng, Wenmao Liu, and Darrell Long. 2020. Real-time prediction of docker container resource load based on a hybrid model of ARIMA and triple exponential smoothing. *IEEE Transactions on Cloud Computing* 1, 1 (2020), 1–17.

[80] Siddhant Kumar, Neha Muthiyan, Shaifu Gupta, A. D. Dileep, and Aditya Nigam. 2018. Association learning based hybrid model for cloud workload prediction. In *Proceedings of the 2018 International Joint Conference on Neural Networks*. 1–8.

[81] Mahmoud Imdoukh, Imtiaz Ahmad, and Mohammad Gh Alfailakawi. 2019. Machine learning-based auto-scaling for containerized applications. *Neural Computing and Applications* 1, 1 (2019), 1–16.

[82] Yao Lu, Lu Liu, John Panneerselvam, Bo Yuan, Jiayan Gu, and Nick Antonopoulos. 2020. A GRU-based prediction framework for intelligent resource management at cloud data centres in the age of 5G. *IEEE Transactions on Cognitive Communications and Networking* 6, 2 (2020), 486–498.

[83] Anirban Das, Shigeru Imai, Stacy Patterson, and Mike P Wittie. 2020. Performance optimization for edge-cloud serverless platforms via dynamic task placement. In *Proceedings of the 2021 IEEE/ACM International Symposium on Cluster, Cloud, and Internet Computing*. 41–50.

[84] Nabeel Akhtar, Ali Raza, Vatche Ishakian, and Ibrahim Matta. 2020. COSE: Configuring serverless functions using statistical learning. In *Proceedings of the IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 129–138.

[85] Zhe Yang, Phuong Nguyen, Haiming Jin, and Klara Nahrstedt. 2019. MIRAS: Model-based reinforcement learning for microservice resource allocation over scientific workflows. In *Proceedings of the 2019 IEEE International Conference on Distributed Computing Systems*. 122–132.

[86] Jieqi Cui, Pengfei Chen, and Guangba Yu. 2020. A learning-based dynamic load balancing approach for microservice systems in multi-cloud environment. In *Proceedings of the 2020 IEEE International Conference on Parallel and Distributed Systems*. 334–341.

[87] Ayesha Abdul Majeed, Peter Kilpatrick, Ivor Spence, and Blesson Varghese. 2019. Performance estimation of container-based cloud-to-fog offloading. In *Proceedings of the 2019 IEEE/ACM International Conference on Utility and Cloud Computing Companion*. 151–156.

[88] Olufogorehan Tunde-Onadele, Jingzhu He, Ting Dai, and Xiaohui Gu. 2019. A study on container vulnerability exploit detection. In *Proceedings of the 2019 IEEE International Conference on Cloud Engineering*. 121–127.

[89] Zhuping Zou, Yulai Xie, Kai Huang, Gongming Xu, Dan Feng, and Darrell Long. 2019. A docker container anomaly monitoring system based on optimized isolation forest. *IEEE Transactions on Cloud Computing* 1, 1 (2019), 1534–1543.

[90] Rui Shu, Xiaohui Gu, and William Enck. 2017. A study of security vulnerabilities on docker hub. In *Proceedings of the 2017 ACM on Conference on Data and Application Security and Privacy*. 269–280.

[91] Maciej Malawski, Adam Gajek, Adam Zima, Bartosz Balis, and Kamil Figiela. 2020. Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda, and Google cloud functions. *Future Generation Computer Systems* 110, 1 (2020), 502–514.

[92] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. 2020. Autopilot: Workload autoscaling at Google. In *Proceedings of the 2020 European Conference on Computer Systems*. 1–16.

[93] Jingze Lv, Mingchang Wei, and Yang Yu. 2019. A container scheduling strategy based on machine learning in microservice architecture. In *Proceedings of the 2019 IEEE International Conference on Services Computing*. 65–71.

[94] Shangguang Wang, Yan Guo, Ning Zhang, Peng Yang, Ao Zhou, and Xuemin Shen. 2021. Delay-aware microservice coordination in Mobile edge computing: A reinforcement learning approach. *IEEE Transactions on Mobile Computing* 20, 3 (2021), 939–951.

Received June 2021; revised November 2021; accepted December 2021