CONTRIBUTED PAPER

Cloud-Native Computing: A Survey From the Perspective of Services

This article surveys the past and present of cloud-native applications with respect to the key problems during their lifecycle from a research perspective.

By Shuiguang Deng, *Senior Member IEEE*, Hailiang Zhao, Binbin Huang, Cheng Zhang, Feiyi Chen, Yinuo Deng, Jianwei Yin, Schahram Dustdar, *Fellow IEEE*, and Albert Y. Zomaya, *Fellow IEEE*

ABSTRACT | The development of cloud computing delivery models inspires the emergence of cloud-native computing. Cloud-native computing, as the most influential development principle for web applications, has already attracted increasingly more attention in both industry and academia. Despite the momentum in the cloud-native industrial community, a clear research roadmap on this topic is still missing. As a contribution to this knowledge, this article surveys key issues during the life cycle of cloud-native applications, from the perspective of services. Specifically, we elaborate on the research domains by decoupling the life cycle of cloud-native applications into four states: building, orchestration, operation, and maintenance. We also discuss the fundamental necessities and summarize the key performance metrics that play critical roles during the development and management of cloud-native applications. We highlight the key implications and limitations of existing works in each state. The challenges, future directions, and research opportunities are also discussed.

KEYWORDS | Cloud-native applications; microservice; research roadmap; service life-cycle management; survey.

NOMENCLATURE

SOC	Service-oriented computing.
DevOps	Combination of development and operations.
CNCF	Cloud Native Computing Foundation.
VM	Virtual machine.
KVM	Kernel-based virtual machine.
SDN	Software-defined network.
CNF	Cloud-native network function.
GRE	Generic routing encapsulation.
LVM	Logical volume manager.
CNI	Container network interface.
RBAC	Role-based access control.
SDK	Software development kit.
NAT	Network address translation.
PaaS	Platform as a Service.
JCT	Job completion time.
REST	Representational state transfer.
GA	Genetic algorithm.
QoS	Quality of Service.
FaaS	Function as a Service.
VPA	Vertical pod autoscaler.

Manuscript received 9 June 2023; revised 28 November 2023; accepted 3 January 2024. Date of publication 12 February 2024; date of current version 4 March 2024. This work was supported in part by the National Science Foundation of China under Grant 62125206, Grant U20A20173, and Grant 62202133; in part by the National Key Research and Development Program of China under Grant 2022YFB4500100; in part by the Key Research Project of Zhejiang Province under Grant 2022C01145; and in part by the Zhejiang Provincial National Science Foundation of China under Grant LY23F020015. *(Corresponding authors: Shuiguang Deng; Hailiang Zhao.)*

Shuiguang Deng and **Hailiang Zhao** are with the First Affiliated Hospital, Zhejiang University School of Medicine, Hangzhou 310003, China, and also with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: dengsg@zju.edu.cn; hliangzhao@zju.edu.cn).

Binbin Huang is with the College of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310012, China (e-mail: huangbinbin@hdu.edu.cn).

Cheng Zhang, **Feiyi Chen**, **Yinuo Deng**, and **Jianwei Yin** are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: coolzc@zju.edu.cn; chenfeiyi@zju.edu.cn; yinuo@zju.edu.cn; zjuyjw@zju.edu.cn).

Schahram Dustdar is with the Distributed Systems Group, Technische Universität Wien, 1040 Vienna, Austria (e-mail: dustdar@dsg.tuwien.ac.at).

Albert Y. Zomaya is with the School of Computer Science, The University of Sydney, Sydney, NSW 2006, Australia (e-mail: albert.zomaya@sydney.edu.au).

Digital Object Identifier 10.1109/JPROC.2024.3353855

ANN	Artificial neural network.
GNN	Graph neural network.
LSTM	Long short-term memory.
SOA	Service-oriented architecture.
CI/CD	Continuous integration and continuous delivery.
API	Application programming interfere.
ISP	Internet service provider.
K8s	Kubernetes.
NFV	Network function virtualization.
VXLAN	Virtual eXtensible Local Area Network.
MPLS	Multiprotocol label switching.
RAID	Redundant array of independent disks.
CSI	Container storage interface.
DNS	Domain name system.
TLS	Transport layer security.
SaaS	Software as a Service.
IaaS	Infrastructure as a Service.
CRUD	Create, read, update, and deletion.
HPC	High-performance computing.
QoE	Quality of Experience.
SLO	Service-level objective.
BaaS	Backend as a Service.
HPA	Horizontal pod autoscaler.
MARL	Multiagent reinforcement learning.
BGP	Border gateway protocol.
RNN	Recurrent neural network.

I. INTRODUCTION

Services are self-describing and technology-neutral computation entities that support rapid and low-cost composition of web applications in distributed network systems [1]. SOA is the principle of designing software systems by provisioning independent, reusable, and automated functions as reusable services and providing a robust and secure foundation for leveraging these services [2]. In recent years, the most influential variant of SOA is *the microservice architecture*, which decouples a monolithic application into a collection of loosely coupled, fine-grained microservices, communicating through lightweight protocols [3]. Over the last decade, the microservice architecture has become more and more appealing, as it allows software organizations to be more productive in building systems with the support of DevOps [4] through CI/CD pipelines [5].

Accompanied by the development of microservices, a new terminology, *cloud-native*, or *cloud-native computing*, is attracting increasing attention in academia. In accordance with CNCF, the open-source, vendor-neutral hub of cloud-native computing,[1] cloud-native is the collection of technologies that *break down applications into microservices and package them in lightweight containers to be deployed and orchestrated across a variety of servers.*[2] In addition to the microservice architecture, cloud-native is also characterized by the following terminologies.

1) *Containerization:* It is a function isolation mechanism that leverages the Linux kernel to isolate resources, creating containers as different processes in Host OS [6], [7]. Docker, with a ten-year development, is the most popular implementation of the containerization techniques [8]. Combining containerization with the microservice architecture, each part of an application, including processes, libraries, and so on, is packaged into its own container. This facilitates reproducibility, transparency, and resource isolation.
2) *Orchestration:* It is the automated configuration, management, and coordination of the interrelated microservices to build elastic and scalable functionalities. Since microservices are deployed in the way of containers, orchestration reduces the automation of the operational effort to manage the containers' life cycle, including resource provisioning, deployment, scheduling, scaling (up and down), networking, load balancing, and so on, in order to execute the applications' workflows or processes. K8s [9], originated from Google's Borg cluster manager [10], is the most popular open-source container orchestration software.

In conclusion, a cloud-native application can be viewed as a distributed, elastic, and horizontal scalable system composed of interrelated microservices, which isolates the state in a minimum of stateful components [11]. Cloud-native can be regarded as cloud computing version 2.0. Specifically, cloud computing provides the infrastructure and backend services over the Internet [12] while being cloud-native involves an application architecture and development approach that maximizes the benefits of cloud computing. Being cloud-native adopts practices including microservices, containerization, and orchestration to enable agility, scalability, and rapid development, and deployment of applications. Applications that are built with cloud-native technologies generally follow these steps: 1) separating the monolith into self-deployed, function-explicit microservices and letting them communicate with each other through REST APIs (for synchronous communication) and lightweight messaging protocols (for asynchronous communication); 2) using lightweight operating system virtualization technology, i.e., containerization, to pack each microservice into a container; 3) orchestrating these containers into an organic whole for functionalities with automatic configuration and management throughout their life cycle; and 4) using DevOps and CI/CDs to deliver the cloud-native applications with reliability and scalability.

Considering that cloud-native is better known in the industry, in this article, we try to survey the past and present of cloud-native applications with respect to the key problems during their life cycle from a research perspective. We attempt to merge the industrial popularity, including the widely used open-source software and platforms, with the trending research, either theoretical or systematic,

[1] https://www.cncf.io/about/who-we-are/
[2] https://github.com/cncf/toc/blob/main/DEFINITION.md

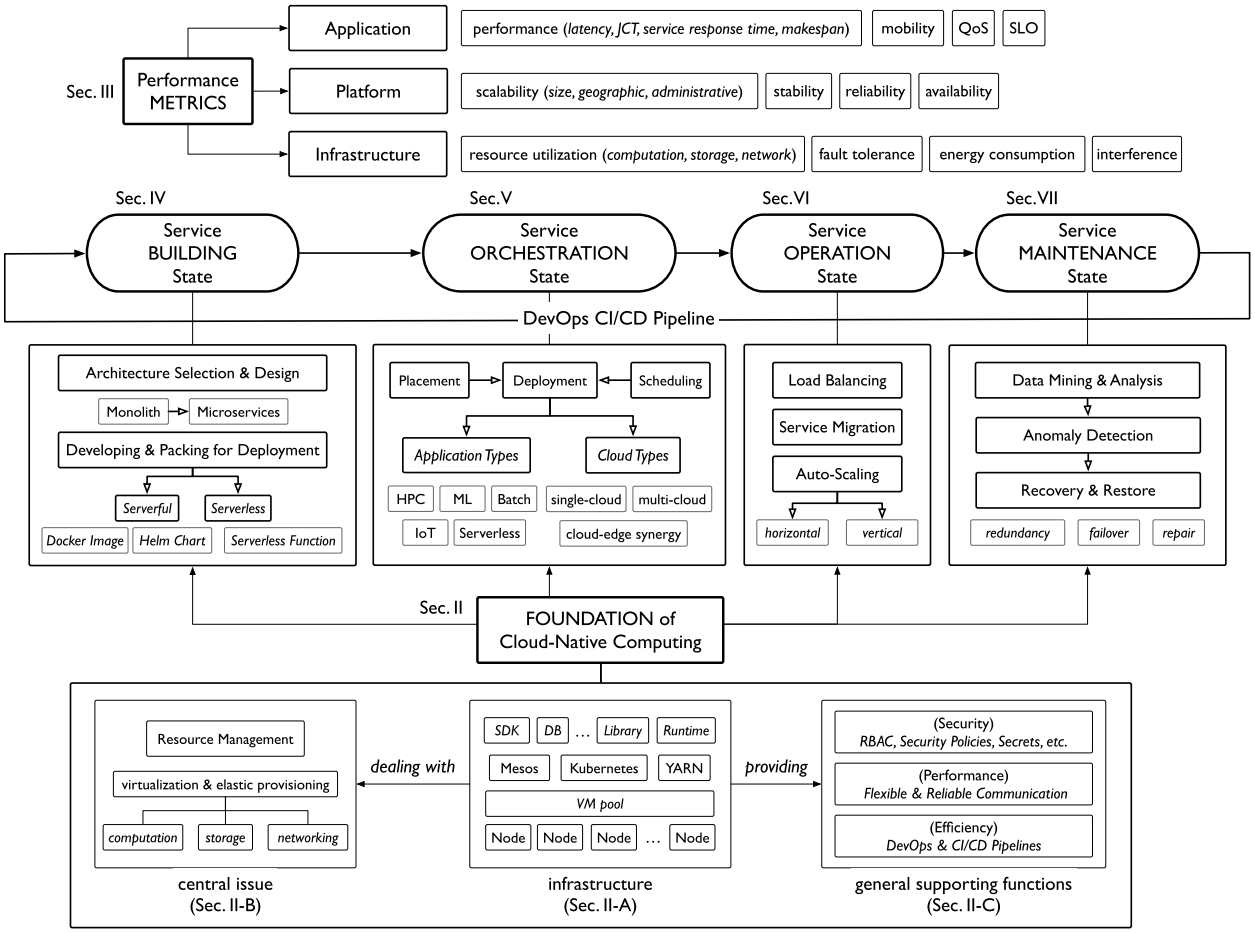


Fig. 1. ***Research roadmap for cloud-native computing, designed from the life cycle of cloud-native applications.***

from the perspective of services computing. We divide the life cycle of a cloud-native application, which is viewed as a *service* in the field of SOC, into four states: *building*, *orchestration*, *operation*, and *maintenance*. As shown in Fig. 1, different key problems are emphasized in different states. In addition, we also collect the performance metrics that are frequently mentioned when building cloud-native applications and analyze them from three levels: *infrastructure*, *platform*, and *software*. Apart from the service life cycle and performance metrics, Fig. 1 also demonstrates the foundation of cloud-native computing. In our opinion, the fundamental issue to be dealt with for building cloud-native applications is adaptive resource management, i.e., elastic resource virtualization and provisioning, such that business agility can come true. Besides, as the foundation, it should provide general function modules for the building and running of cloud-native applications. We divide the functions into three cases: security, performance, and efficiency, and demonstrate the corresponding popular platforms and tools. Sections II–VII will be presented surrounding the roadmap.

To the best of our knowledge, this is the first survey focusing on key issues during the life cycle of cloud-native applications from the perspective of services. There are some studies on the origin, status quo, challenges, and opportunities of cloud-native applications [11], [13], [14], [15], [16]. However, they mainly provide high-level opinions and ignore the comprehensive review of the state-of-the-art research works. As a result, researchers may find it struggling to grasp and comprehend each issue in the development and management of cloud-native applications. It is worth mentioning that there are surveys focusing on specific topics in cloud-native computing. For example, Duan [17] discusses the recent developments of architectural frameworks for intelligent and autonomous management for cloud-native networks. This article comprehensively reviewed the technical trend toward cloud-native network design and network-cloud/edge convergence. Moreover, the scheduling of microservices and containers, which has a strong connection with the orchestration platform K8s, is reviewed in [18] and [19]. However, due to the lack of systematic knowledge, challenges and proposed solutions will lack high portability and compatibility for various cloud-native applications. To this end, this survey is inspired to propose a *pipelined* design and summarize the research domains

Table 1 Typical Open-Source Projects in Cloud-Native Computing

NAME	OPEN-SOURCE URL	FUNCTIONS
Kubernetes	https://github.com/kubernetes	Manage containerized applications across multiple hosts
KubeEdge	https://github.com/kubeedge/kubeedge	Extend native containerized application orchestration and device management of Kubernetes to hosts at the edge
Helm	https://github.com/helm/helm	Manage packages of pre-configured Kubernetes resources
Istio	https://github.com/istio/istio	An implementation of service mesh that layers transparently onto existing distributed applications
Envoy	https://github.com/envoyproxy/envoy	A service mesh proxy that provides features such asload balancing, routing, health checking, and observability for microservices-based architectures
Prometheus	https://github.com/prometheus/prometheus	Provide monitoring and alerting functionality for cloud-native environments, including Kubernetes
Grafana	https://github.com/grafana/grafana	A composable observability and data visualization platform for cloud-native
Flannel	https://github.com/flannel-io/flannel	A tool to configure a layer 3 network fabric designed for Kubernetes
Calico	https://github.com/projectcalico/calico	Provide network connectivity and security policy enforcement between Kubernetes workloads
KubeSphere	https://github.com/kubesphere	A distributed operating system for cloud-native application management, using Kubernetes as its kernel

from different views. It can help researchers and practitioners to further understand the nature of cloud-native applications. As shown in Fig. 1, we analyze the key problems across the whole life cycle of cloud-native applications, from the building state to the maintenance state. We also discuss the performance metrics and the fundamental necessities when developing cloud-native applications.

The rest of the survey is organized as follows. Section II–VII introduces the foundation, performance metrics, and the four states demonstrated in this roadmap. Section VIII discusses the issues, challenges, limitations, and opportunities of cloud-native. We conclude this article in Section IX. We summarize the definitions of the acronyms that will be frequently used in this article in Nomenclature for ease of reference.

II. FOUNDATION OF CLOUD-NATIVE COMPUTING

In this section, we demonstrate the fundamental necessities of cloud-native. From the perspective of clusters, we demonstrate the central issue of the foundation and general function modules that play critical roles when building cloud-native applications. Widely used tools and open-source projects in cloud-native are listed in Table 1 for ease of reference.

A. Hierarchical Structure of Infrastructure

A cluster is a set of computing nodes that are connected to each other through fast local area networks. On top of the physical nodes, OS-level virtualization techniques are utilized to create VMs such that operating costs and downtime can be minimized. By maintaining a pool of VMs, fast provisioning of resources can be realized with cluster management software. Cluster management is always tightly coupled with resource management and task scheduling. K8s, as we have mentioned before, is widely adopted across industries and has become a *de facto* standard. K8s has the ability to manage nodes (including both physical servers and VMs) with the module *Node Controller*,[3] which is responsible for node registration, keeping the nodes up-to-date, and monitoring their health. In addition to K8s, Mesos [20], Docker Swarm [21], and Hadoop YARN [22] are also widely used cluster managers.

1) *K8s:* It is the most influential open-source platform in the cloud-native ecology since its release in 2014. K8s originates in Google's Borg [10], which has been used for managing containerized workloads in Google's inner clusters for more than a decade. K8s is a distributed software deployed across nodes, whose target is the automation of the deployment, management, and scaling of containerized applications by efficiently managing heterogeneous resources.[4] With the center being K8s, an open-source ecosystem gradually forms to improve and facilitate the management of cloud-native applications. In the ever-growing K8s ecosystem, publishing, installing, removing, and upgrading cloud-native applications can be managed with Helm.[5] The traffic between internal microservices within an application can be managed with Istio,[6] which is the most influential implementation of Service Mesh [23]. Istio extends K8s to establish a programmable, application-aware network by taking Envoy[7] as the proxy. The monitoring of nodes

[3]https://kubernetes.io/docs/concepts/architecture/nodes/
[4]The architecture of K8s can be found in the official document: https://kubernetes.io/docs/concepts/overview/components/
[5]https://helm.sh/
[6]http://istio.io/
[7]https://www.envoyproxy.io/

and applications can be realized with Prometheus[8] and Grafana,[9] while logging can be managed with Kibana.[10]

2) *Mesos:* It is implemented in a two-level architecture. The first level is a *master resource manager* that controls which resources each *framework scheduler* owns. In the second level, each *framework scheduler*, such as Hadoop, MPI, and Mesos Marathon, is responsible for scheduling specific tasks at the application level [20].
3) *YARN:* It manages Hadoop clusters. It consists of clients, containers, resource managers, node managers, and application masters. Here, the container is defined as a collection of physical resources, such as RAM, CPU cores, and disk on a single node. It is a node manager that takes care of each node and manages applications and workflows on that particular node. When a job is submitted by a client, the corresponding application master negotiates resources with the resource manager and requests a container from the node manager [22].
4) *Docker swarm:* This is the native inbuilt orchestration tool for Docker, which is called "swarm mode." Docker Swarm is composed of Docker nodes, Docker services, and Docker tasks. There are two kinds of Docker nodes, manager and worker, which are similar to the master/worker in K8s. The manager, as the name suggests, is responsible for maintaining the cluster status, scheduling the services, and serving swarm mode HTTP API endpoints. By contrast, the workers are nothing but the instances of the Docker engine for running Docker containers [24].

The following contents are mainly focusing on K8s since, in cloud-native, many rigor progresses are achieved based on K8s. Apart from the above cluster managers, it is worth pointing out that there are orchestration frameworks that extend the native capabilities of K8s to the network edge. KubeEdge[11] is a representative one. Edge computing has a three-level hierarchy: *cloud-edge-device* [25]. To take full advantage of this hierarchy, KueEdge divides its components into two parts: CloudCore and EdgeCore. CloudCore handles the communication between it and the API server of a K8s cluster and communicates with the edge nodes. EdgeCore is responsible for communicating with CloudCore and manages the containers and services that are deployed on the edge devices.[12] In a recent project, KubeEdge is reported to stably support 100 000 concurrent edge nodes and manage more than one million pods[13] [26].

[8]https://prometheus.io
[9]https://grafana.com
[10]https://www.elastic.co/kibana/
[11]https://kubeedge.io/en/
[12]The architecture of KubeEdge can be found in the official document: https://kubeedge.io/en/docs/kubeedge/
[13]In K8s, a pod is the smallest deployable unit that represents a single instance of a running process. A pod encapsulates one or more containers, storage resources, and unique network IP, as well as options that govern how the container(s) should run.

On top of cluster managers, development tools, including SDKs and middleware, are developed as the building blocks for cloud-native applications. The hierarchical structure of hardware, OS-level software, cluster managers, and middleware constructs the fundamental necessities for building cloud-native applications.

B. Resource Provisioning and Management

Virtualization refers to a collection of techniques for building and managing virtual resources on top of actual hardware, with key benefits including high redundancy, unified interfaces for users, and highly efficient resource utilization. It is the infrastructural foundation of today's cloud-native orchestration at all scales. When provisioning resources at a large scale, virtualization manages all low-level and possibly heterogeneous resources such that better global efficiency can be reached in comparison to the traditional way of letting service users decide on their own since service users often do not have access to the global resource utilization information. Typical computation virtualization technologies are summarized in Table 2.

There are multiple types of virtualization in the computing field, among which computation virtualization, storage virtualization, and network virtualization are most commonly used in cloud-native design. Due to the diversity of resources that can be virtualized, orchestrating all these heterogeneous types of devices is challenging. In this section, we will review virtualization technologies of different resources, their latest development, and the role they play when building cloud-native applications.

1) Computation Virtualization: Computation virtualization refers to creating an abstraction layer over computation, often in the form of VMs or containers. This is the core of all cloud-native deployments. The properties and efficiency of a specific virtualization technology deeply influence the entire deployment.

From the perspective of where the hypervisor resides, virtualization technologies can be divided into Hypervisor Type 1 and Type 2. A hypervisor is a software layer that controls the creation and execution of VMs. Type 1 hypervisors, also known as "bare-metal" hypervisors, directly run on hardware, while Type 2 hypervisors rely on an OS. Both hypervisors support unmodified guest OSs. Since Type 1 hypervisors directly communicate with hardware, they offer better performance and efficiency. Type 2 hypervisors, on the other hand, offer the best flexibility and compatibility, at the cost of a small portion of performance loss.

Another commonly used hypervisor in today's server hosting industry is KVM [27]. It is worth noting that KVM cannot be simply put into Type 1 or Type 2 hypervisor. While KVM runs in the Linux kernel and turns the kernel into a Type 1 hypervisor, the entire set of solutions does operate on an existing operating system, making it Type 2 by definition.

Table 2 Computation Virtualization Technologies

NAME	ADVANTAGES	DISADVANTAGES	COMMENTS	EXAMPLES
Type 1 hypervisor	High efficiency, unmodified OS	Low flexibility	Also called "bare metal"	VMware ESXi, Microsoft Hyper-V
Type 2 hypervisor	High compatibility, high flexibility, unmodified OS	Slightly lower efficiency than Type 1 hypervisor	Runs on OS	Oracle VirtualBox, VMware Workstation
Container	Low overhead, flexible deployment	No vendored kernel	Foundation of cloud-native systems	Docker, FreeBSD jail

Instead of running the entire OS, containers choose another approach to virtualize computation resources. This new approach has been highly successful in today's cloud-native scenarios due to its high efficiency [28]. Containers share the OS kernel with the host OS but have a dedicated userland filesystem. Moreover, the filesystem only contains necessary binaries, libraries, and resource files, so the final image could be as low as a few hundred kilobytes. In many applications, having a completely isolated environment and a dedicated kernel is, in fact, a huge overkill. Currently, most cloud-native orchestration implementations, including K8s, are based on Docker or other container engines [8], [29]. This is mainly due to several unique advantages containers possess, including simplicity, low overhead, fast deployment, and ease of design and build.

2) Network Virtualization: Network virtualization is another important level of virtualization in cloud-native. With network virtualization, traditional switches and routers are replaced with programmable devices, therefore allowing smarter operation. In this section, we will make a brief introduction to several key network virtualization technologies in the cloud-native context, including SDN, NFV, service mesh, and overlay networks, to understand how they enable efficient and intelligent network management.

SDN includes a set of techniques to decouple the data plane and control plane to enable programmatical a dynamical management of network forwarding devices, such as routers and switches. Traditionally, network operators leverage white-label devices from vendors to run their networks. This approach does not fit current quickly developing cloud-native environments due to the inflexibility and high price of vendor-made devices. A typical SDN system consists of several controllers and more forwarders. Traditional route or forward tables are replaced by unified flow tables, which are dynamically calculated by controllers and sent to forwarders. Forwarders then simply forward or drop traffic by looking up relevant table items from flow tables. The logically centralized control plane provides good visibility of the entire network, easing the management of network resources [30].

NFV is another layer of virtualization in networking technologies. While SDN decouples the data plane and control plane, NFV focuses on decoupling software and hardware. With the quick development of computation virtualization, using virtualized software to replace network devices has become possible. Together with SDN, there have been several solutions, including firewall [31] and router [32]. Furthermore, CNF, a new cloud-native aware trend of virtual network function, has emerged. It is designed to run in containers instead of VMs, with the advantages of cloud-native fully leveraged. To sum up, by utilizing NFV and SDN, network operators such as ISPs and cloud computing companies will benefit from reduced cost and improved flexibility to keep up with today's fast-evolving cloud-based trends.

3) Storage Virtualization: Storage virtualization is the technique of creating an abstraction layer over storage devices, to provide large, fast, and redundant storage pools across multiple hard disks. As I/O operations take a large portion of the entire turnaround time, storage virtualization deeply affects the efficiency of the entire system. Furthermore, data redundancy and safety are inherent requirements of cloud-native systems, which is often offered by storage. We identify three major layers of storage virtualization: host-based virtualization, storage device-based virtualization, and network-based virtualization.

Host-based virtualization is building the storage pool on the end host. An example of host-based virtualization is LVM. LVM is installed onto the OS and creates a storage pool using storage devices connected to the host. Device-based virtualization moves the virtualization layer from the OS to the device itself. A well-known example of this is RAID. There have been multiple combinations of RAID technologies (e.g., RAID-0, RAID-1, RAID-10, and RAID-60), for different requirements in regard to speed, redundancy, and other specialized needs. Finally, network-based virtualization is often used in data centers. The network-based storage pool is built as a dedicated cluster of storage devices and is connected to the end host using fast network links. As the storage cluster often lives in the same data center as the hosts, optimal performance can still be achieved. All these three solutions do not need any modification on high-level applications, as they have the same behavior as regular disk partitions. Therefore, good compatibility can be guaranteed.

To integrate low-level storage systems into containers, CSI[14] is proposed and introduced since K8s v1.9. The CSI

[14]https://kubernetes-csi.github.io/docs/

is a standard for exposing arbitrary low-level storage systems to containerized applications orchestrated by cloud orchestration systems. To start using a new type of storage system, developers of the storage system are able to create a CSI plugin, without modifying the core of the cloud orchestration system in use. This is helpful in customized cloud-native deployments.

C. General Supporting Function Modules

In this section, we demonstrate the general function modules provided by the K8s ecosystem that plays critical roles in the building and managing of cloud-native applications.

1) Security: K8s is designed with several security mechanisms to ensure the safety and confidentiality of data and resources within the cluster.

1) *RBAC:* It[15] is a security feature in K8s that enables system administrators to define specific access levels and permissions for each user or group of users within the cluster. With RBAC, it is possible to restrict certain privileges to only authorized entities.
2) *Pod security policies:* K8s provides pod security policies that restrict the behavior of containers running inside the pods. They can prevent containers from executing privileged actions and running as root. By default, K8s also isolates pods from one another, which adds an additional layer of protection.
3) *Secret management:* K8s offers a facility, named secret,[16] for securely storing and managing sensitive data, such as passwords, certificates, and keys. The secret data are encrypted at rest and in transit, and access to this data is restricted using RBAC.

2) Performance: Since the performance of containerization is mainly guaranteed by the underlying container engines, here, we mainly discuss the performance of pod-to-pod (container-to-container, pod-to-service, and so on) communications.

While SDN and NFV offer flexible network environments, they focus more on low-level communication, which is not easy to integrate with microservices. Ideally, microservices should focus on application logic, rather than low-level communications. In addition, with hundreds even thousands of microservices cooperating with each other, it is harder to manage network communications as the number grows. K8s's inbuilt network support is able to provide basic network connectivity. Nevertheless, it is more common to use third-party network implementations that plug into K8s using the CNI APIs. Typical implementations include Flannel,[17] Calico,[18] Weave,[19] and so on. Flannel is the most popular implementation to configure a layer 3 network fabric for K8s. By using Flannel, each node will be installed a binary called *flanneld*, which is responsible for allocating a subnet lease to each node out of a larger, preconfigured address space. The network built by Flannel uses VXLAN and many other cloud integrations for package forwarding [33]. Different implementations of CNI are compared in [34], [35], and [36]. K8s also has an inbuilt DNS such that *Pods* and *Services* can be discovered and visited though their domain names.

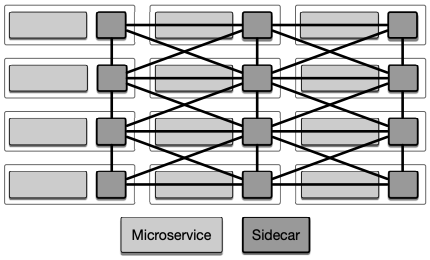


Fig. 2. ***Sidecar proxies along with each microservice form a mesh-like network.***

Note that the implementation of CNI is mandatory for building a working K8s cluster. However, to have reliable, observable, and secure communication, the vanilla network is far from enough. Under the circumstances, the service mesh is proposed, which is a software infrastructure working in *the application layer* for controlling and monitoring internal, service-to-service traffic in microservice-based applications [23]. Fig. 2 shows an illustration. A service mesh provides dynamic discovery of services, intelligent load balancing across services, security features with encryption and authentication, and observability tracing by leveraging a so-called *Sidecar* Design Pattern, where a sidecar proxy is dynamically injected into each pod for handling incoming requests. In service mesh, the control plane manages and configures proxies to route traffic and collects and consolidates data plane telemetry. Correspondingly, the data plane is implemented as sidecar proxies. As we have mentioned before, the envoy is the most popular high-performance open-source implementation of sidecar proxy.

3) Efficiency: DevOps, as a portmanteau of development and operation, is a collaborative and multidisciplinary effort within an organization to automate the continuous delivery of new software versions while guaranteeing their correctness and reliability [4]. With DevOps, development and operation teams work together across the entire cloud-native applications' life cycle. DevOps is strongly connected with CI/CD, which is a multistage pipeline of continuous integration (*build* → *test* → *merge*), continuous delivery (*automatically release the code to repository*), and continuous deployment (*automatically deploy the application in production*). DevOps has already become the cornerstone of developing microservice-based cloud-native applications nowadays.

[15]https://kubernetes.io/docs/reference/access-authn-authz/rbac/
[16]https://kubernetes.io/docs/concepts/configuration/secret/
[17]https://github.com/coreos/flannel
[18]https://github.com/projectcalico/cni-plugin
[19]https://www.weave.works/oss/net/

In the K8s ecosystem, KubeSphere DevOps is a powerful CI/CD platform that has attracted much attention from the industry [37]. KubeSphere DevOps provides CI/CD pipelines based on Jenkins.[20] It offers automation toolkits, including binary-to-image (B2I) and source-to-image (S2I), and boosts continuous delivery across K8s clusters. A key character of KubeSphere DevOps is that it scales Jenkins Agents dynamically such that the CI/CD workflows can be accelerated flexibly.

III. PERFORMANCE METRICS

By the top-down decomposition, the cloud-native architecture is mainly composed of the application layer, the platform layer, and the infrastructure layer. Different layers provide different types of services from different perspectives. First, in the application layer, various softwares are deployed as different services that can be accessed by customers over the network. The performance metrics related to the application contextual are latency, response time, completion time, makespan, and so on. Then, the platform layer provides the facilities and APIs to support the building and delivering of various services. The performance indicators related to the platform layer are scalability, stability, availability, and so on. Finally, the infrastructure layer provides the raw computing, storage, and network resources required by the service providers. The main performance metrics related to the infrastructure contextual are resource utilization, resource failure, energy consumption, and so on. All in all, the service provision should be determined by a single performance metric or jointly considering multicriteria. We describe these performance metrics in detail as follows.

A. Performance Metrics in Application Level

Various applications are encapsulated in different containers and provide services to customers by deploying these containers to physical hosts in a cloud environment. Benefiting from the convenience of installing and running the application programs, in the application layer, the key problem is to guarantee the QoS for applications that can be measured by multiple performance metrics, such as latency, response time, completion time, and makespan. These performance metrics are very similar and are often easily confused. However, in fact, they have different meanings. In this article, we distinguish these performance indicators and review the related work optimizing these indicators.

1) *Latency:* It is usually referred to as the time between when something happens and when it is perceived. Some efforts presented in [38] and [39] take latency as an important metric. For example, Zheng et al. [39] extend K8s mechanisms to support multitenants at the cost of introducing moderate latency and throughput overheads.
2) *Response time:* It refers to the time from the request submission to the result return. Response time usually consists of the transmission time of data required by the request, the queuing time, the processing time, and the result return time. Many works also consider optimizing the response time. For example, Wojciechowski et al. [40] extend the K8s mechanism to schedule pod according to dynamic network metrics, the goal of which is to reduce the application response time.
3) *JCT:* It is the time from the start time of the entry task in the job to the finish time of the last task. Different from the response time, JCT mainly concentrates on its processing time. Some efforts presented in [41], [42], [43], and [44] explicitly consider the JCT.
4) *Throughput:* It is defined as the ratio of processed requests to the total number of arrived requests at the system. A higher throughput indicates that many more requests are processed in unit time, and much less response time is incurred by each request. Hence, optimizing throughput essentially is to optimize response time as well. Many works take the throughput maximization as an optimization objective [45], [46].
5) *Mobility:* The mobility of terminals brings great challenges to service performance assurance. Specifically, the service provision problem in the driver-less scenario has attracted extensive attention from academics and industry. Many works explore a series of problems of service provision in terms of service placing, service scheduling, and service migration, aiming at guaranteeing service performance in the case of terminal moving. For instance, Chen et al. [38] explicitly investigate the influence of terminal mobility on service performance and design some strategies to guarantee service performance.
6) *SLO:* It is a key performance indicator that defines the level of service. It typically defines the minimum level of service that a provider must deliver to its customers and can be used to set expectations and establish accountability. A number of works presented in [47] and [48] try to guarantee SLO for an application.

B. Performance Metrics in Platform Level

PaaS provides all facilities and APIs to build and deliver various services conveniently, which efficiently avoids the tedious overhead incurred by downloading and installing the required software. The metrics to measure the platform service mainly include scalability, stability, reliability, and availability. These metrics characterize the performances of platform service from different perspectives. We describe these metrics and review these related studies optimizing these indicators in detail.

1) *Scalability:* It is the ability of a system to dynamically adjust the amount of resources allocated to containerized applications according to their potential

[20] https://www.jenkins.io/

workload fluctuations, which ensures that the applications are supported with enough resources to minimize SLA violations. Scaling can be performed vertically, horizontally, or both [49]. At present, some efforts presented in [49], [50], [51], [52], and [53] explicitly consider horizontal autoscaling, vertical autoscaling, and both. For example, in [54], it can create many more container replicas to meet more application resource requirements. In [55], it reallocates the amount of resource to the existing containerized application to best utilize the new hardware resource capacity.

2) *Stability:* It is a system's ability to keep a number of required properties (e.g., queue length and waiting time) within a bounded region when the system encounters some disturbances. Guaranteeing the stability of a system is a fundamental issue to ensure service performance. Therefore, some works investigate system stability problems and design various container deployment or placement approaches to guarantee service performance. For example, in [56], an adapted reinforcement learning algorithm is adopted to achieve horizontal and vertical elasticities of cloud applications for increasing the flexibility to cope with varying workloads and guarantee performance stability. In [57], a two-step algorithm is designed to solve the container deployment problem in a geodistributed computing environment. In the first step, a reinforcement learning approach is adopted to dynamically control the number of replicas of individual containers on the basis of the application response time. In the second step, a network-aware heuristic algorithm is designed to place containers on geodistributed computing resources. Its main goal is to satisfy the QoS requirements of latency-sensitive applications.
3) *Reliability:* It refers to the system's ability to deliver services without service disruption, errors, or significant reductions in performance even when one or several of its software of hardware components fail. System reliability is also a very important performance metric. Many research efforts investigate the system reliability problem and design different software and hardware schemes to optimize this performance metric. To improve the reliability of the system and reduce makespan, a heuristic algorithm is proposed to balance load among VMs in [58]. To maintain reliability and elasticity for the system, a dynamic scheduling algorithm is proposed to balance the workload of VMs in a cloud environment elastically based on resource provisioning and deprovisioning methods. The above works mainly concentrate on solving software component failure to guarantee system reliability. Different from these above works, other works concentrate on solving hard component failure problems to guarantee system reliability. De Santo et al. [59] predict the disk drives' failure and overlap the time of regular data operation and data restoring to significantly improve service reliability and reduce data center downtime.
4) *Availability:* It is the proportion of time a system is in a functioning condition. Along with scalability, stability, and reliability, availability is also a prevailing issue for platform service. To cope with possible failure caused by the mobility of parked vehicles and improve service availability, Nguyen et al. [60] design the dual cost and utility-aware heuristic algorithm to solve the problem of multireplica task scheduling in a collaborative computing paradigm consisting parked vehicles.

C. Performance Metrics in Infrastructure Level

IaaS provides the raw computing, network and storage resources, and corresponding operating middleware software to customers on demand. One of the main benefits of IaaS is free from the burden of infrastructure maintenance. In contrast to application as a service and PaaS, IaaS mainly provides the resource service at the lowest level. The performance metrics to measure the resource service mainly include resource utilization (computation, storage, and network), failure rate, interference, or energy. We describe these performance metrics and review the related research optimizing these indicators.

1) *Resource utilization:* It is an important performance metric used to describe the percentage of a system's available resource, such as computation resource, storage resource, and network resource, which is occupied over an amount of available time (or capacity). In recent years, some efforts presented in [61], [62], [63], [64], and [65] explore resource planning and resource scheduling problems with the maximization of the CPU and RAM utilization. Specifically, Beltre et al. [62] extend the K8s mechanisms to fairly allocate multiresource (such as CPU, memory, and disk) for containerized workloads of multitenants. In [66], a storage service orchestration platform is designed and implemented to support stateful applications. In [67], a workload orchestration framework is proposed to match infrastructure owner and tenants, aiming at optimizing the use of infrastructure while satisfying the application requirements. Not only that, but the network traffic is also taken as an optimization indicator [68], [69], [70]. For example, Santos et al. [70], [71] design a network-aware scheduler to automatically manage and deploy containerized applications, aiming at reducing the network latency.
2) *Interference:* In cloud-native environments, various types of workloads are encapsulated in the form of containers. However, the isolation of the container is weaker than that of the VM. Multiple containerized workloads (such as computing intensive and storage intensive) colocated on the same server can interfere

with each other, which seriously affects system performance. The interference issue incurred by colocating different types of workloads becomes a pressing issue. Therefore, many works presented in [41], [45], [72], [73], and [74] explicitly consider the inference between colocate containerized jobs. For instance, Fu et al. [41] propose a container placement scheme that balances the resource contention on the worker nodes.

3) *Energy consumption:* It is the amount of energy used. The significant amount of energy consumed by data centers can incur high costs and environmental pollution. Moreover, the energy consumption problem is also very important for resource-constrained terminals due to their limited battery capacity. In recent years, there exist research efforts on designing various energy-efficient schedulers [72], [75], [76], [77], [78]. For example, Chhikara et al. [75] propose an energy-efficient container migration scheme to migrate containers for reducing energy consumption. Kaur et al. [72] design a scheduler to minimize energy consumption and interference.
4) *Cost:* Currently, the big infrastructure service providers, such as AWS, Azure, and Alibaba, mainly adopt the pay-as-you-go payment model. Therefore, the financial cost of renting infrastructure resources is also a very import performance metric. In recent years, some research efforts presented in [79], [80], [81], and [82] take financial costs as an optimization goal and find an optimal orchestration solution by selecting diverse cloud services according to their pricing models and computing capability. Their main goals are to minimize the overall financial costs while satisfying the QoS requirements.
5) *Fault tolerance:* It is the ability of a system to behave in a well-defined manner once faults occur. Failures could occur due to dynamic changes in the execution environment. The failures in the IaaS layer or the physical hardware have a heavily negative effect on the system. Hence, it is import to design various fault tolerance mechanisms to cope with this problem and minimize the risk of failure. For instance, Kim et al. [83] develop a new container storage driver to solve the global failures and bundled performance problem.

IV. SERVICE BUILDING

In the service building state, the key steps are: 1) architecture selection and 2) code development and packing.

A. From Monolith to Microservices

Before the rise of the microservice architecture, many traditional applications adopt monolithic architectures. In this case, the application is deployed in the shape of a single-tiered monolith, which combines different components into a single program. Typical components are listed as follows.

1) *Business logic:* The application's core business logic, for example, e-commerce websites, the logic of inventory, and shipping management.
2) *Database:* The data access objects responsible for the CRUD of data.
3) *Interaction and presentation:* The component responsible for handling HTTP requests and responding with either HTML or JSON/XML (for web services APIs) objects.
4) *Integration:* The component responsible for the integration with other internal or external services though message protocols or REST APIs.

With monolithic architectures, all components are tightly coupled and run as a single service. As a result, any component of the application experiences a spike in demand, and the entire architecture has to be scaled. Besides, adding or improving a monolithic application's features becomes complex as the code base grows. This greatly increases the risk for availability since many dependent and tightly coupled components increase the impact of a single failure.

To solve the above problems, the microservice architecture is proposed, and it becomes the dominant architectural style choice for service-oriented software [3]. With a microservice architecture, an application is built as independent components that run separately as a single service. These services communicate via a well-defined interface using lightweight APIs. Services are built for business capabilities, and each service performs a single function module. Since they are independently run, each service can be updated, deployed, and scaled to meet the demand for specific functions of an application [84], [85], [86]. The microservice architecture brings in many benefits, such as agility, flexible scaling, easy deployment, and resilience. When developing cloud-native applications, the primary task is to select an appropriate architecture (monolith or microservice) based on specific business logic.

B. Packing Microservices Into Containers

Microservices are usually packaged as container images using container technologies such as Docker and then published to an image registry. Containers provide a consistent environment across different development, testing, and production stages. In K8s, the basic building block, i.e., pod, is the smallest deployable unit. A pod contains one or more tightly coupled containers that share the same network namespace and storage. Microservices are often deployed as *individual* containers within separate pods. The most popular choice for deploying these container images in K8s is Helm. Helm charts contain references to the publicly accessible container registry in order to pull the necessary container images. Nevertheless, certain companies and organizations uphold their own private cloud infrastructure. In such cases, accessing public container image registries or the Internet from within the private cloud is restricted. To deploy an application in such a limited environment, it becomes necessary to bundle all

Table 3 Representative Works in Service Orchestration

Work	App. Type	Resource			Cloud Type			Metric	Implementation
		CPU	Storage	B.W.	Single-Cloud	Multi-Cloud	Cloud-Edge		
[88]	ML	✓	✓	✓			✓	JCT	simulation
[89]		✓	✓	✓			✓	JCT	simulation
[90]		✓	✓	✓	✓			training time	simulation
[91]		✓	✓	✓	✓			JCT & makespan	system
[92]			✓		✓			makespan	system
[93]		✓			✓			JCT	simulation
[94]		✓			✓			latency & res. utilization	simulation
[45]		✓			✓			inference time	simulation
[95]		✓	✓		✓			response time	system
[96]	HPC	✓	✓		✓			response time	system
[97]		✓	✓	✓	✓			response time	simulation
[98]		✓	✓		✓			througput	simulation
[99], [100]		✓	✓	✓	✓			response time	system
[42]		✓	✓				✓	makespane	simulation
[101]		✓			✓			prediction accuracy	simulation
[102]	serverless	✓	✓	✓	✓			response time	simulation
[103]		✓	✓		✓			scalability	system
[104]		✓	✓	✓	✓			JCT	simulation
[105]		✓	✓		✓			latency & throughput	simulation
[106]			✓		✓			JCT	simulation
[107]		✓	✓		✓			JCT & res. utilization	system
[108]	batch job	✓	✓		✓			SLO & res. utilization	simulation
[109]		✓	✓	✓	✓			res. utilization	simulation
[110]		✓	✓	✓		✓		JCT	simulation
[111]		✓	✓	✓	✓			res. utilization & cost	simulation
[112]		✓	✓	✓	✓			JCT & res. utilization	simulation

the required artifacts, including container images, Helm charts, documentation, and so on, into an archive.

It is worth mentioning that if the cloud-native application is published through serverless functions, the developer only needs to upload the code to the serverless platform, and the containerization and orchestration are automatically executed by the underlying middleware and tools. Serverless computing is a method of providing backend services on an as-used basis [87]. A serverless provider allows users to write and deploy code without the hassle of worrying about the underlying infrastructure. A company that gets backend services from a serverless vendor is charged based on their computation and does not have to reserve and pay for a fixed amount of bandwidth or number of servers, as the service is autoscaling. Note that despite the name serverless, physical servers are still used, but developers do not need to be aware of them. Serverless computing allows developers to purchase backend services on a flexible "pay-as-you-go" basis, meaning that developers only have to pay for the services they use. Detailed reviews of recent works on serverless computing will be given in Section V-A3.

V. SERVICE ORCHESTRATION

Service orchestration is the automated configuration, management, and coordination of multiple microservices to deliver end-to-end services. Since microservices are encapsulated in the form of containers, service orchestration is essentially container orchestration. As a popular open-source container orchestration tool, K8s is able to automatically deploy a large number of containers and coordinate them to work together in congruence, thereby greatly reducing operational burdens. The key technology to support service orchestration lies in effective service placement and dynamic service scheduling [19], [46]. In K8s, the scheduling and placement of containers are performed by `kube-scheduler`. We illustrate the workflow of the kube-scheduler as follows: 1) the kube-scheduler maintains a queue of pods called `podQueue` that keeps listening to the `API Server`; 2) when a pod is created, the pod `metadata` is first written to etcd through the `API Server`; 3) the `kube-scheduler` watches the unbound pods from the etcd; it takes an unbound pod from the etcd and adds it into the `podQueue` at each time; 4) the fourth step is that the main process continuously extracts pods from `podQueue` and assigns them to the most suitable servers; 5) the `kube-scheduler` updates the binding information of the pod in the etcd; and 6) the kubelet component running in the selected server, which monitors the object store for assigned pods, is notified that a new pod is in pending execution and it executes the pod. Finally, the pod starts running on the selected server. The more details can be further referred to [19, sec. 3.2]. For the research academia, there are also a lot of studies about these two key technologies. The service orchestration solution is mainly affected by the characteristics of the applications and the computational architectures. Hence, these orchestration solutions can be classified based on the type of applications and the computational architectures. The subcategories match the following questions. Representative works are listed in Table 3.

1) What types of applications are orchestrated in a cloud-native system?
2) What computational architectures are used in the service orchestration?

A. Application Types

Different types of applications have significantly distinct characteristics, such as their QoS requirements, type, and

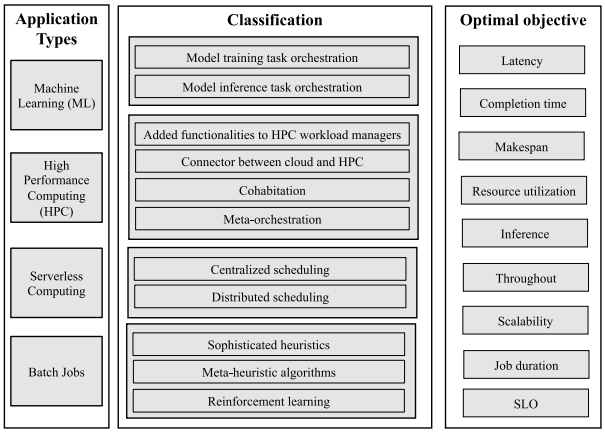


Fig. 3. ***Service orchestration under different application types.***

structure. These characteristics of the applications have a significant impact on the service orchestration. A wide range of applications from HPC, machine learning, batch, web service, or serverless is handled in a cloud-native system. In the following, we review these research studies of service orchestration for different types of applications. Fig. 3 outlines the structure of this section.

1) Machine Learning: As a subfield of machine learning, deep learning has become a very popular research topic due to its advancement in various applications. A standard deep learning development pipeline consists of model training and model inference in two stages. These two-stage tasks are characterized by unique and complicated features. Specifically, the model training task is a long-lived offline task, while the model inference task is a short-lived online task. Moreover, these two-stage tasks focus on different performance metrics, in which the model training task focuses on achieving high performance and the model inference task pays more attention to the response latency and inference accuracy. Their unique characteristics and different performance requirements impose some specific challenges to orchestrating model training tasks and model inference tasks. We comprehensively review and summarize these studies related to model training task orchestration and model inference tasks, respectively.

a) Model training task orchestration: Model training is the process of learning a model over a large dataset using a machine learning algorithm. Due to increasingly complicated models and larger datasets, model training is an extremely time-consuming and resource-consuming task. Thus, it is urgent to train deep learning models in a distributed manner. Distributed training of deep learning model is to train a neural network across multiple devices or machines, thereby accelerating the training process. The procedure of the distributed training of deep learning can vary based on the types of parallelism model training. At present, there are four types of mainstream parallel model training: data parallelism [88], [113], [114], [115], [116], model parallelism [89], pipeline parallelism [90], [117], [118], [119], and mixed parallelism [91], [120]. In the following, we illustrate these parallelism model training.

Data parallelism, as illustrated in Fig. 4, is to place multiple replicas of a model on multiple workers and divide the datasets into many subsets to feed to these multiple workers. These multiple workers simultaneously perform the model training tasks and synchronize their training results in the form of parameter servers or all-reduce and so on. By data parallelism, the speed for model training can be greatly accelerated, and the performance for model training can be enhanced. For example, in [88], a novel online preemptive scheduling framework is designed to dispatch machine learning jobs to workers and parameter servers to reduce the average JCT. Analogously, Albahar et al. [92] present a heterogeneity-aware scheduler that can efficiently collocate deep learning jobs on GPUs by exploiting the predicting information of GPU memory demand and JCTs. Its main goal is to improve the GPU resource utilization and reduce the makespan. However, with the increase in model complexity, a model with a large number of parameters cannot be launched on a single worker. Thus, model parallelism is proposed.

Model parallelism, as illustrated in Fig. 5, is to divide a model into multiple disjoint partitions and place these partitions on multiple workers. Since each worker only has one part of the model, only one worker is performing the model training task at any one time. One of the problems for the model parallelism is the long training latency incurred by the communication among multiple workers. To address this problem, a novel parallelism model training, called pipeline parallelism, is proposed.

Pipeline parallelism, as illustrated in Fig. 6, is to divide a model into multiple stages and place multiple stages on multiple workers. Besides, pipeline parallelism further divides the datasets into multiple microbatches. Multiple workers can process multiple microbatches simultaneously. Thus, pipeline parallelism training can greatly reduce the

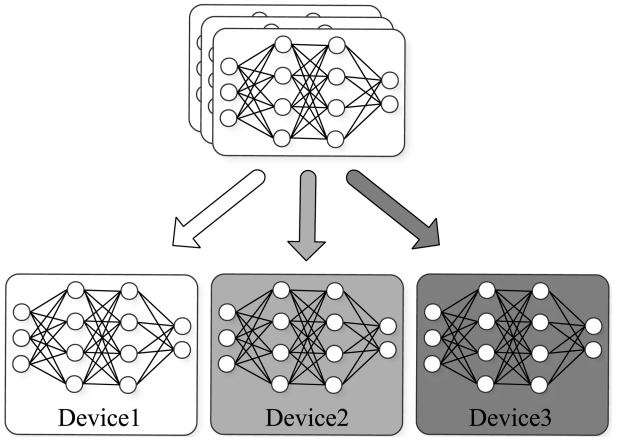


Fig. 4. ***Data parallelism.***

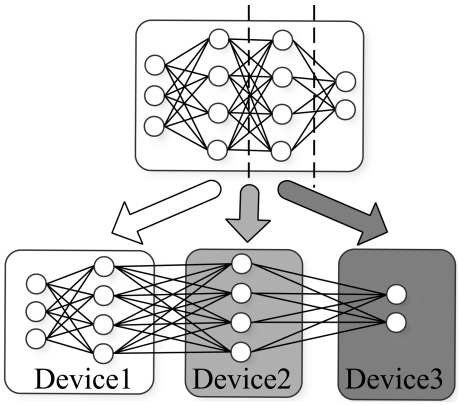


Fig. 5. ***Model parallelism.***

training latency. For example, to achieve an efficient and task-independent model parallelism, Huang et al. [90] introduce a pipeline parallelism library to partition a deep neural network across multiple workers and split the datasets into minibatches. Finally, mixed parallelism with data and model parallelism is proposed to reduce the training latency and resource consumption. For example, Wang et al. [91] design a job scheduling system that enables machine learning jobs to be implemented with data parallelism and model parallelism in clusters. The proposed system greatly reduces the JCT and improves accuracy.

b) Model inference task orchestration: Model inference is the service's ability to make predictions on new data based on a trained model. Model inference tasks are usually deployed as online short-lived services (e.g., automatic driving and face recognition). How to deploy and orchestrate model inference tasks has attracted much attention in industry and academia. For industry, many mainstream deep learning frameworks, such as TensorFlow Serving [121] and MXNet Model Server [122], have implemented the orchestration function for model inference tasks. For academia, there are also a lot of studies about model inference task orchestration. These studies mainly can be classified into two types: the individual model inference task orchestration and the multiple model inference task orchestration. For the single model inference task orchestration, Liang et al. [123] and Gholami et al. [124] design some optimization techniques to efficiently orchestrate model inference task. However, the execution of a single model inference task not only fails to meet the requirement of application scenarios but also causes a waste of resources. Thus, many researchers further study multiple model inference task orchestration. They design different heuristic-, modeling-, or prediction-based mechanisms to orchestrate multiple model inference tasks [93], [125], [126]. Specifically, Shen et al. [94] adopt a heuristic approach to select the requests to be colocated on the same GPU. First, they assign the optimal batch sizes, given the latency requirement of existing inference task requests. Finally, they establish the node runtime cycles, aiming at maximizing the resource utilization while satisfying the latency requirement. Analogously, Wu et al. [93] colocate these model inference tasks where the total of their peak GPU requirement does not exceed the capacity and heuristically schedule the newly arrived model inference task to the worker with the smallest completion time, aiming at reducing the total delay. However, all of the above studies mainly exploit the heuristic method to orchestrate certain deep learning model scenarios at a limited scale. The performance of these heuristic methods could dramatically degrade when the deep learning models vary. Thus, these heuristic methods cannot be applied to cope with dynamic colocation mechanisms for managing the inference workloads. With the complexity and dynamics of model inference tasks, many researchers turn to learning-based methods such as multiarmed bandit and reinforcement learning. For instance, Yeung et al. [45] investigate the JCT slowdown problem caused by the interference between colocated deep learning jobs. To address this problem, an interference-aware resource manager is designed to effectively colocate heterogenous deep learning jobs for improving resource utilization and job throughput.

2) High-Performance Computing: HPC jobs are usually large workloads such as large-scale financial, scientific computing, and engineering simulation. To execute these HPC jobs, an amount of computing power, memory, and network speeds tends to be required. HPC jobs are often submitted to an HPC cluster and wait to be scheduled by an HPC job scheduler. However, the existing HPC job schedulers lack microservice support and container management capacities. Therefore, it is a challenge how to efficiently support HPC workloads on K8s. In recent years, there exist research efforts on efficiently orchestrating HPC jobs on cloud clusters [127], [128], [129], [130]. These state-of-the-art studies on HPC job orchestration can be divided to four categories: added functionalities to

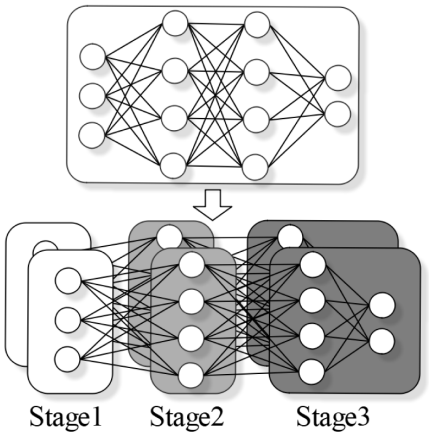


Fig. 6. ***Pipeline parallelism.***

HPC workload managers [131], [132], [133], connector between cloud and HPC [95], [134], [135], [136], cohabitation [98], [137], [138], and metaorchestration [139], [140], [141]. For added functionalities to HPC workload managers, it mainly extends HPC workload manager to support container orchestrator for HPC application. For example, Zhou et al. [95] and Misale et al. [96] investigate the problem of running HPC workloads efficiently on K8s clusters and implement a plug-in to efficiently schedule the HPC workloads. The benefit of this HPC job orchestration approach is less intrusive. However, its disadvantage is that the added functionalities are limited. To address the shortcomings of added functionalities to HPC workload managers, the connector between the cloud and HPC is proposed. The connector enables to bridge the gap between HPC and cloud systems and achieves HPC job orchestration on the cloud platform. For example, in [97], a scheduler plug-in for K8s is implemented to efficiently schedule the HPC applications on cloud platforms managed by K8s. In [142], a workflow management system is implemented to schedule the containerized HPC applications, such as nextflow, which greatly improves the numerical instability incurred by variations across computational platforms. The benefit of the connector is nonintrusive and enables to exploit orchestration strategies of orchestration platforms. However, its disadvantage is that the network latency between the cloud and HPC is high. Therefore, an HPC job orchestration approach, called cohabitation, is proposed. The cohabitation is to coexist HPC workloads manager and cloud orchestrators on an HPC cluster. For instance, Beltre et al. [98] investigate the service orchestration problem toward HPC workloads. To address this problem, the authors modify the configuration and setup of K8s to support HPC workloads and evaluate the performance of HPC workloads. The cohabitation has the advantage of fully exploiting the functionalities of orchestration platforms. However, the cohabitation is extremely intrusive. Therefore, an HPC job orchestration approach, called metaorchestration, is designed. The metaorchestration approach is to implement an additional orchestrator on top of the cloud orchestrator and HPC workload manager. For example, in [143], a framework called Kube-batch is designed to enable HPC workloads execution on K8s. In [99], an open-source tool is designed to manage the full life cycle of HPC workloads in cloud architectures. In [100], a framework that is compatibility with Prometheus is proposed to automatically deploy the benchmarking workload for containerized HPC applications and analyze their performances. The advantage of the metaorchestration approach is less intrusive. However, its disadvantage is to increase the complexity of the architecture and the efforts of maintenance.

3) Serverless Computing: Serverless computing is a new execution model of cloud computing, which is an integration of both FaaS and BaaS, allowing developers to focus on the business logic without dealing with the underlying servers. There are some key characteristics of serverless computing, such as microservices architecture, automatic scaling, and high availability. Specifically, serverless applications are often built using a microservice architecture, where each function represents a small, independent piece of functionality. This enables modular development and easier maintenance. Benefiting from its advantages, serverless computing recently attracted a lot of attention in both industry and academia. However, the inextricable dependencies between massive functions pose a great challenge to serverless orchestration. Moreover, how to automatically scale applications to response to demand and provide high availability is also another challenge to serverless orchestration. To address these challenges, there are plenty of related research efforts. In the industry community, several open-source platforms and serverless computing frameworks, such as Kubeless [144], OpenFaas [145], OpenWhisk [146], or Fission [147], are designed to support the serverless computing orchestration. These open-source frameworks with different architectures enable them to dynamically manage, scale, and provide different types of resources for serverless applications. In the academic community, some research efforts presented in [101] and [102] design diverse scheduling schemes for serverless applications. These strategies can roughly be divided into two categories: centralized scheduling [103], [104], [105] and distributed scheduling [106], [148]. For the centralized scheduler, Fan and He [101] present a double exponential smoothing approach to calculate the optimal number of pods for serverless applications. Analogously, in [102], a serverless computing framework, called Pigeon, is presented to schedule the FaaS function to prewarmed containers. Moreover, the framework introduces a static prewarmed container pool to cope with the burst function arrival. Both novelty mechanisms can greatly reduce the response time for serverless applications and improve the system's performance. Moreover, Deng et al. [42] investigate the influence of the composite property of services on the scheduling scheme at the serverless edge. To address this problem, a dependent function embedding algorithm is designed to get the optimal edge server for each function, aiming to minimize its completion time. All of these approaches are centralized. The centralized schedulers are vulnerable to a single point of failure and high communication overhead. To address these problems of the centralized scheduler, some distributed scheduling strategies are proposed. For example, Wang et al. [106] design a scheduler based on deep reinforcement learning to dynamically make decisions on the number of functions and their resources, aiming at making a tradeoff between cost and performance.

4) Batch Jobs: More and more diverse tasks are running on cloud data centers, of which batch jobs account for a large proportion. There exist many works dealing with batch job scheduling. These works are mainly carried from the system implementation and algorithm optimiza-

tion two aspects. For the works on system implementation, Gu et al. [107] design a cloud-native platform called Fluid that can co-orchestrate the data cache and deep learning jobs to improve the overall performance of multiple deep learning jobs. Analogously, in [108], a scheduling system based on the real server utilization and a sliding window-based algorithm are designed to schedule and reschedule batch jobs and, thereby, effectively improve the resource utilization in K8s. For the works on optimization algorithm, there are mainly three kinds of methods to solve it: sophisticated heuristics, metaheuristic algorithms [109], [110], [111], [112], and reinforcement learning [149], [150]. The sophisticated heuristics, such as fair scheduling [151], first-fit [152], and simple packing strategies [153], are usually easy to understand and implement. However, it needs manual adjustment to gradually improve the algorithm. Therefore, metaheuristic algorithms, such as GA or ant colony algorithm, are proposed to orchestrate batch jobs. For example, Chen et al. [109] adopt the metaheuristics optimization algorithm to schedule batch jobs to achieve higher resource utilization. In [110], a redundant placement problem for microservice-based applications is formulated to be a stochastic optimization problem. To address this problem, a GA-based server selection algorithm is designed to efficiently decide how many instances and on which edge sites to place them for each microservice. Its main goal is to reduce service execution latency and improve service availability. Analogously, in [111], a stochastic hybrid workflow scheduling algorithm is designed to jointly schedule offline batch workflows and online stream workflows in cloud container services. Its main goal is to minimize the cost and improve resource utilization in cloud container services. Moreover, Hu et al. [112] formulate the concurrent container scheduling problem to be a minimum cost flow problem. To address this problem, an efficient solution is designed to lower the average container completion time and improve resource utilization. However, the batch job orchestrations based on metaheuristic algorithms cannot efficiently cope with the dynamics of the batch jobs and the variety of the execution environment. To address this problem, reinforcement learning methods are adopted to handle dynamic orchestration problems of batch jobs. For instance, Huang et al. [149] adopt a deep reinforcement learning algorithm to schedule independent batch jobs among multiple clusters adaptively. Analogously, Gu et al. [150] propose a graph learning approach to discover the insightful properties and patterns of batch jobs. Based on these characteristics, batch jobs can be better scheduled in a production cloud computing environment.

It is worth mentioning that service orchestration for various applications can be combined with *cloud service recommendation*. This integrated approach streamlines the process of selecting and provisioning the right cloud services based on specific requirements, optimizing resource utilization, and ensuring a cohesive and well-orchestrated deployment [154], [155]. Specifically, cloud service recommendations can be part of a larger workflow that involves multiple services. Service orchestration ensures that these services are executed in the correct sequence, automating the end-to-end process from recommendation to deployment.

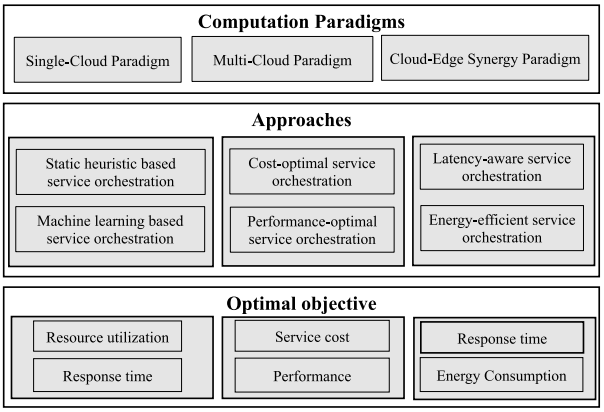


Fig. 7. ***Service orchestration under different computation paradigms.***

B. Cloud Types

Existing mainstream computing paradigms include single-cloud, multicloud, and cloud-edge synergies. Different computing paradigms have different characteristics, which have an important impact on service orchestration. Plenty of research studies have investigated service orchestration under three different computing paradigms. We categorize them by the mainstream computing paradigms and overview them. Fig. 7 outlines the structure of this section.

1) Single Cloud: The single-cloud paradigm is a new service model that delivers complex hardware and software services to external customers through the Internet [109]. In order to improve the utilization of cloud resources and reduce the response time of cloud service, efficient cloud service orchestration is the key. Currently, a large number of approaches are proposed to handle service orchestration [63], [64], [73], [79], [80], [156], [157], [158], [159], [160], [160]. These approaches can be classified into two types: static heuristic-based service orchestration [156], [157] and machine learning-based service orchestration [64], [73], [158]. For static heuristic-based service orchestration, various heuristic algorithms, including bin-packing algorithms, GA, particle swarm optimization, and so on, are adopted to orchestrate service in certain workload scenarios at a limited scale. For example, Wang et al. [156] formulate the placement of containers to be a variable-sized bin packing problem. To address this problem, an elastic scheduling algorithm for microservices in clouds is proposed. Its main goal is to minimize the cost of VMs while meeting deadline constraints. Analogously,

Zhao et al. [157] design some heuristic algorithms to schedule container cloud services to implement load balance and improve application performance. However, the performance of static heuristic algorithms could dramatically degrade when the system scales up. They cannot cope with the increasingly diverse and dynamic workloads and environments. To address this problem, machine learning algorithms, including reinforcement learning, K-means, RNN, and so on, are accordingly employed to orchestrate service. For example, in [73], a containerized task scheduler employs K-means++ algorithms to characterize the workload features and identify their behavior, and accurately colocate heterogeneous workloads in an interference-aware manner. This scheme greatly improves resource utilization and reduces the rescheduling rate. Moreover, in [64], a K8s scheduler extension that adopts a machine learning algorithm to predict QoE and schedule the resource based on the predicted QoE is designed to improve the average QoE and eliminate overprovisioning altogether in the cloud. Also, in [158], a self-adaptive K8s scheduler (re)deploys these time-sensitive applications by predicting their required resource in the cloud. These machine learning-based service orchestration schemes can build certain machine learning models for diverse and dynamic workloads and environments and predict multidimensional performance metrics. These schemes could further improve the quality of resource provisioning decisions in response to the changing workloads under complex environments.

2) Multicloud: With the surge of cloud workloads, the single-cloud paradigm cannot meet their various requirements, such as resource requirements, cost requirements, and reliability requirements. Therefore, a multicloud paradigm is proposed. The multicloud paradigm enables resources among different clouds to be shared to cope with a burst of incoming tasks. In addition, the multicloud paradigm can efficiently improve service reliability and reduce service costs. Although benefiting from these advantages of the multicloud paradigm, the heterogeneity of the underlying resources and services for different cloud systems brings some new challenges to service orchestration in the multicloud paradigm. To cope with these new challenges, some related research works about service orchestration in multicloud paradigm [161], [162], [163], [164], [165], [166] are conducted. Their main optimization objectives are service cost and service performance. According to these two optimization objectives, these research works can be divided into two types: cost-optimal service orchestration and performance-optimal service orchestration. For cost-optimal service orchestration, Rossi et al. [161], Das et al. [162], and Akhtar et al. [163] evaluate and select diverse cloud services according to their pricing models and computing capacity, and design various service orchestration strategies to minimize the financial costs. For performance-optimal service orchestration, Aldwyan et al. [164] adopt a metaheuristic algorithm to continuously make elastic container deployment plans in geographically distributed clouds and aim to maintain performance while minimizing operating costs. Also, Shi et al. [166] propose a hybrid GA-based approach to deploy a new type of composite application in multicloud. Its main goal is to optimize performance and control the budget. However, these heuristic algorithms rely on the prior knowledge of the system and cannot cope with the high variable workloads. Thus, Shi et al. [165] turn to the learning-based method. They adopt deep reinforcement learning to dispatch the new arriving requests for applications in multicloud, the goal of which is to minimize the network latency and satisfy the budget satisfaction.

3) Cloud-Edge Synergy: With the explosive growth of data generated by the terminal devices of IoT, transmitting these massive data to the remote cloud to process commonly leads to significant propagation delays, bandwidth, and energy consumption. It drives the centralized cloud to sink its computation and storage resources down to the network edge to process data, which is called the cloud-edge synergy paradigm. The cloud-edge synergy paradigm has the characteristics of resource heterogeneity, device mobility, and connection uncertainty. These characteristics bring some new challenges to the service orchestration in the cloud-edge synergy paradigm. There are plenty of researcher studies to investigate these challenges [46], [75], [162], [167], [168], [169], [170], [171], [172], [173]. Their optimization objectives mainly include response time and energy consumption. Based on their optimization objectives, we classify these studies into two types: latency-aware service orchestration and energy-efficient service orchestration. For latency-aware service orchestration, Han et al. [46], Das et al. [162], Sami et al. [167], Yan et al. [168], Wang et al. [170], and Tang et al. [174] adopt Markov decision process, reinforcement learning, deep reinforcement learning, and heuristic methods to offload the containerized applications in cloud-edge synergy paradigm. Their main goal is to optimize latency. Specifically, in [46], a learning-based scheduling framework for edge-cloud systems is designed to dispatch service requests and orchestrate multiple microservices instances, the goal of which is to improve the long-term system throughput rate. Analogously, in [169], a network-aware scheduler plugin is designed to place containerized applications on distributed cloud-edge clusters. The placement strategy of these applications considers both current network conditions and communication requirements between microservices, which is suitable for the placement of time-critical applications. For energy-efficient service orchestration, Chhikara et al. [75] adopt best-fit algorithms to place the containers, aiming to reduce energy consumption. Also, in [72], a competent controller is presented to schedule containerized applications in an edge-cloud system, aiming at minimizing interference and energy consumption.

Table 4 Representative Works in Service Operation

Work	Metric					Resource				Algorithm
	QoS	Throughput	Makespan	Response time	Revenue	CPU	RAM	B.W.	Storage	
[175]			✓			✓		✓		Heuristic
[157]		✓				✓	✓	✓	✓	Heuristic
[176]					✓	✓			✓	Game theory
[58], [177]			✓			✓				Heuristic
[178]	✓	✓				✓				LP
[179]	✓							✓	✓	Heuristic
[180]	✓					✓		✓		Approx. algorithm
[181]				✓		✓	✓			Heuristic
[182]	✓					✓				Epidemic
[183]	✓					✓		✓		Game theory
[184]			✓			✓				MARL
[185]				✓		✓				DQN
[186]		✓		✓		✓		✓		MARL
[187]				✓		✓	✓			Heuristic
[188], [189]				✓		✓	✓			-
[190]				✓		✓	✓	✓		Linear regression
[191]	✓				✓	✓		✓		MIP
[192]					✓	✓		✓		-
[193]				✓		✓		✓		-
[194]		✓		✓		✓		✓		-
[195]	✓			✓	✓	✓				RL
[196]		✓		✓		✓	✓	✓		-
[197]				✓			✓	✓		-
[198]	✓			✓			✓	✓		-
[199]				✓		✓		✓		-
[200], [201]		✓		✓	✓	✓		✓		Heuristic
[202]		✓		✓		✓	✓	✓		-
[203]		✓		✓		✓				PID
[204]				✓		✓				MAP
[205]		✓		✓		✓	✓			MPC
[206]				✓		✓	✓			PID
[207]	✓			✓		✓				Fuzzy logic
[208]	✓	✓			✓	✓		✓		FTRL
[209]	✓			✓		✓	✓			ILP
[210]	✓			✓		✓	✓	✓		Heuristic
[211]		✓		✓		✓		✓		RL

VI. SERVICE OPERATION

Service operation, which encompasses load balancing, service migration, and resource autoscaling, is crucial for maintaining a high-performing and efficient system infrastructure. K8s abstracts a group of pods with service objects. Services distribute requests to backend pods through K8s components based on preconfigured load balancing policies, such as round-robin, IP hashing, or least connections. K8s supports automatic pod scheduling and migration. By employing controllers, K8s maintains the desired state of pods such that high availability is ensured. In the case of node failure or maintenance, K8s migrates pods to available nodes without service interruption. To do this, K8s provides both horizontal and vertical scaling capabilities. Horizontal scaling is managed by HPA, which automatically adjusts the number of pods based on monitored metrics. Meanwhile, vertical scaling is handled by VPA, changing resource requests and limits for individual pods based on their resource usage. By integrating load balancing, service migration, and resource autoscaling in the operation state, we can enable robust and efficient service management dynamically and at scale. These mechanisms establish a K8s operating environment, guaranteeing applications with high availability, scalability, and reliability. Representative works are listed in Table 4.

A. Load Balancing

Load balancing is a technique used to address the problem of workload imbalance across multiple containers. It enables optimal utilization of resources, improves throughput, and reduces response time and makespan. In cloud-native environments, the primary goal of load balancing is to prevent the overloading of a single container or cluster while keeping other containers idle. Cloud-native applications with high throughput and parallel computing architectures require effective load-balancing techniques. One such technique involves redistributing heavy workloads from a single virtual server to multiple virtual servers, ensuring optimal resource utilization. In Section VI-A1, we will provide a comprehensive analysis of load-balancing algorithms. Section VI-A2 introduces the current techniques for implementing load balancing in cloud-native.

1) Algorithm Design and Analysis: Load balancing can be divided into two categories: centralized and distributed, as illustrated in Fig. 8. Centralized load balancing can further be classified into static and dynamic algorithms based on

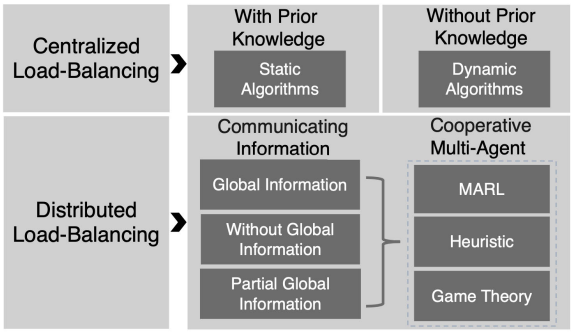


Fig. 8. ***Algorithms for load balancing.***

whether the algorithm incorporates prior knowledge of the system. On the other hand, distributed load balancing employs multiagent algorithms that offer greater flexibility and scalability than centralized algorithms.

a) Static algorithms: In cloud-native environments, static load-balancing algorithms are widely adopted due to their ability to leverage prior knowledge of system states. These algorithms, such as round-robin, first in, first out (FIFO), and MIN-MIN, do not require detailed information about the current workload running in the system. Instead, they rely on factors such as CPU usage, memory usage, storage availability, or network bandwidth utilization. Implementing static algorithms in a cloud-native system is generally straightforward. Heuristic algorithms play a significant role in developing load-balancing strategies for cloud systems under fixed conditions. For example, Miao et al. [175] propose a static algorithm called APDPSO that utilizes a particle swarm optimization method. They treat the allocation of suitable hosts as a discrete optimization problem. Similarly, Zhao et al. [157] propose a load-balancing solution based on service performance. They employ a statistical method to optimize the collaboration problem heuristically.

While static load-balancing algorithms are efficient and easy to implement, it is essential to consider server performance and continuously monitor the current load status to prevent exacerbating load imbalances during long-term execution. Regular maintenance and adjustment of the load-balancing algorithm are necessary to ensure optimal performance and resource utilization.

b) Dynamic algorithms: Dynamic load-balancing algorithms offer superior performance in adaptively executing load balancing, especially when dealing with sudden changes in workload. They can effectively operate on a cloud-native platform without prior knowledge of the system or workload. These algorithms adjust the allocation of resources dynamically based on real-time information, such as server availability and network bandwidth utilization, ensuring optimal resource utilization and reduced response times.

Load balancing involves transferring tasks to appropriate servers to alleviate the burden on overloaded servers. One crucial aspect is designing an efficient scheduling strategy to minimize the average load across all servers and reduce the makespan of the system. Lu et al. [176] propose a theoretical game algorithm that balances tasks by offloading them to edge servers while ensuring SLOs. Another approach to dynamic load balancing revolves around resource allocation. Ebadifard et al. [58] design a heuristic algorithm-based optimization algorithm that evaluates overloaded, loaded, and balanced VMs to achieve load balancing. However, these efficiency-focused load-balancing algorithms may not be suitable for large-scale VM environments.

To meet the requirements of reliability and elasticity in large-scale systems, Kumar and Sharma [177] propose a dynamic scheduling algorithm based on the last optimal k-interval VMs strategy that balances workload through resource provisioning and de-provisioning methods. This algorithm effectively balances the workload of VMs in a cloud environment through resource provisioning and deprovisioning methods. Regarding load balancing among microservices, Yu et al. [178] introduce a graph-based model to analyze dependencies among microservices and adopt a polynomial approximation method to solve the QoS-aware load-balancing optimization problem. Network traffic is another critical metric for monitoring the state of cloud-native systems, often triggering load-balancing operations. Huang et al. [179] address service unreliability and dynamic network traffic challenges by designing a load-balancing strategy based on traffic allocation consistency and DNS granularity, aiming to achieve an approximate solution to the QoS optimization problem. To efficiently route network traffic, Wang et al. [180] propose an approximation algorithm with a polynomial-time complexity that follows a two-step process to implement service deployment.

c) Multiagent algorithms: In large-scale clusters, centralized load-balancing solutions can become time-consuming due to the reliance on a single machine for decision-making. These solutions gather system information from the involved servers, which introduces delays in the decision-making process. On the other hand, distributed load-balancing schemes offer advantages in terms of scalability and flexibility, particularly in cloud-native environments. Imbalanced workloads among heterogeneous servers in the cloud can result in performance degradation within the cloud platform. To address this challenge, Gutierrez-Garcia and Ramirez-Nafarrate [181] design a distributed approach that focuses on migrating VMs to achieve load balancing. Their approach outperforms the centralized load-balancing method. However, it should be noted that collecting global information for load-balancing decisions in each agent can still be time-consuming. To tackle this issue, Menon and Kalé [182] propose a distributed load-balancing scheme that leverages partial information about the global state of the cloud system. Their scheme involves two steps: global information propagation and workload transfer. By utilizing partial

information, the load-balancing process can be expedited while still achieving effective load distribution. Another proposed scheme, F-TORA, by Xu et al. [183], focuses on task load balancing. It utilizes fuzzy neural networks and game theory to optimize task allocation and resource utilization. F-TORA aims to ensure timely and high-quality services by intelligently distributing tasks among available resources. These distributed load-balancing schemes offer advantages over centralized solutions in cloud-native environments. They provide scalability, flexibility, and improved performance by efficiently distributing workloads across servers or VMs. However, it is important to consider the specific requirements and characteristics of the system before choosing the most suitable load-balancing scheme.

The advent of intelligent algorithms, such as reinforcement learning, has opened up new possibilities for distributed load balancing. Reinforcement learning techniques, including Q-learning and MARL, have gained popularity in this domain. Yao et al. [184] propose an MARL framework that specifically addresses the dynamics of arrival workload. This framework overcomes the limitations of independent and selfish algorithms commonly used in load-balancing schemes. By leveraging MARL, the proposed approach enables agents to collaborate and make coordinated decisions, leading to more effective load balancing in dynamic workload scenarios. Asghari and Sohrabi [185] design a multiagent deep Q-network with coral reefs optimization (MDQ-CR) to minimize the energy consumption of cloud computing. This approach combines the power of deep Q-networks, a variant of reinforcement learning, with coral reefs optimization, a nature-inspired optimization algorithm. The combination of these techniques enables efficient load balancing while considering energy consumption as a critical factor. Houidi et al. [186] utilize a GNN-based method to model the network as a graph and apply MARL techniques to tackle the load-balancing problem while scheduling traffic flow. By representing the network as a graph, the authors capture the dependencies between nodes and leverage GNNs to process and aggregate information effectively. MARL techniques are then used to optimize load balancing and traffic scheduling based on the learned graph representations. These studies highlight the application of reinforcement learning, particularly MARL, in distributed load balancing. These intelligent algorithms provide a promising avenue for addressing load-balancing challenges and optimizing various aspects, such as workload dynamics, energy consumption, and traffic flow in cloud computing environments.

2) Tools and Systems: The most widely used load balancing techniques possess several desirable characteristics, including scalability, flexibility, low cost, simple deployment, and security. The load balancer allows the system to adapt to dynamic workloads by scaling in or out as needed. It should work seamlessly with various operating systems, cloud environments, and VMs and can be easily deployed. In addition, the load balancer should provide a secure environment for the system and its users. Some popular load balancing solutions include Nginx [212], a widely deployed reverse proxy server, and HAProxy [213], a fast and efficient reverse proxy software. Recent advancements in load-balancing technology include Maglev [214], which is able to balance sudden spikes in network traffic based on ECMP rules. Maglev is Google's production load balancer, which fully utilizes multiple networking techniques to achieve flexible and scalable load balancing. Specifically, Maglev utilizes Google's global backbone to announce IP prefixes at the same cost so that BGP routers can provide the first layer of load balancing. Then, IP packets are evenly distributed among service endpoints, providing another layer of load balancing. Since Maglev is entirely software-based, adding more load-balancing capability is simple as long as the backbone or service endpoints are not saturated. CHEETAH [215] is another load balancer that supports uniform load distribution with per-connection consistency.

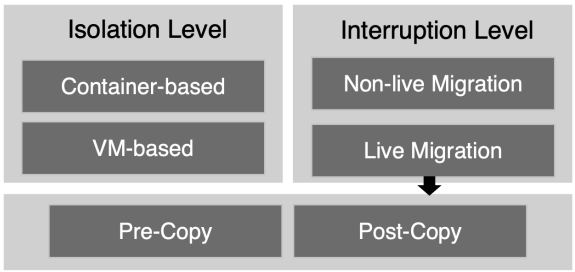


Fig. 9. *Classification of service migration.*

B. Service Migration

Service migration refers to moving the service application from the original clouds or machines to the destination. The host transfers all system states, including the memory, file system, and network connectivity profiles, to the destination host, keeping conditions without changes.

1) Algorithm Design and Analysis: Service migration is a critical aspect of cloud-native environments, encompassing live migration, VM-based migration, and container-based migration. Live migration offers minimal impact on running services and preserves memory data. VM-based migration focuses on optimizing the process through modeling, prediction, and analysis. Container-based migration benefits from efficient migration techniques and tools, enabling seamless migration of container-based services. Evaluating migration performance under various conditions is essential. Service migration enhances the flexibility, efficiency, and reliability of cloud-native systems. We introduce service migration in the following aspects, as shown in Fig. 9.

a) Live versus nonlive service migration: The main difference between live and nonlive migration is given as

follows: nonlive migration requires the system to be shut down during the migration process, while live migration involves migrating running systems. Nonlive migration is a simpler approach but does not support the preservation of memory data, leading to memory data loss. On the other hand, live migration offers the advantage of minimal impact on running services, with low interruption time and the ability to preserve data in memory. In addition, applications running in the system remain unaware of the migration. However, live migration can be a complex operation, and the migration process may encounter interruptions. Despite the challenges associated with live migration, it is widely used in cloud-native environments due to the flexibility it provides through its techniques [216].

Live migration predominantly employs two methods: precopy and postcopy [187]. The precopy migration algorithm involves iterative copy operations, which can sometimes result in the migration process failing to converge, leading to prolonged overall migration time. On the other hand, the postcopy migration algorithm offers a shorter overall migration time. However, this approach can cause page faults during the migration process, resulting in degraded performance and reduced stability of the VM [217]. To address the challenges of postcopy migration, fault tolerance becomes necessary after a failure during the recovery of a VM. One method, called PostCopyFT, tackles this issue by utilizing an efficient reverse incremental checkpoint mechanism. This approach resolves the problem without increasing the total migration time [188]. In addition, an optimized postcopy mechanism based on fabric-attached memories (FAMs) has been proposed. This mechanism, which is FAM-aware and employs system-level checkpointing, reduces both the migration time and the system's busy time [189]. These advancements in live migration techniques aim to improve the efficiency, reliability, and performance of the migration process. The choice between precopy and postcopy methods depends on factors such as migration time, system stability, fault tolerance requirements, and the impact on the VM's performance.

b) VM-based versus container-based service migration: Existing works have optimized the VM-based migration process from various perspectives, including modeling [190], [191], prediction [190], [192], latency [193], [194], energy consumption [195], and so on. The dynamics of workloads make live migration modeling challenging. Jo et al. [190] propose a machine learning-based model to enhance the prediction accuracy, considering critical characteristics of live migration. Khai et al. [191] develop a two-phase migration optimization model aimed at optimizing VM movement. The first phase computes an optimal embedding strategy to reduce demands on other virtual networks, while the second phase executes the migration using this strategy [191]. Maintaining uninterrupted uptime is crucial for live migration, particularly in large-scale systems with frequent infrastructure changes. Ruprecht et al. [192] propose a live VM migration scheme that minimizes the impact on users while addressing version updates and security concerns. Bandwidth-aware compression (BAC) focuses on the tradeoff between VM compression and transmission during migration [193]. The utilization of multipage compression techniques enables an efficient migration scheme, reducing total migration latency while maintaining performance comparable to benchmarks. To enhance performance during live migration, Le and Nahum [194] propose a new multiPath TCP method over WAN, which significantly decreases round-trip latency and improves responsiveness and user engagement. For energy consumption reduction and resource allocation in cloud-native environments, Basu et al. [195] adopt a reinforcement learning algorithm to make optimal decisions regarding VM migration. These research efforts aim to address various challenges and optimize live migration processes in terms of prediction accuracy, network optimization, uninterrupted uptime, bandwidth management, performance improvement, and resource allocation.

Container-based migration has gained popularity in cloud-native environments compared to VM-based migration [196], [197], [198], [199], [218]. Cloud-native platforms such as K8s and Docker Swarm offer efficient handoff capabilities during container migration. To reduce handoff latency during migration, Ma et al. [218] propose a framework that enables mobile users to offload their tasks to edge servers through seamless migration of container-based services. In order to provide users with the freedom to choose cloud-native platforms, Benjaponpitak et al. [196] introduce a tool called CloudHopper that facilitates the movement of containers between different platforms. Live migration is widely utilized in cloud-native platforms, but the cost of copying numerous memory pages from a source to a destination server can be high. To tackle this challenge, Sinha et al. [197] present mWarp, a live container migration tool that efficiently remaps the physical memory of containers. Xu et al. [198] propose an efficient live migration system called Sledge, which integrates images and management context to reduce migration overhead and improve QoS with minimal downtime. The system employs a dynamic context-loading mechanism to minimize downtime during migration. Although containers boot faster than VMs, their behavior during live migration under nonideal conditions remains a question. Torre et al. [199] develop a testbed to evaluate latency and downtime during live container migration in adjusted conditions. They find that network overload significantly impacts migration performance while stressing a container within a host has minimal effect [199]. These advancements in container-based migration address various challenges and offer solutions for efficient handoff, provider flexibility, memory optimization, migration overhead reduction, and evaluation of migration performance in different conditions.

2) Tools and Systems: Service migration aims to solve problems such as upgrade during service operations,

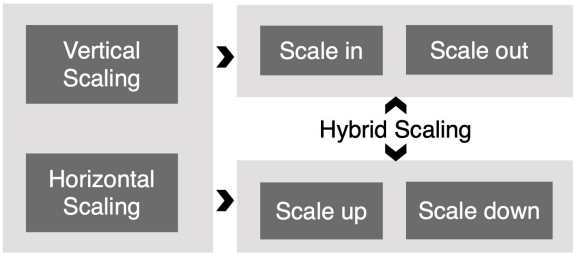


Fig. 10. *Techniques for resource autoscaling.*

load balancing between clusters, and service deployment between cloud vendors. The most popular hypervisors used for migration in cloud-native are given as follows.

1) *KVM* is a module in the Linux kernel used to visualize physical machines. It enables the host machine to turn into a hypervisor running multiple isolated virtual environments. KVM was first announced in 2006 and merged into Linux kernel releases a year later [27].
2) *Xen* focuses on the virtualization technology that supports multiple cloud platforms. The most significant feature of Xen is that it can support multiple guest operating systems, for instance, Linux, Windows, NetBSD, FreeBSD, and so on. It allows live migration between multiple hosts seamlessly [219].
3) *OpenVZ* is a virtualization technology for Linux based on an operating system level. It can support multiple operating systems and allow live container migration using checkpointing features with little delay [220].
4) *Checkpoint/restore in userspace* (CRIU) is a software tool for Linux to freeze the system states by a checkpoint technology. With CRIU, we can operate live migration in user space, which is mainly distinctive to other migration tools. During live migration, CRIU can convert the frozen running applications into a collection of files and then restore them in the checkpoint frozen [221].

C. Resource Autoscaling

With the *pay-as-you-go* principle, a cloud vendor allows applications to dynamically acquire or release their resources on their demands. Thus, the application provider can leverage the *autoscaling* method to efficiently utilize the elastic feature of resources according to its budget and profit. This section introduces *autoscaling* in three categories, i.e., vertical scaling, horizontal scaling, and hybrid autoscaling in Section VI-C1. Tools and systems are presented in Section VI-C2.

1) Algorithm Design and Analysis: According to different policies adopted by autoscaling, we summarize autoscaling into three categories, as shown in Fig. 10.

a) Horizontal: Horizontal scaling is a widely adopted approach for autoscaling in cloud-native environments, enabling applications to dynamically adjust the number of VMs to scale resources. Researchers have explored cost-efficient methods [200], [201] and fast scaling approaches [202], [222] in this domain. Cost-effectiveness is a crucial consideration in implementing horizontal scaling in cloud-native systems. Romero et al. [200] introduce INFaaS that optimizes resource cost efficiency for machine learning inference applications with evolving dynamic requirements. Zhang et al. [201] propose a solution for autoscaling in ML-as-a-Service using an LSTM network for workload prediction and a heuristic method for optimal instance provisioning decisions. The ability to respond quickly with guaranteed response times is a key requirement in cloud-native platforms. Somma et al. [222] present a fast resource provisioning method consisting of deploying containers responsible for application services and autoscaling resource allocations among containers. Shillaker and Pietzuch [202] design Faaslets, a lightweight horizontal-scaling approach for containers in clusters. In the context of virtualized network functions (VNFs) in cloud-native systems, autoscaling techniques have a significant impact on on-the-fly provisioned infrastructure performance. Salhab et al. [223] propose a framework to address resource provisioning on-demand for the 5G core network through autoscaling of constrained resources. Akhtar et al. [203] design a horizontal scaling manager on virtualized infrastructure to balance traffic workload for security network functions. These research efforts contribute to addressing the challenges of cost-effectiveness, fast scaling, and resource provisioning in horizontal scaling for cloud-native environments, benefiting applications with improved efficiency, responsiveness, and performance.

b) Vertical: Vertical scaling, which involves adjusting resources within an individual VM such as CPU, RAM, and storage, allows applications to modify their serviceability. Recent studies have focused on optimizing the cost and efficiency of vertical scaling in cloud-native environments [204], [205], [206], [207]. To achieve cost-effective resource allocation in vertical scaling, Russo et al. [204] propose MEAD that utilizes a prediction algorithm based on Markovian arrival processes to handle bursty workloads, along with an autoscaling module for resource allocation. Lakew et al. [205] address the resource allocation problem by employing a fine-grained vertical scaling technique that adapts to varying workloads in cloud-native systems. Efficiency improvement in vertical scaling has also been a focus of research. Sfakianakis et al. [206] introduce LatEst, a vertical scaling strategy that predicts bursts in serverless cloud systems and allocates resources efficiently within minimal time. Tesfatsion et al. [207] aim to increase resource usage and reduce energy consumption through long-term optimization using vertical scaling techniques. In the context of avoiding overloaded VNFs, Fei et al. [208] propose an approximation algorithm that minimizes the prediction error caused by VNF workload, followed by the implementation of a vertical scaling technique to achieve load balancing for VNFs. These studies contribute to the

optimization of cost, efficiency, and workload management in vertical scaling, enabling applications to adapt their resource allocation dynamically within individual VMs in cloud-native environments.

c) Hybrid: Hybrid scaling is a method commonly used in cloud-native networks that combine horizontal and vertical scaling mechanisms simultaneously. This approach leverages the advantages of both horizontal scaling and vertical scaling, making it more flexible and robust in managing resource provisioning. To achieve efficient resource provisioning in hybrid scaling, Shahidinejad et al. [224] utilize the imperialist competition algorithm (ICA) and K-means methods to evaluate the workload from users. Based on these evaluations, they make optimal decisions using a combination of horizontal and vertical scaling techniques. Avgeris et al. [209] employ a control-theoretic approach to establish a hybrid scaling method that maximizes the number of offloading requests in a cloud-native edge network, aiming for efficient resource allocation. In terms of minimizing resource costs, Mahmud et al. [210] propose a framework that integrates a latency-aware and deadline-satisfied strategy in a hybrid scaling approach. This framework optimizes the number of edge nodes required to meet application requirements while minimizing resource expenses [210]. Schuler et al. [211] introduce a reinforcement learning-based algorithm to minimize resource provisioning in serverless environments. By adopting a hybrid scaling method, they dynamically adjust resources to meet the dynamic demands of users while optimizing resource allocation [211]. These studies demonstrate the benefits of hybrid scaling in cloud-native networks, allowing for efficient and cost-effective resource provisioning by combining horizontal and vertical scaling techniques.

2) Tools and Systems: The typical tools, plugins, and systems that are used for autoscaling are listed as follows.

1) HPA [225] is a fundamental horizontal-scaling strategy in the k8s framework, with the target of reallocating resources for the dynamic workload to satisfy its demand. HPA can respond to the increasing workload by running more Pods to support overloaded traffic. On the contrary, due to the decreasing workload, HPA releases its Pods to the configured minimum.
2) AWS Lambda function scaling [226] supports a commercial scaling method in the service of serverless function. Lambda can invoke a scaling strategy to avoid an overloaded service supply when the incoming traffic increases.
3) Knative Pod Autoscaler (KPA) [227] is an autoscaling method supported in the recently popular framework *Knative*. KPA offers the automated scaling of applications to fit incoming demand, even for the clusters.

D. Challenges and Research Opportunities

In cloud-native environments, load balancing, service migration, and autoscaling are essential. Load balancing optimizes resource utilization, prevents congestion, and manages workloads efficiently by considering factors such as resource allocation granularity, migration time, workload detection, and algorithm efficiency. Service migration in edge-cloud environments focuses on improving QoS, ensuring network connection continuity, and overall efficiency. Autoscaling in cloud-native platforms involves determining the optimal monitoring interval, selecting appropriate metrics for scaling decisions, and making accurate and efficient decisions based on system states and workload predictions. We summarize the main challenges in these three key problems as follows.

1) *Heterogeneous workloads:* Different workloads have different resource demands for computing and bandwidth. The resource allocation granularity is a key for the performance of load balancing. Allocating too many resources leads to waste while allocating too few resources causes congestion.
2) *Congestion detection:* Developing efficacious algorithms to predict the unknown workload is a vital issue in cloud-native. Efficient load detection can avoid network resource congestion, especially in a resource-constraint environment.
3) *Configuration management:* Migrating services often involves configuring multiple components (e.g., databases and web servers) to work together seamlessly. Keeping track of configurations and ensuring that they are properly migrated can be challenging.

VII. SERVICE MAINTENANCE

Service maintenance collects and analyzes service and system indicators, adjusts and develops strategies, and performs fault recovery. K8s provides several techniques to monitor its components and collect log information. K8s monitors clusters through a built-in tool, heapster,[21] which can deal with the metric collecting problem caused by Pod migration effectively. Besides, there are also several tools to collect log information, such as Logstash[22] and Filebeat.[23] Though they all can effectively collect logs, Filebeat consumes much less memory than Logstash. This section is organized, as shown in Fig. 11. After introducing the data collection, we describe the research status of data analysis of the cloud and the evolution based on data analysis. We summarize and classify the representative works based on their main purpose, as shown in Table 5.

A. Data Collection

Data collection aims to collect metrics of cloud service and physical infrastructure to guide strategy development. There are mainly two aspects to collect and analyze the performance of cloud service: first, monitor the whole physical infrastructure of the cloud environment; second, monitor the performance of each service. For the former,

[21]https://github.com/kubernetes-retired/heapster
[22]https://github.com/elastic/logstash
[23]https://github.com/elastic/beats

Table 5 Representative Works in Service Maintenance

Work	QoS	Server Utilization	Risk Control	Cost and Profit
[228]–[230]	✓		✓	
[231]		✓		✓
[232]	✓	✓		✓
[233]	✓	✓		
[234], [235]	✓	✓		
[59], [236]–[239]	✓		✓	
[240], [241]			✓	✓
[242]	✓		✓	
[243], [244]			✓	
[245]–[247]			✓	✓

the healthy state of disks, the usage of memory, processors, and network bandwidth are usually focused on. There are lots of mature tools to detect the metrics of cloud infrastructures, such as DX [248] and CA [249]. When analyzing the performance of cloud services, there are mainly two kinds of contexts: the temporal context and the operation context [233]. The operation context means the metrics directly related to the service at each level, from software to the kernel, for example, the cache, processors, and memory usage directly caused by the monitoring service. Besides, the temporal context is also important. The temporal context is the behavior of the services that have a resource competition relationship with the monitoring service. The challenge for collecting the operation context is that it is difficult to trace the operation from layer to layer (i.e., from service layer to kernel layer). DTrace can solve this problem by instrumenting all code [250]. Besides, Ardelean et al. [233] propagate the operation by system call *getid*. *getid* will ignore all the arguments passed to it, but the kernel trace will record the arguments. Thus, they use the arguments to inject operation information. As for temporal context, Magpie [251] collects all the requests across multiple nodes. However, it is not suitable for the cloud environment with billions of users. To reduce the overhead of Magpie, a common way is to use bursty tracing [252], which just samples partial temporal context.

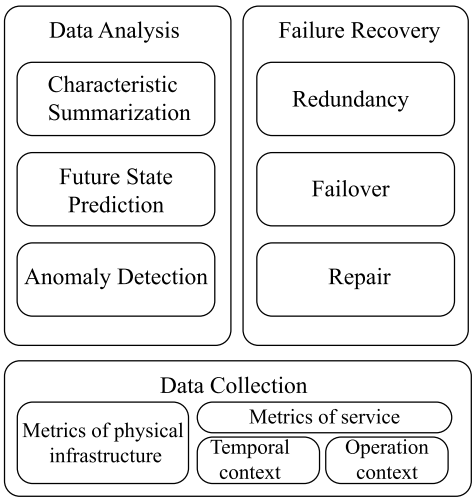


Fig. 11. ***Organization of service maintenance.***

B. Data Analysis

Data analysis at the service maintenance state is to mine the property and regular patterns of running jobs from monitoring indexes and guide job allocation and scaling. Generally, there are three main research directions for data analysis: analyze and summarize the characteristics of running jobs and anomalous events to provide a better understanding of cloud services, predict the future state of the cloud services, detect cloud service anomalies, and find root causes automatically. We organize this section as follows. In Section VII-B1, we try to answer the two questions: what kinds of data are worth and suitable to analyze and what kinds of analysis are useful for cloud providers? In Section VII-B2, we introduce the research status of system state prediction, which includes the prediction of workload, prediction of the healthy state of the disk, and prediction of service failures. In Section VII-B3, we introduce the main challenges of anomaly detection for cloud services and the research states for each challenge.

1) Summarize Cloud Services Characteristic: It is important to summarize the characteristics of cloud systems and the running cloud jobs, as it provides valuable insight to improve the utilization of servers and QoSs. There are two important questions to summarize the characteristics. First, what kind of data would contain valuable information and be suitable to be analyzed? Second, researchers and cloud service providers are interested in what kind of characteristics? For the first question, Hauswirth et al. [253] claimed that the virtualization introduced by the cloud-native environment provides a significant challenge to understanding complete system performance, not found in traditionally compiled languages. Thus, they propose vertical profiling to provide profiling of all levels of the execution stack. Vertical profiling can just apply to Java-based applications. To make it more general, Ardelean et al. [233] extend it to applications based on any language. Besides, code snippets [254], functional-level variance [255], and control flow [256] are also suitable data. For the second question, characteristic summarizations usually concentrate on reducing the cost of providers and improving the QoSs and the healthy state of cloud systems. We

list some of the most popular topics in the following: the different and similar impacts of different failures [228], the causes of failures [229], [230], the causes of low utilizations [231], pricing strategy analysis [232], and the analysis of time-varying mixture load [233].

2) Future State Prediction: The cloud system and service future state prediction on the one hand can alarm forthcoming failures and risks, so as to prevent them. On the other hand, it can provide basic information for task scheduling, autoscaling, service migration, and so on to improve the utilization of cloud infrastructures. The prediction of future system state focuses on three aspects: the workload, the healthy state of cloud servers, and the forthcoming service failures.

a) Workload prediction: Workload prediction focuses on predicting future processor usage, memory usage, and bandwidth usage, which can help to improve the QoS and improve the utilization of cloud servers. There are mainly two kinds of workload prediction methods: statistical methods and neural-network-based methods. For statistical methods, AR [257], MA [257], ARIMA [258], and Bayesian models [259] are used to predict the future workloads. Among them, ARIMA is one of the most popular and classical methods, which assumes that the value at present is affected by the trend information, the history values, and some noises. One of the problems of using ARIMA is that the workload of different jobs can have different regular patterns, and it will lead to low accuracy in predicting the workload by a single model. Thus, an adaptive statistic model [260] is proposed to solve this problem, which combines linear regression, ARIMA, and support vector regression.

Recently, it has been reported that these statistic methods rely on strong mathematical assumptions (e.g., ARIMA is based on the assumption that the time series should be stationary after difference) and predict inaccurately when the workload is highly variable [235]. Thus, many researchers turn to neural-network-based methods, such as RNN [261] and LSTM [262]. LSTM is one of the classical time series prediction methods, which can capture both the long-term dependent information and the short-term dependent information. However, these recurrent network-based methods give the same weights to the workload in the observing window, while the history workload has a different impact on predicting workload. Thus, a method that combines LSTM and attention mechanism is proposed to put different weight on history workload [234]. Besides, another problem of using recurrent network-based methods is the forgetting effect [263] when extracting long-term dependent information. Thus, a method [235] combines top-sparse autoencoder, and GRU is proposed to effectively extract the essential representations of workloads from the original high-dimensional workload data and predict highly variable workloads accurately.

b) Healthy state of cloud servers' prediction: Researchers in this domain mainly focus on disk drives' failure prediction, as it can dramatically reduce data restoring time to predict disk failures in advance. Disk drives' failure prediction plays a very important and crucial role in reducing data center downtime and significantly improving service reliability [59], as it alarms forthcoming disk drives' failure and the system can overlap the time of regular data operation and the time of data restoring. At the beginning, the task of disk drives' failure prediction is regarded as a binary classification problem, and lots of classification models are used to predict the disk failure, such as Bayesian models [264], Wilcoxon rank-sum test [265], support vector machines (SVMs) [266], and ANN [267].

However, these methods have reported poor performance in real-world environments. First, the status of disk drives corrupt gradually and is not only either good or bad [59] [236]. Thus, De Santo et al. [59] propose a method that first divides the healthy status of disk drives into seven levels by regression tree and then uses LSTM to predict the disk healthy status in the future. The fine-grained disk drivers' healthy status supports a more flexible data-restore mechanism, which can plan data restoring in advance according to the different predictions of fine-grained disk drivers' healthy status. However, LSTM used in this work is recurrent networks, and it is reported to be vulnerable to the highly variant interval between triggering events and hardware failures [237]. Thus, Sun et al. [237] use the temporal convolutional neural network (CNN) to leverage CNN's characteristic of translation invariance, which can make the CNN insensitive to various delays between triggering and failure events in the time dimension. Second, the data imbalance between disk failure data and normal data hinders the models from predicting accurately. Thus, Sun et al. [237] also designed a new loss function to prevent the gradient from vanishing in front of the huge data imbalance. Moreover, the above methods are based on offline training and cannot adapt to the continuous update systems [238]. Thus, Xiao et al. [238] propose a method based on an online random forest algorithm to maintain stable predicting accuracy for long-term usage.

c) Service failure prediction: Different from the above sections, service failure prediction focuses on service QoS and predicts the failure from the level of service. The service failure can lead to penalty payments, profit margin reduction, reputation degradation, customer churn, and service interruptions [239]. Thus, it is worthwhile to know the possible failure in advance. By doing so, the cloud systems can take steps to prevent predicting failures.

Generally, service failure prediction can be divided into three categories: rule-based methods, statistic methods, and deep learning methods. Rule-based methods rely on manually defined rules to predict failures, which are limited to the human experience and its adaptability is poor. The other two methods are data-driven methods;

compared with rule-based methods, they are more flexible and convenient.

The rule-based methods require experts to define specific rules in advance. For example, PerfAugur [268] is designed to predict failures by specified features. Generally, these methods are accurate but only suitable for specific scenarios.

The statistic methods include ARMA [269], ARIMA [270], SVM [271], hidden semi-Markov model [272], and so on. Among the classical statistic methods, Cavallo et al. [273] have claimed that ARIMA forecasting has the best compromise in ensuring a good prediction error, being sensible to outliers, and being able to predict likely violations of QoS constraints [274]. However, Amin et al. [274] point out that the traditional ARIMA model cannot deal with the high volatility of QoS properly. Thus, they propose a model that integrates GARCH and ARIMA to solve this problem.

However, statistic methods rely on some mathematics assumptions and cannot work well on high-dimensional features and dependent sequence data. Thus, many researchers apply deep learning models to service failure prediction. Chen et al. [275] propose a deep learning method-based RNN to predict task-level failures. However, the drawback of RNN is that it will definitely forget the information at long distances, which will degrade the predicting accuracy. Although some modified RNNs, such as LSTM [276], can mitigate this effect, the weights put on each value in the observing window are unequal and degrade as the distance goes farther. Thus, Gao et al. [277] propose a method based on bidirection LSTM to further improve the accuracy.

3) Anomaly Detection: In the cloud-native scene, researchers detect anomalies from different aspects: components' anomalous usage (e.g., anomalous processor and memory usage, network attacks, and disk drives' failures) and service anomalous QoS (e.g., high latency and low throughput). It is worth noticing that though anomaly detection and system state prediction both study different component usages and QoS, the prediction is to infer the forthcoming system states, while anomaly detection is to discover the anomaly already happening. The organization of this section is illustrated in Fig. 12. There are three main challenges when detecting anomaly: labeled data obtaining issues, high variance of cloud environment, and alert storm. The labeled data obtaining issues are caused by the requirements of expertise and experience and lots of efforts to label the anomalous data. The high variance of the cloud environment suggests that the detecting patterns that the models learned from history data may become outdated frequently. Besides, there are lots of APIs and components in the cloud environment. Lots of them are relevant to each other. When an anomaly occurs in one part, the related parts will also be abnormal. Thus, when an anomaly occurs, the system always suffers from an alerting storm. This phenomenon suggests the necessity of root cause finding.

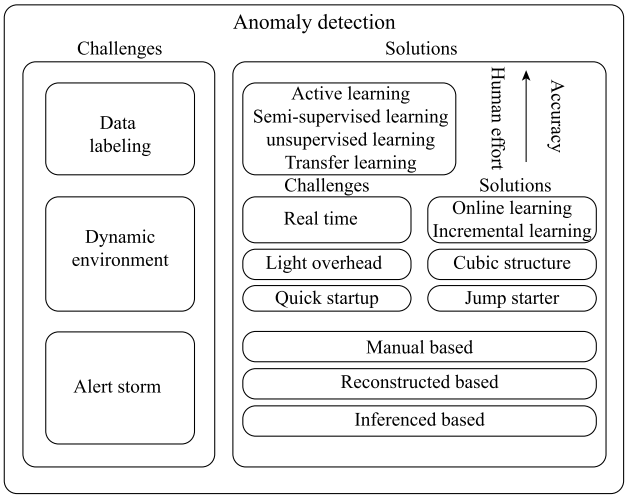


Fig. 12. ***Organization of anomaly detection.***

a) Data labeling issue: The anomalous data labeling in the cloud is different from labeling tasks in many other domains (e.g., image recognition), as it requires expertise and experience to label the anomaly, which makes the labeled data rarer. There are mainly four kinds of methods to solve it: semisupervised learning [278], [279], [280], [281], unsupervised learning [282], [283], transfer learning [240], [284], [285], and active learning [240], [241]. Semisupervised learning is suitable for datasets with a small amount of labeled data. It first uses the labeled data to train a primary model. After that, it uses the primary model to assign pseudolabel to the remaining unlabeled data. Then, it uses the data with pseudolabel and data with labels to retrain the primary model. If we not only have a dataset with a small amount of labeled data but also have experts to label some data in the process of model training, we can use active learning to further improve the detecting accuracy. Active learning first uses the labeled data to train a primary model. After that, it uses the primary model to pick a small part of unlabeled data that need label most and label the remaining data with pseudolabels. Then, experts label the data picked by the primary model. Then, it retrains the primary model with newly labeled data, labeled data, and pseudolabeled data. However, if we have no labeled data at all, we can use transfer learning or unsupervised learning. Both of them rely on assumptions: transfer learning assumes that there are some similarities between the source dataset and target dataset, and unsupervised learning assumes that the normal state has some unified and invariant latent regularity that can be learned by models. Autoencoder [286] is one of the most classical unsupervised learning models in anomaly detection. It first compresses the features into a smaller latent variable. After that, it reconstructs the features from the latent variable. In the training process, it makes the reconstructed features as similar as possible to the original features. It assumes that in the training process, the autoencoder can learn

the reconstructing patterns of normal data, and when the autoencoder meets anomalous data in the inferring process, it will fail to reconstruct. By then, the autoencoder can detect anomalies. As each feature in cloud anomaly detection is time-dependent, recently, some researchers combine the RNN with the autoencoder to capture the correlation between features and the time dependence [287], [288].

Generally speaking, the detecting accuracy of active learning, semisupervised learning, unsupervised learning, and transfer learning decreases one by one, as well as the inputting of human efforts. Therefore, the choice of model depends on the constraints and main objectives in the practical environment.

b) Cloud environment high variance issue: The cloud environment is highly variable. For example, Google and Baidu reported that thousands of software changes are deployed every day [242]. These changes will continuously degrade the predicting accuracy of anomaly detection models trained by outdated data. Frequent model retraining is costly and impractical. This problem calls for lightweight and self-evolving algorithms. The data-driven anomaly detection methods on the cloud scene can be roughly divided into two streams: one is based on time-dependent information and the other is based on the correlation between different metrics. The former is to capture the time-dependent normal pattern from time series and compare the tested data with the normal pattern, while the latter is to capture the normal correlation of different metrics and compare the correlation of tested metrics with the normal correlation. For the former anomaly detection methods, Xu et al. [244] propose a method based on online machine learning to mitigate the negative effect of cloud environment variance. Besides, they also propose a reduction method to further reduce the overhead of retraining and inferring. For the latter anomaly detection methods, Peng et al. [243] propose a method that uses a cuboid structure to store the relationship of different metrics, which significantly reduces the storing overhead. Besides, they also propose an incremental learning method suitable for cuboid structures, which significantly reduces the retraining overhead. However, it is reported the incremental retraining methods need lots of data points to converge [289] and still need more data to reach a steady state [290], which requires the cloud system to collect enough data points before retraining the models. The data collecting time is long (generally tens of days), and there will be a period called initial time [242] when the accuracy of the old model is low and there is no enough data to apply the incremental learning. To reduce the initial time, Ma et al. [242] propose a quick start method that can work well without relying on a big amount of data as incremental learning and report high accuracy.

c) Alert storm issue: The cloud system consists of thousands of components with extraordinarily complex dependencies [245]. Besides, business transactions in a cloud-native system usually have a much longer calling path with dozens of distributed microservices participating [246]. Thus, an anomaly in one part will trigger lots of anomaly alerts in related components and tasks calling the anomalous task, which is called Alert storm [246]. This problem calls for root cause-finding techniques. There are mainly three kinds of root cause-finding techniques: manual-rule-based methods, causal inference-based methods, and reconstruction-based methods. Manual-rule-based methods are based on expertise and experience, which is accurate though human-work costly and easily outdated. For example, Demirbaga et al. [247] defined three kinds of anomaly causes: data locality (i.e., data needed are not at the same server as a task), resource heterogeneity (i.e., jobs are scheduled to machine remaining few computing resources), and network failure (i.e., network disconnection). They detect these anomalies by defining the threshold of several metrics; when the defined combined metrics exceed the thresholds, the corresponding anomaly is detected. Due to the limitations of manual-rule-based methods, researchers also pay attention to data-driven methods. Some of them are just suitable for specific anomaly detection methods, such as reconstruction-based methods [282]. They can just apply to the anomaly-detecting method-based feature reconstruction. This root cause-finding method works by computing the distances between every pair of reconstructed features and original features. The greater the distance is, the more the feature contributes to the anomaly. More generally, causal inference-based methods can apply to more kinds of anomaly detection methods though they are more computationally expensive. CloudRanger [246] uses conditional dependence to establish the topology of cause and effect relationship among tasks and use the topology to find the root cause. Sipple [291] uses an integrated gradient method to compute the contribution of each feature and find the root cause.

C. Failure Recovery

One of the main advantages of cloud-native solutions lies in the possibility of automatically detecting and overcoming failures. Failure recovery is made possible by multiple technologies leveraged in building a cloud-native application. First, containerization technologies such as Docker introduce almost zero overhead when launching a container, making it possible to respawn a failed container within seconds. Second, modern metric collection and processing systems such as Prometheus [292] allow high integration into cloud-native systems, further enabling well-informed decisions. Third, with the fast development of machine learning techniques, automatically made decisions are becoming more efficient and effective. In this section, we will study three main topics to recover from an error: redundancy, failover, and repair.

1) Redundancy: Redundancy refers to deploying multiple replicas of one resource (microservice, computation, storage, and so on), preferably in multiple locations, in

order to minimize disruption even if one or several of them go down. This method has already been in use before the container-based cloud-native era [293]. In addition to improved reliability, distributing data from multiple locations can lead to reduced latency for end users. Kang et al. [294] design and implement a custom controller in K8s to select and use multiple replicas of VNFs, so high processing ability can be achieved. However, making N redundant copies means N-time of resource consumption, which could be costly. Uluyol et al. [295] propose a novel encoding mechanism to save encode data before saving to multiple locations, instead of simply creating replicas. Furthermore, the authors mitigated the increase in latency introduced by distributed storage by rethinking how consensus can be reached to offer near-optimal latency versus cost tradeoffs.

2) Failover: Failover is the process of switching to a backup server or other types of resource when disruption is detected. This process is based on the redundant deployment discussed in Section VII-C1. By having proper algorithms configured, the cloud orchestrator is able to make efficient use of replicas to switch to other healthy replicas. There are multiple optimization targets in the context of failover. Aldwyan and Sinnott [296] identify that failover between distributed data centers can lead to degraded performance due to added network latencies and propose a latency-aware failover strategy leveraging GAs to take latencies into consideration when making a failover decision. Jin et al. [297] build an SDN failover mechanism, FAVE, which is aware of physical link failure, to be used in virtualized SDN environments. Landa et al. [298] utilize TCP retransmission metrics to declare network failure in CDN networks and quickly reroute traffic through redundant links to keep high availability.

3) Repair: Repair tries to fix the error instead of redirecting traffic to other service instances. Considering the nature of today's container-based cloud-native solutions, repairing a failed service is likely to be more efficient compared to failover into backup. Giannakopoulos et al. [299] consider the complexity in modern cloud deployments and identify that such complexity could lead to failure in deployments. They build AURA that transforms a deployment into a directed acyclic graph, so whenever an error occurs, it is possible to respawn only a small portion of the entire deployment, thus keeping the repair process efficient.

D. Challenges and Research Opportunities

Commonality and Special Individuality of Data Distribution: In data-driven models; training, either for future state prediction or for anomaly detection, there is a main concern about whether to train only one model for all the servers or train a single model for each server. On the one hand, it is cost to train a model for each server. On the other hand, it predicts inaccurately when training only a model for all the servers, as every server has its own data distribution. Thus, there is a tradeoff between efficiency and accuracy.

Dynamic Evolution of Cloud Environment: The cloud environment is dynamic. New missions arrive and old missions end at every moment. The models generally become outdated and need retraining frequently. Designing a lightweight retraining method can bring huge benefits.

VIII. OPEN ISSUES AND FUTURE DIRECTIONS

Cloud-native computing has been gaining a lot of attention in recent years due to its ability to enable agile, scalable, and resilient software systems. However, there are still some open issues and future directions that need to be addressed. In the following, we list some primary open issues as follows.

1) *Hybrid multicloud integration:* As the popularity of cloud-native computing continues to grow, many organizations are using multiple cloud providers or leveraging both public and private clouds. Besides, the services and applications are deployed across the continuum of cloud-edge-device. It is important to develop better tools and techniques with interoperability capabilities for integrating these environments, including standardization of APIs and data exchange while retaining control over sensitive private data across multiple clouds.
2) *User-friendly service shapes (forms):* As cloud-native is the foundation of today's most web applications, more user-friendly service shapes to erase the heavy burden of application deployment for hybrid edge-clouds are urgently needed. Serverless computing is a good practice. It allows developers to focus on their business logic without worrying about infrastructure management. However, serverless computing is criticized for its long cold time, inefficient state management, and other related issues. Better service shapes and forms for wider application scenarios are required.
3) *Advanced automation and resource utilization:* Automation is crucial to realizing the full benefits of cloud-native architectures. However, there is still a lot of room for improvement in terms of automating deployment, scaling, and maintenance activities, especially for the distributed training of the heavy big models. Another key benefit of cloud-native computing is the ability to dynamically allocate resources based on demand. However, this can also lead to inefficiencies if not properly managed. There is a need for improved tools and algorithms that can optimize resource allocation and reduce waste.
4) *Enhanced cross-platform observability and security:* Cloud-native architectures tend to be highly distributed and dynamic, which can make it difficult to observe and troubleshoot issues. There is a need

for better observability and monitoring tools to help developers and operations teams quickly identify and resolve problems. Security-related concerns are becoming more critical, especially for hybrid multicloud scenarios. There is a string need for better security mechanisms that can effectively protect against cyber threats, especially as attacks become more sophisticated.

In response to the above challenges, there are some future research directions that can be investigated.

1) *Cross-cloud federation and interoperability:* To provide interoperability and data portability across service providers in a hybrid multicloud environment, future research could investigate methods to enhance cross-cloud federation, enabling seamless communication and data exchange between applications deployed in geographically distributed servers. Particularly, it is encouraged to study the seamless integration of edge computing and cloud-native architectures. This includes optimizing communication between edge devices and the cloud, developing edge-native applications, and addressing latency and reliability challenges in edge environments.
2) *Autonomous cloud-native systems:* To provide advanced automation in cloud-native architectures, future research is encouraged to explore the concept of autonomous cloud-native systems that can self-manage, self-heal, and optimize resource usage based on dynamic workloads. This involves research in autonomic computing, machine learning, and AI-driven automation.
3) *Resilience engineering for cloud-native systems:* To enhance observability and security, deeper knowledge of resilience engineering adapted to cloud-native computing is urgently needed. This might include research in chaos engineering, fault tolerance mechanisms, and disaster recovery strategies in the face of failures, attacks, and unpredictable conditions.
4) *Green computing in cloud-native architectures:* Further development of environmentally sustainable practices in cloud-native computing is of great importance for improving resource utilization and reducing energy consumption. This includes optimizing energy consumption, reducing carbon footprint, and developing green computing metrics for evaluating the environmental impact of cloud-native applications.
5) *Multimodal fusion in cloud-native LLMs:* Large language models (LLMs) have demonstrated substantial commercial value across diverse industries, showcasing their transformative impact on various applications. It is promising to design and develop techniques for efficient fusion of multimodal information (text, image, and audio) within LLMs deployed in cloud-native architectures, considering data storage, processing, and feature representation challenges.
6) *Quantum-based cloud-native computing:* Although still in its early stage, quantum computing has demonstrated great commercial value in many cross-disciplinary scenarios. Future research could investigate the implications of quantum computing on cloud-native architectures. It is promising to focus on adapting applications and algorithms for quantum computing, addressing security challenges, and exploring the integration of quantum resources in cloud-native setups.

In summary, while cloud-native computing has come a long way, there are still many open issues and future directions that need to be addressed to fully realize its potential. By continuing to innovate and address these challenges, we can create more efficient, secure, and scalable software systems for the future.

IX. CONCLUSION

Cloud-native, as the most influential principle for web applications, has attracted more and more researchers and companies to get involved in studying and using it. This survey attempts to provide possible research opportunities through a succinct and effective classification. We present the research roadmap of cloud-native from the perspective of services computing. Specifically, we divide the development of cloud-native applications into four states: building, orchestration, operation, and maintenance. State-of-the-art research works and industrial applications are provided. We attempted to provide some enlightening thoughts on the research of cloud-native computing and services computing. We hope that this article can stimulate fruitful discussions on potential future research directions on this topic. ■

REFERENCES

[1] M. P. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proc. 7th Int. Conf. Properties Appl. Dielectr. Mater.*, Dec. 2003, pp. 3–12.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: A research roadmap," *Int. J. Cooperat. Inf. Syst.*, vol. 17, no. 2, pp. 223–255, Jun. 2008.

[3] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *Proc. IEEE 9th Int. Conf. Service-Oriented Comput. Appl. (SOCA)*, Nov. 2016, pp. 44–51.

[4] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of DevOps concepts and challenges," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–35, Nov. 2020.

[5] F. Zampetti, S. Geremia, G. Bavota, and M. Di Penta, "CI/CD pipelines evolution and restructuring: A qualitative and quantitative study," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2021, pp. 471–482.

[6] S. Barlev, Z. Basil, S. Kohanim, R. Peleg, S. Regev, and A. Shulman-Peleg, "Secure yet usable: Protecting servers and Linux containers," *IBM J. Res. Develop.*, vol. 60, no. 4, pp. 12:1–12:10, Jul. 2016.

[7] M. Mattetti, A. Shulman-Peleg, Y. Allouche, A. Corradi, S. Dolev, and L. Foschini, "Securing the infrastructure and the workloads of Linux containers," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Sep. 2015, pp. 559–567.

[8] *Docker: Modernize Your Applications, Accelerate Innovation*. [Online]. Available: https://www.docker.com/

[9] *Kubernetes: Production-Grade Container Orchestration*. [Online]. Available: https://kubernetes.io/

[10] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th Eur. Conf. Comput. Syst.*, Bordeaux, France, Apr. 2015, pp. 1–17.

[11] N. Kratzke and P.-C. Quint, "Understanding cloud-native applications after 10 years of cloud computing—A systematic mapping study," *J. Syst. Softw.*, vol. 126, pp. 1–16, Apr. 2017.

[12] H. Mezni, S. Aridhi, and A. Hadjali, "The

uncertain cloud: State of the art and research challenges," *Int. J. Approx. Reasoning*, vol. 103, pp. 139–151, Dec. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888613X18300574

[13] D. Gannon, R. Barga, and N. Sundaresan, "Cloud-native applications," *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 16–21, Sep. 2017.

[14] N. Kratzke, "A brief history of cloud application architectures," *Appl. Sci.*, vol. 8, no. 8, p. 1368, Aug. 2018.

[15] G. Gil, D. Corujo, and P. Pedreiras, "Cloud native computing for Industry 4.0: Challenges and opportunities," in *Proc. 26th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2021, pp. 01–04.

[16] C. Surianarayanan and P. R. Chelliah, *Demystifying Cloud-Native Comput. Paradigm*. Cham, Switzerland: Springer, 2023, pp. 321–345, doi: 10.1007/978-3-031-32044-6_12.

[17] Q. Duan, "Intelligent and autonomous management in cloud-native future networks—A survey on related standards from an architectural perspective," *Future Internet*, vol. 13, no. 2, p. 42, 2021.

[18] A. Senthuran and S. Hettiarachchi, "A review of dynamic scalability and dynamic scheduling in cloud-native distributed stream processing systems," in *Proc. ICDSMLA*, 2020, pp. 1539–1553.

[19] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–37, Jul. 2023.

[20] B. Hindman et al., "Mesos: A platform for Fine-Grained resource sharing in the data center," in *Proc. 8th USENIX Symp. Networked Syst. Design Implement. (NSDI)* Boston, MA, USA: USENIX Association, Mar. 2011. [Online]. Available: https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center

[21] N. Naik, "Building a virtual system of systems using Docker swarm in multiple clouds," in *Proc. IEEE Int. Symp. Syst. Eng. (ISSE)*, Oct. 2016, pp. 1–3.

[22] V. K. Vavilapalli et al., "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, 2013, pp. 1–16.

[23] W. Li, Y. Lemieux, J. Gao, Z. Zhao, and Y. Han, "Service mesh: Challenges, state of the art, and future research opportunities," in *Proc. IEEE Int. Conf. Service-Oriented Syst. Eng. (SOSE)*, Apr. 2019, pp. 122–1225.

[24] F. Soppelsa and C. Kaewkasi, *Native Docker Clustering With Swarm*. Birmingham, U.K.: Packt Publishing Ltd, 2016.

[25] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, "A survey on edge computing systems and tools," *Proc. IEEE*, vol. 107, no. 8, pp. 1537–1562, Aug. 2019.

[26] W. Xu. (2022). *Test Report on Kubeedge's Support for 100,000 Edge Nodes*. [Online]. Available: https://kubeedge.io/en/blog/scalability-test-report/

[27] *Kernel-Based Virtual Machine*. Accessed: May 11, 2022. [Online]. Available: https://www.redhat.com/en/topics/virtualization/what-is-KVM

[28] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," in *Proc. IEEE Int. Conf. Cloud Eng.*, Mar. 2015, pp. 386–393.

[29] K. Kushwaha and N. Center, "How container runtimes matter in kubernetes?" NTT OSS Center, Linux Found., Yokohama, Japan, Tech. Rep., Nov. 2017.

[30] D. B. Rawat and S. R. Reddy, "Software defined networking architecture, security and energy efficiency: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 325–346, 1st Quart., 2017.

[31] J. Deng et al., "VNGuard: An NFV/SDN combination framework for provisioning and managing virtual firewalls," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2015, pp. 107–114.

[32] M. A. Harrabi, M. Jeridi, N. Amri, M. R. Jerbi, A. Jhine, and H. Khamassi, "Implementing NFV routers and SDN controllers in MPLS architecture," in *Proc. World Congr. Inf. Technol. Comput. Appl. (WCITCA)*, Jun. 2015, pp. 1–6.

[33] M. Mahalingam et al. (2020). *Virtual Extensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks Over Layer 3 Networks*. [Online]. Available: https://datatracker.ietf.org/doc/rfc7348/

[34] Y. Park, H. Yang, and Y. Kim, "Performance analysis of CNI (Container networking Interface) based container network," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2018, pp. 248–250.

[35] S. Qi, S. G. Kulkarni, and K. K. Ramakrishnan, "Assessing container network interface plugins: Functionality, performance, and scalability," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 656–671, Mar. 2021.

[36] R. Kumar and M. C. Trivedi, "Networking analysis and performance comparison of kubernetes CNI plugins," in *Advances in Computer, Communication and Computational Sciences*. Cham, Switzerland: Springer, 2021, pp. 99–109.

[37] T. K. Community. (2022). *Kubesphere Devops: A Powerful CI/CD Platform Built on Top of Kubernetes for Devops-Oriented Teams*. [Online]. Available: https://kubesphere.io/devops/

[38] H. Chen et al., "Mobility-aware offloading and resource allocation for distributed services collaboration," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2428–2443, Oct. 2022.

[39] C. Zheng, Q. Zhuang, and F. Guo, "A multi-tenant framework for cloud container services," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2021, pp. 359–369.

[40] L. Wojciechowski et al., "NetMARKS: Network metrics-AwaRe kubernetes scheduler powered by service mesh," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2021, pp. 1–9.

[41] Y. Fu et al., "Progress-based container scheduling for short-lived applications in a kubernetes cluster," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2019, pp. 278–287.

[42] S. Deng et al., "Dependent function embedding for distributed serverless edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2346–2357, Oct. 2022.

[43] M. C. Ogbuachi, C. Gore, A. Reale, P. Suskovics, and B. Kovács, "Context-aware K8S scheduler for real time distributed 5G edge computing applications," in *Proc. Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2019, pp. 1–6.

[44] G. Zhang, R. Lu, and W. Wu, "Multi-resource fair allocation for cloud federation," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun., IEEE 17th Int. Conf. Smart City, IEEE 5th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Aug. 2019, pp. 2189–2194.

[45] G. Yeung, D. Borowiec, R. Yang, A. Friday, R. Harper, and P. Garraghan, "Horus: Interference-aware and prediction-based scheduling in deep learning systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 1, pp. 88–100, Jan. 2022.

[46] Y. Han, S. Shen, X. Wang, S. Wang, and V. C. M. Leung, "Tailored learning-based scheduling for Kubernetes-oriented edge-cloud system," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2021, pp. 1–10.

[47] T. Pusztai et al., "SLO script: A novel language for implementing complex cloud-native elasticity-driven SLOs," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Sep. 2021, pp. 21–31.

[48] T. Pusztai et al., "A novel middleware for efficiently implementing complex cloud-native SLOs," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, Sep. 2021, pp. 410–420.

[49] M. Imdoukh, I. Ahmad, and M. G. Alfailakawi, "Machine learning-based auto-scaling for containerized applications," *Neural Comput. Appl.*, vol. 32, no. 13, pp. 9745–9760, Jul. 2020.

[50] R. Pinciroli, A. Ali, F. Yan, and E. Smirni, "CEDULE+: Resource management for burstable cloud instances using predictive analytics," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 945–957, Mar. 2021.

[51] S. N. A. Jawaddi, M. H. Johari, and A. Ismail, "A review of microservices autoscaling with formal verification perspective," *Software: Pract. Exper.*, vol. 52, no. 11, pp. 2476–2495, Nov. 2022.

[52] Z. Wang, X. Tang, Q. Liu, and J. Han, "Jily: Cost-aware AutoScaling of heterogeneous GPU for DNN inference in public cloud," in *Proc. IEEE 38th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Oct. 2019, pp. 1–8.

[53] J. P. K. S. Nunes, T. Bianchi, A. Y. Iwasaki, and E. Y. Nakagawa, "State of the art on microservices autoscaling: An overview," in *Proc. Anais do XLVIII Seminário Integrado de Softw. e Hardw.*, 2021, pp. 30–38.

[54] T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, "Horizontal pod autoscaling in kubernetes for elastic container orchestration," *Sensors*, vol. 20, no. 16, p. 4621, Aug. 2020.

[55] J. Liu, S. Zhang, Q. Wang, and J. Wei, "Coordinating fast concurrency adapting with autoscaling for SLO-oriented Web applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3349–3362, Dec. 2022.

[56] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in *Proc. IEEE 12th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2019, pp. 329–338.

[57] F. Rossi, V. Cardellini, F. Lo Presti, and M. Nardelli, "Geo-distributed efficient deployment of containers with kubernetes," *Comput. Commun.*, vol. 159, pp. 161–174, Jun. 2020.

[58] F. Ebadifard, S. M. Babamir, and S. Barani, "A dynamic task scheduling algorithm improved by load balancing in cloud computing," in *Proc. 6th Int. Conf. Web Res. (ICWR)*, Apr. 2020, pp. 177–183.

[59] A. De Santo, A. Galli, M. Gravina, V. Moscato, and G. Sperlì, "Deep learning for HDD health assessment: An application based on LSTM," *IEEE Trans. Comput.*, vol. 71, no. 1, pp. 69–80, Jan. 2022.

[60] K. Nguyen, S. Drew, C. Huang, and J. Zhou, "Collaborative container-based parked vehicle edge computing framework for online task offloading," in *Proc. IEEE 9th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2020, pp. 1–6.

[61] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 505–513.

[62] A. Beltre, P. Saha, and M. Govindaraju, "KubeSphere: An approach to multi-tenant fair scheduling for kubernetes clusters," in *Proc. IEEE Cloud Summit*, Aug. 2019, pp. 14–20.

[63] M. F. Bestari, A. I. Kistijantoro, and A. B. Sasmita, "Dynamic resource scheduler for distributed deep learning training in kubernetes," in *Proc. 7th Int. Conf. Advance Inform., Concepts, Theory Appl. (ICAICTA)*, Sep. 2020, pp. 1–6.

[64] M. Carvalho and D. F. Macedo, "QoE-aware container scheduler for co-located cloud environments," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2021, pp. 286–294.

[65] L. Toka, "Ultra-reliable and low-latency computing in the edge with kubernetes," *J. Grid Comput.*, vol. 19, no. 3, pp. 1–23, Sep. 2021.

[66] A. Warke, M. Mohamed, R. Engel, H. Ludwig, W. Sawdon, and L. Liu, "Storage service orchestration with container elasticity," in *Proc. IEEE 4th Int. Conf. Collaboration Internet Comput. (CIC)*, Oct. 2018, pp. 283–292.

[67] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini, and S. Cretti, "Foggy: A platform for workload orchestration in a fog computing environment," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Dec. 2017, pp. 231–234.

[68] A. C. Caminero and R. Muñoz-Mansilla, "Quality of service provision in fog computing: Network-aware scheduling of containers," *Sensors*, vol. 21, no. 12, p. 3978, Jun. 2021.

[69] N. D. Nguyen, L.-A. Phan, D.-H. Park, S. Kim, and

T. Kim, "ElasticFog: Elastic resource provisioning in container-based fog computing," *IEEE Access*, vol. 8, pp. 183879–183890, 2020.

[70] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *Proc. IEEE Conf. Netw. Softw. (NetSoft)*, Jun. 2019, pp. 351–359.

[71] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Resource provisioning in fog computing: From theory to practice," *Sensors*, vol. 19, no. 10, p. 2238, May 2019.

[72] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "KEIDS: Kubernetes-based energy and interference driven scheduler for industrial IoT in edge-cloud ecosystem," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4228–4237, May 2020.

[73] Z. Zhong, J. He, M. A. Rodriguez, S. Erfani, R. Kotagiri, and R. Buyya, "Heterogeneous task co-location in containerized cloud computing environments," in *Proc. IEEE 23rd Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2020, pp. 79–88.

[74] Y. Yim, H. Jang, and H. Jin, "QoS for best-effort batch jobs in container-based cloud," *Concurrency Comput., Pract. Exper.*, vol. 35, no. 15, Jul. 2023, Art. no. e6422.

[75] P. Chhikara, R. Tekchandani, N. Kumar, and M. S. Obaidat, "An efficient container management scheme for resource-constrained intelligent IoT devices," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12597–12609, Aug. 2021.

[76] M. Xu and R. Buyya, "Brownout approach for adaptive management of resources and applications in cloud computing systems: A taxonomy and future directions," *ACM Comput. Surveys*, vol. 52, no. 1, pp. 1–27, Jan. 2020.

[77] J. R. Gunasekaran, P. Thinakaran, N. C. Nachiappan, M. T. Kandemir, and C. R. Das, "Fifer: Tackling resource underutilization in the serverless era," in *Proc. 21st Int. Middleware Conf.*, Dec. 2020, pp. 280–295.

[78] P. Thinakaran, J. R. Gunasekaran, B. Sharma, M. T. Kandemir, and C. R. Das, "Kube-knots: Resource harvesting through dynamic container orchestration in gpu-based datacenters," in *Proc. IEEE Int. Conf. Clust. Comput.*, Sep. 2019, pp. 1–13.

[79] A. Chung, J. W. Park, and G. R. Ganger, "Stratus: Cost-aware container scheduling in the public cloud," in *Proc. ACM Symp. Cloud Comput.*, Oct. 2018, pp. 121–134.

[80] Z. Zhong and R. Buyya, "A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources," *ACM Trans. Internet Technol.*, vol. 20, no. 2, pp. 1–24, May 2020.

[81] Y. Xu, J. Yao, H.-A. Jacobsen, and H. Guan, "Cost-efficient negotiation over multiple resources with reinforcement learning," in *Proc. IEEE/ACM 25th Int. Symp. Quality Service (IWQoS)*, Jun. 2017, pp. 1–6.

[82] M. Xu, A. N. Toosi, and R. Buyya, "A self-adaptive approach for managing applications and harnessing renewable energy for sustainable cloud computing," *IEEE Trans. Sustain. Comput.*, vol. 6, no. 4, pp. 544–558, Oct. 2021.

[83] I. Kim, K. J. Oh, and Y. I. Eom, "Overlit: New storage driver for localization and specialization," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Feb. 2019, pp. 1–4.

[84] K. Gos and W. Zabierowski, "The comparison of microservice and monolithic architecture," in *Proc. IEEE XVIth Int. Conf. Perspective Technol. Methods MEMS Design (MEMSTECH)*, Apr. 2020, pp. 150–153.

[85] L. De Lauretis, "From monolithic architecture to microservices architecture," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2019, pp. 93–96.

[86] F. Ponce, G. Márquez, and H. Astudillo, "Migrating from monolithic architecture to microservices: A rapid review," in *Proc. Int. Conf. Chilean Comput. Sci. Soc. (SCCC)*, Nov. 2019, pp. 1–7.

[87] E. Jonas et al., "Cloud programming simplified: A Berkeley view on serverless computing," 2019, *arXiv:1902.03383*.

[88] N. Wang, R. Zhou, L. Jiao, R. Zhang, B. Li, and Z. Li, "Preemptive scheduling for distributed machine learning jobs in edge-cloud networks," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 8, pp. 2411–2425, Aug. 2022.

[89] A. Xu, Z. Huo, and H. Huang, "On the acceleration of deep learning model parallelism with staleness," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2088–2097.

[90] Y. Huang et al., "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–11.

[91] H. Wang, Z. Liu, and H. Shen, "Job scheduling for large-scale machine learning clusters," in *Proc. 16th Int. Conf. Emerg. Netw. EXp. Tech.*, 2020, pp. 108–120.

[92] H. Albahar, S. Dongare, Y. Du, N. Zhao, A. K. Paul, and A. R. Butt, "SchedTune: A heterogeneity-aware GPU scheduler for deep learning," in *Proc. 22nd IEEE Int. Symp. Clust. Cloud Internet Comput. (CCGRID)*, May 2022, pp. 695–705.

[93] X. Wu, H. Xu, and Y. Wang, "Irina: Accelerating DNN inference with efficient online scheduling," in *Proc. 4th Asia–Pacific Workshop Netw.*, Aug. 2020, pp. 36–43.

[94] H. Shen et al., "NEXUS: A GPU cluster engine for accelerating DNN-based video analysis," in *Proc. ACM Symp. Oper. Syst. Principle (SOSP)*, 2019, pp. 322–337.

[95] N. Zhou et al., "Container orchestration on HPC systems through kubernetes," *J. Cloud Comput.*, vol. 10, no. 1, pp. 1–14, Dec. 2021.

[96] C. Misale et al., "It's a scheduling affair: GROMACS in the cloud with the KubeFlux scheduler," in *Proc. 3rd Int. Workshop Containers New Orchestration Paradigms Isolated Environ. HPC (CANOPIE-HPC)*, Nov. 2021, pp. 10–16.

[97] C. Misale et al., "Towards standard Kubernetes scheduling interfaces for converged computing," in *Proc. Smoky Mountains Comput. Sci. Eng. Conf.* Cham, Switzerland: Springer, 2021, pp. 310–326.

[98] A. M. Beltre, P. Saha, M. Govindaraju, A. Younge, and R. E. Grant, "Enabling HPC workloads on cloud infrastructure using Kubernetes container orchestration mechanisms," in *Proc. IEEE/ACM Int. Workshop Containers New Orchestration Paradigms Isolated Environ. HPC (CANOPIE-HPC)*, Nov. 2019, pp. 11–20.

[99] S. López-Huguet, J. D. Segrelles, M. Kasztelnik, M. Bubak, and I. Blanquer, "Seamlessly managing HPC workloads through Kubernetes," in *Proc. Int. Conf. High Perform. Comput.* Cham, Switzerland: Springer, 2020, pp. 310–320.

[100] S. Choochotkaew, T. Chiba, S. Trent, T. Yoshimura, and M. Amaral, "AutoDECK: Automated declarative performance evaluation and tuning framework on kubernetes," in *Proc. IEEE 15th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2022, pp. 309–314.

[101] D. Fan and D. He, "Knative autoscaler optimize based on double exponential smoothing," in *Proc. IEEE 5th Inf. Technol. Mechatronics Eng. Conf. (ITOEC)*, Jun. 2020, pp. 614–617.

[102] W. Ling, L. Ma, C. Tian, and Z. Hu, "Pigeon: A dynamic and efficient serverless and FaaS framework for private cloud," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2019, pp. 1416–1421.

[103] K. Kaffes, N. J. Yadwadkar, and C. Kozyrakis, "Centralized core-granular scheduling for serverless functions," in *Proc. ACM Symp. Cloud Comput.*, Nov. 2019, pp. 158–164.

[104] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica, "The power of choice in Data-Aware cluster scheduling," in *Proc. 11th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2014, pp. 301–316.

[105] S. Venkataraman et al., "Drizzle: Fast and adaptable stream processing at scale," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 374–389.

[106] H. Wang, D. Niu, and B. Li, "Distributed machine learning with a serverless architecture," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2019, pp. 1288–1296.

[107] R. Gu et al., "Fluid: Dataset abstraction and elastic acceleration for cloud-native deep learning training jobs," in *Proc. IEEE 38th Int. Conf. Data Eng. (ICDE)*, May 2022, pp. 2182–2195.

[108] X. Zhang, L. Li, Y. Wang, E. Chen, and L. Shou, "Zeus: Improving resource efficiency via workload colocation for massive kubernetes clusters," *IEEE Access*, vol. 9, pp. 105192–105204, 2021.

[109] X. Chen et al., "A WOA-based optimization approach for task scheduling in cloud computing systems," *IEEE Syst. J.*, vol. 14, no. 3, pp. 3117–3128, Sep. 2020.

[110] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundant placement for microservice-based applications at the edge," 2019, *arXiv:1911.03600*.

[111] L. Ye, Y. Xia, L. Yang, and C. Yan, "SHWS: Stochastic hybrid workflows dynamic scheduling in cloud container services," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 3, pp. 2620–2636, Jul. 2022.

[112] Y. Hu, H. Zhou, C. de Laat, and Z. Zhao, "Concurrent container scheduling on heterogeneous clusters with multi-resource constraints," *Future Gener. Comput. Syst.*, vol. 102, pp. 562–573, Jan. 2020.

[113] M. Yu, Y. Tian, B. Ji, C. Wu, H. Rajan, and J. Liu, "GADGET: Online resource optimization for scheduling ring-all-reduce learning jobs," in *Proc. IEEE Conf. Comput. Commun.*, May 2022, pp. 1569–1578.

[114] Y. Mao, Y. Fu, W. Zheng, L. Cheng, Q. Liu, and D. Tao, "Speculative container scheduling for deep learning applications in a kubernetes cluster," *IEEE Syst. J.*, vol. 16, no. 3, pp. 3770–3781, Sep. 2022.

[115] W. Zheng, M. Tynes, H. Gorelick, Y. Mao, L. Cheng, and Y. Hou, "FlowCon: Elastic flow configuration for containerized deep learning applications," in *Proc. 48th Int. Conf. Parallel Process.*, Aug. 2019, pp. 1–10.

[116] H. Zheng, F. Xu, L. Chen, Z. Zhou, and F. Liu, "Cynthia: Cost-efficient cloud resource provisioning for predictable distributed deep neural network training," in *Proc. 48th Int. Conf. Parallel Process.*, Aug. 2019, pp. 1–11.

[117] D. Narayanan et al., "PipeDream: Generalized pipeline parallelism for DNN training," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, Oct. 2019, pp. 1–15.

[118] P. Li, E. Koyuncu, and H. Seferoglu, "Respipe: Resilient model-distributed DNN training at edge networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 3660–3664.

[119] Z. Luo et al., "Efficient pipeline planning for expedited distributed DNN training," 2022, *arXiv:2204.10562*.

[120] X. Yi et al., "Fast training of deep learning models over multiple GPUs," in *Proc. 21st Int. Middleware Conf.*, Dec. 2020, pp. 105–118.

[121] C. Olston et al., "TensorFlow-serving: Flexible, high-performance ML serving," 2017, *arXiv:1712.06139*.

[122] (2022). *Multi Model Server: A Tool for Serving Neural Net Models for Inference*. [Online]. Available: https://github.com/awslabs/multi-modelserver

[123] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, Oct. 2021.

[124] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021, *arXiv:2103.13630*.

[125] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *Proc. IEEE Int. Symp.*

High Perform. Comput. Archit. (HPCA), Feb. 2020, pp. 220–233.
[126] D. Mendoza, F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "Interference-aware scheduling for inference serving," in *Proc. 1st Workshop Mach. Learn. Syst.*, Apr. 2021, pp. 80–88.
[127] N. Zhou, H. Zhou, and D. Hoppe, "Containerization for high performance computing systems: Survey and prospects," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 2722–2740, Apr. 2023.
[128] A. Reuther et al., "Scalable system scheduling for HPC and big data," *J. Parallel Distrib. Comput.*, vol. 111, pp. 76–92, Jan. 2018.
[129] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, "HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 1–29, Jan. 2019.
[130] Y. Fan, "Job scheduling in high performance computing," 2021, *arXiv:2109.09269*.
[131] Q. Wofford, P. G. Bridges, and P. Widener, "A layered approach for modular container construction and orchestration in HPC environments," in *Proc. 11th Workshop Sci. Cloud Comput.*, Jun. 2020, pp. 1–8.
[132] S. Julian, M. Shuey, and S. Cook, "Containers in research: Initial experiences with lightweight infrastructure," in *Proc. XSEDE Conf. Diversity, Big Data, Sci. Scale*, Jul. 2016, pp. 1–6.
[133] J. Higgins, V. Holmes, and C. Venters, "Orchestrating Docker containers in the HPC environment," in *Proc. Int. Conf. High Perform. Comput.* Cham, Switzerland: Springer, 2015, pp. 506–513.
[134] N. Zhou, Y. Georgiou, L. Zhong, H. Zhou, and M. Pospieszny, "Container orchestration on HPC systems," in *Proc. IEEE 13th Int. Conf. Cloud Comput. (CLOUD)*, Oct. 2020, pp. 34–36.
[135] N. Zhou, "Containerization and orchestration on hpc systems," in *Sustained Simulation Performance 2019 and 2020*. Cham, Switzerland: Springer, 2021, pp. 133–147.
[136] N. Zhou et al., "Cybele: A hybrid architecture of hpc and big data for ai applications in agriculture," in *HPC, Big Data, and AI Convergence Towards Exascale*. Boca Raton, FL, USA: CRC Press, 2022, pp. 255–272.
[137] F. Liu, K. Keahey, P. Riteau, and J. Weissman, "Dynamically negotiating capacity between on-demand and batch clusters," in *Proc. SC, Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2018, pp. 493–503.
[138] M. E. Piras, L. Pireddu, M. Moro, and G. Zanetti, "Container orchestration on hpc clusters," in *Proc. Int. Conf. High Perform. Comput.* Cham, Switzerland: Springer, 2019, pp. 25–35.
[139] J. Carnero and F. J. Nieto, "Running simulations in HPC and cloud resources by implementing enhanced TOSCA workflows," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2018, pp. 431–438.
[140] E. Di Nitto et al., "An approach to support automated deployment of applications on heterogeneous cloud-HPC infrastructures," in *Proc. 22nd Int. Symp. Symbolic Numeric Algorithms Sci. Comput. (SYNASC)*, Sep. 2020, pp. 133–140.
[141] I. Colonnelli, B. Cantalupo, I. Merelli, and M. Aldinucci, "StreamFlow: Cross-breeding cloud with HPC," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 4, pp. 1723–1737, Oct. 2021.
[142] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature Biotechnol.*, vol. 35, no. 4, pp. 316–319, Apr. 2017.
[143] (2019). *Kube-Batch*. [Online]. Available: https://github.com/kubernetes-sigs/kube-batch
[144] Kubeless 2022 Project Authors. (2022). *The Kubernetes Native Serverless Framework*. [Online]. Available: https://kubeless.io/.
[145] H. Govind and H. González–Vélez, "Benchmarking serverless workloads on kubernetes," in *Proc. IEEE/ACM 21st Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, May 2021, pp. 704–712.
[146] K. Djemame, M. Parker, and D. Datsev, "Open-source serverless architectures: An evaluation of apache OpenWhisk," in *Proc. IEEE/ACM 13th Int. Conf. Utility Cloud Comput. (UCC)*, Dec. 2020, pp. 329–335.
[147] S. K. Mohanty et al., "An evaluation of open source serverless computing frameworks," *CloudCom*, vol. 2018, pp. 115–120, 2018.
[148] O. Mashayekhi, H. Qu, C. Shah, and P. Levis, "Execution templates: Caching control plane decisions for strong scaling of data analytics," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2017, pp. 513–526.
[149] J. Huang, C. Xiao, and W. Wu, "RLSK: A job scheduler for federated kubernetes clusters based on reinforcement learning," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Apr. 2020, pp. 116–123.
[150] Z. Gu, S. Tang, B. Jiang, S. Huang, Q. Guan, and S. Fu, "Characterizing job-task dependency in cloud workloads using graph learning," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Jun. 2021, pp. 288–297.
[151] (Apr. 2019). *Hadoop: Fair Scheduler*. [Online]. Available: https://hadoop.apache.org/docs/current/hadoopyarn/hadoop-yarnsite/FairScheduler.html
[152] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Trans. Comput.*, vol. 63, no. 11, pp. 2647–2660, Nov. 2014.
[153] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 455–466, Feb. 2015.
[154] F. Aznoli and N. J. Navimipour, "Cloud services recommendation: Reviewing the recent advances and suggesting the future research directions," *J. Netw. Comput. Appl.*, vol. 77, pp. 73–86, Jan. 2017.
[155] H. Nabli, R. Ben Djemaa, and I. A. Ben Amor, "Description, discovery, and recommendation of cloud services: A survey," *Service Oriented Comput. Appl.*, vol. 16, no. 3, pp. 147–166, Sep. 2022.
[156] S. Wang, Z. Ding, and C. Jiang, "Elastic scheduling for microservice applications in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 98–115, Jan. 2021.
[157] D. Zhao, M. Mohamed, and H. Ludwig, "Locality-aware scheduling for containers in cloud computing," *IEEE Trans. Cloud Comput.*, vol. 8, no. 2, pp. 635–646, Apr. 2020.
[158] L. Bulej, T. Bureš, P. Hnetynka, and D. Khalyeyev, "Self-adaptive K8S cloud controller for time-sensitive applications," in *Proc. 47th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Sep. 2021, pp. 166–169.
[159] G. Ambrosino, G. B. Fioccola, R. Canonico, and G. Ventre, "Container mapping and its impact on performance in containerized cloud environments," in *Proc. IEEE Int. Conf. Service Oriented Syst. Eng. (SOSE)*, Aug. 2020, pp. 57–64.
[160] H. Zhao, S. Deng, F. Chen, J. Yin, S. Dustdar, and A. Y. Zomaya, "Learning to schedule multi-server jobs with fluctuated processing speeds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 1, pp. 234–245, Jan. 2023.
[161] F. Rossi, V. Cardellini, and F. L. Presti, "Elastic deployment of software containers in geo-distributed computing environments," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2019, pp. 1–7.
[162] A. Das, S. Imai, S. Patterson, and M. P. Wittie, "Performance optimization for edge-cloud serverless platforms via dynamic task placement," in *Proc. 20th IEEE/ACM Int. Symp. Cluster, Cloud Internet Comput. (CCGRID)*, May 2020, pp. 41–50.
[163] N. Akhtar, A. Raza, V. Ishakian, and I. Matta, "COSE: Configuring serverless functions using statistical learning," in *Proc. IEEE INFOCOM - IEEE Conf. Comput. Commun.*, Jul. 2020, pp. 129–138.
[164] Y. Aldwyan, R. O. Sinnott, and G. T. Jayaputera, "Elastic deployment of container clusters across geographically distributed cloud data centers for web applications," *Concurrency Comput., Pract. Exper.*, vol. 33, no. 21, Nov. 2021, Art. no. e6436.
[165] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Location-aware and budget-constrained service brokering in multi-cloud via deep reinforcement learning," in *Proc. Int. Conf. Service-Oriented Comput.* Cham, Switzerland: Springer, 2021, pp. 756–764.
[166] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 8, pp. 1954–1969, Aug. 2020.
[167] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "FScaler: Automatic resource scaling of containers in fog clusters using reinforcement learning," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, Jun. 2020, pp. 1824–1829.
[168] M. Yan, X. Liang, Z. Lu, J. Wu, and W. Zhang, "HANSEL: Adaptive horizontal scaling of microservices using bi-LSTM," *Appl. Soft Comput.*, vol. 105, Jul. 2021, Art. no. 107216.
[169] A. Marchese and O. Tomarchio, "Network-aware container placement in cloud-edge Kubernetes clusters," in *Proc. 22nd IEEE Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, May 2022, pp. 859–865.
[170] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Mar. 2021.
[171] T. Pusztai et al., "Polaris scheduler: SLO- and topology-aware microservices scheduling at the edge," in *Proc. IEEE/ACM 15th Int. Conf. Utility Cloud Comput. (UCC)*, Dec. 2022, pp. 61–70.
[172] T. Pusztai, F. Rossi, and S. Dustdar, "Pogonip: Scheduling asynchronous applications on the edge," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, Sep. 2021, pp. 660–670.
[173] S. Nastic et al., "Polaris scheduler: Edge sensitive and SLO aware workload scheduling in cloud-edge-IoT clusters," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, Sep. 2021, pp. 206–216.
[174] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 712–725, Sep. 2019.
[175] Z. Miao, P. Yong, Y. Mei, Y. Quanjun, and X. Xu, "A discrete PSO-based static load balancing algorithm for distributed simulations in a cloud environment," *Future Gener. Comput. Syst.*, vol. 115, pp. 497–516, Feb. 2021.
[176] H. Lu, G. Xu, C. W. Sung, S. Mostafa, and Y. Wu, "A game theoretical balancing approach for offloaded tasks in edge datacenters," in *Proc. IEEE 42nd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2022, pp. 526–536.
[177] M. Kumar and S. C. Sharma, "Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment," *Comput. Electr. Eng.*, vol. 69, pp. 395–411, Jul. 2018.
[178] R. Yu, V. T. Kilari, G. Xue, and D. Yang, "Load balancing for interdependent IoT microservices," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 298–306.
[179] L. Huang, S. Cheng, Y. Guan, X. Zhang, and Z. Guo, "Consistent user-traffic allocation and load balancing in mobile edge caching," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 592–597.
[180] J. Wang, G. Zhao, H. Xu, H. Huang, L. Luo, and Y. Yang, "Robust service mapping in multi-tenant clouds," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2021, pp. 1–10.
[181] J. O. Gutierrez-Garcia and A. Ramirez-Nafarrate, "Agent-based load balancing in cloud data centers," *Cluster Comput.*, vol. 18, no. 3, pp. 1041–1062, Sep. 2015.
[182] H. Menon and L. Kalé, "A distributed dynamic load balancer for iterative applications," in *Proc. SC Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2013, pp. 1–11.

[183] X. Xu et al., "Game theory for distributed IoV task offloading with fuzzy neural network in edge computing," *IEEE Trans. Fuzzy Syst.*, vol. 30, no. 11, pp. 4593–4604, Nov. 2022.

[184] Z. Yao, Z. Ding, and T. Clausen, "Multi-agent reinforcement learning for network load balancing in data center," in *Proc. 31st ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2022, pp. 3594–3603.

[185] A. Asghari and M. K. Sohrabi, "Combined use of coral reefs optimization and multi-agent deep Q-network for energy-aware resource provisioning in cloud data centers using DVFS technique," *Cluster Comput.*, vol. 25, no. 1, pp. 119–140, Feb. 2022.

[186] O. Houidi, S. Bakri, and D. Zeghlache, "Multi-agent graph convolutional reinforcement learning for intelligent load balancing," in *Proc. NOMS IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2022, pp. 1–6.

[187] A. Shribman and B. Hudzia, "Pre-copy and post-copy vm live migration for memory intensive applications," in *Proc. Eur. Conf. Parallel Process.* Cham, Switzerland: Springer, 2012, pp. 539–547.

[188] D. Fernando, J. Terner, K. Gopalan, and P. Yang, "Live migration ate my VM: Recovering a virtual machine after failure of post-copy live migration," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 343–351.

[189] C. C. Chou, Y. Chen, D. Milojicic, N. Reddy, and P. Gratz, "Optimizing post-copy live migration with system-level checkpoint using fabric-attached memory," in *Proc. IEEE/ACM Workshop Memory Centric High Perform. Comput. (MCHPC)*, Nov. 2019, pp. 16–24.

[190] C. Jo, Y. Cho, and B. Egger, "A machine learning approach to live migration modeling," in *Proc. Symp. Cloud Comput. (SoCC)*, 2017, pp. 351–364.

[191] N. T. Khai, A. Baumgartner, and T. Bauschert, "A multi-step model for migration and resource reallocation in virtualized network infrastructures," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, May 2017, pp. 730–735.

[192] A. Ruprecht et al., "VM live migration at scale," *ACM SIGPLAN Notices*, vol. 53, no. 3, pp. 45–56, Dec. 2018.

[193] C. Li et al., "BAC: Bandwidth-aware compression for efficient live migration of virtual machines," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2017, pp. 1–9.

[194] F. Le and E. M. Nahum, "Experiences implementing live VM migration over the WAN with multi-path TCP," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 1090–1098.

[195] D. Basu, X. Wang, Y. Hong, H. Chen, and S. Bressan, "Learn-as-you-go with megh: Efficient live migration of virtual machines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 8, pp. 1786–1801, Aug. 2019.

[196] T. Benjaponpitak, M. Karakate, and K. Sripanidkulchai, "Enabling live migration of containerized applications across clouds," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 2529–2538.

[197] P. K. Sinha, S. S. Doddamani, H. Lu, and K. Gopalan, "MWarp: Accelerating intra-host live container migration via memory warping," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 508–513.

[198] B. Xu et al., "Sledge: Towards efficient live migration of Docker containers," in *Proc. IEEE 13th Int. Conf. Cloud Comput. (CLOUD)*, Oct. 2020, pp. 321–328.

[199] R. Torre, E. Urbano, H. Salah, G. T. Nguyen, and F. H. P. Fitzek, "Towards a better understanding of live migration performance with Docker containers," in *Proc. Eur. Wireless, 25th Eur. Wireless Conf.*, May 2019, pp. 1–6.

[200] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "$INFaaS$: Automated model-less inference serving," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2021, pp. 397–411.

[201] C. Zhang, M. Yu, W. Wang, and F. Yan, "$Mark$: Exploiting cloud services for $Cost-Effective$,$SLO-Aware$ machine learning inference serving," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2019, pp. 1049–1062.

[202] S. Shillaker and P. Pietzuch, "Faasm: Lightweight isolation for efficient stateful serverless computing," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2020, pp. 419–433.

[203] N. Akhtar, I. Matta, A. Raza, and Y. Wang, "EL-SEC: ELastic management of security applications on virtualized infrastructure," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2018, pp. 778–783.

[204] G. R. Russo, V. Cardellini, G. Casale, and F. L. Presti, "MEAD: Model-based vertical auto-scaling for data stream processing," in *Proc. IEEE/ACM 21st Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, May 2021, pp. 314–323.

[205] E. B. Lakew, A. V. Papadopoulos, M. Maggio, C. Klein, and E. Elmroth, "KPI-agnostic control for fine-grained vertical elasticity," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2017, pp. 589–598.

[206] Y. Sfakianakis, M. Marazakis, C. Kozanitis, and A. Bilas, "LatEst: Vertical elasticity for millisecond serverless execution," in *Proc. 22nd IEEE Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, May 2022, pp. 879–885.

[207] S. K. Tesfatsion, L. Tomás, and J. Tordsson, "OptiBook: Optimal resource booking for energy-efficient datacenters," in *Proc. IEEE/ACM 25th Int. Symp. Quality Service (IWQoS)*, Jun. 2017, pp. 1–10.

[208] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 486–494.

[209] M. Avgeris, D. Dechouniotis, N. Athanasopoulos, and S. Papavassiliou, "Adaptive resource allocation for computation offloading: A control-theoretic approach," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–20, May 2019.

[210] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, pp. 1–21, Feb. 2019.

[211] L. Schuler, S. Jamil, and N. Kühl, "AI-based resource allocation: Reinforcement learning for adaptive auto-scaling in Serverless environments," in *Proc. IEEE/ACM 21st Int. Symp. Clust. Cloud Internet Comput. (CCGrid)*, 2021, pp. 804–811.

[212] *F5 Nginx Management Suite*. [Online]. Available: https://www.nginx.com/

[213] *Haproxy: The Reliable, High Performance TCP/HTTP Load Balancer*. [Online]. Available: https://www.haproxy.org/

[214] D. E. Eisenbud et al., "Maglev: A fast and reliable software network load balancer," in *Proc. 13th {USENIX} Symp. Netw. Syst. Design Implement. ({NSDI} 16)*, 2016, pp. 523–535.

[215] T. Barbette et al., "A high-speed load-balancer design with guaranteed per-connection-consistency," in *Proc. 17th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2020, pp. 667–683.

[216] M. E. Elsaid, H. M. Abbas, and C. Meinel, "Virtual machines pre-copy live migration cost modeling and prediction: A survey," *Distrib. Parallel Databases*, vol. 40, nos. 2–3, pp. 441–474, Sep. 2022.

[217] M. Noshy, A. Ibrahim, and H. A. Ali, "Optimization of live virtual machine migration in cloud computing: A survey and future directions," *J. Netw. Comput. Appl.*, vol. 110, pp. 1–10, May 2018.

[218] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 2020–2033, Sep. 2019.

[219] *Xen Project Brings the Power of Virtualization Everywhere*. [Online]. Available: https://xenproject.org/

[220] *Open Source Container-Based Virtualization for Linux*. [Online]. Available: https://openvz.org/

[221] *A Project to Implement Checkpoint/Restore Functionality for Linux*. [Online]. Available: criu.org

[222] G. Somma, C. Ayimba, P. Casari, S. P. Romano, and V. Mancuso, "When less is more: Core-restricted container provisioning for serverless computing," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 1153–1159.

[223] N. Salhab, R. Rahim, and R. Langar, "NFV orchestration platform for 5G over on-the-fly provisioned infrastructure," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 971–972.

[224] A. Shahidinejad, M. Ghobaei-Arani, and M. Masdari, "Resource provisioning using workload clustering in cloud computing environment: A hybrid approach," *Cluster Comput.*, vol. 24, no. 1, pp. 319–342, Mar. 2021.

[225] *Horizontal Pod Autoscaler Walkthrough*. Accessed: Nov. 7, 2023. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

[226] *Scaling and Concurrency in Lambda*. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/operatorguide/scaling-concurrency.html

[227] *An Open-Source Enterprise-Level Solution to Build Serverless and Event Driven Applications*. [Online]. Available: https://knative.dev/docs/serving/autoscaling/kpa-specific/

[228] R. Birke, I. Giurgiu, L. Y. Chen, D. Wiesmann, and T. Engbersen, "Failure analysis of virtual and physical machines: Patterns, causes and characteristics," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2014, pp. 1–12.

[229] D. Ford et al., "Availability in globally distributed storage systems," in *Proc. 9th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2010, pp. 1–14.

[230] V. N. Padmanabhan, S. Ramabhadran, S. Agarwal, and J. Padhye, "A study of end-to-end Web access failures," in *Proc. ACM CoNEXT Conf. (CoNEXT)*, 2006, pp. 1–13.

[231] C. Lu, K. Ye, G. Xu, C.-Z. Xu, and T. Bai, "Imbalance in the cloud: An analysis on Alibaba cluster trace," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 2884–2892.

[232] Y. Jiang, M. Shahrad, D. Wentzlaff, D. H. K. Tsang, and C. Joe-Wong, "Burstable instances for clouds: Performance modeling, equilibrium analysis, and revenue maximization," *IEEE/ACM Trans. Netw.*, vol. 28, no. 6, pp. 2489–2502, Dec. 2020.

[233] D. Ardelean, A. Diwan, and C. Erdman, "Performance analysis of cloud applications," in *15th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2018, pp. 405–417.

[234] Y. Zhu, W. Zhang, Y. Chen, and H. Gao, "A novel approach to workload prediction using attention-based LSTM encoder–decoder network in cloud environment," *EURASIP J. Wireless Commun. Netw.*, vol. 2019, no. 1, pp. 1–18, Dec. 2019.

[235] Z. Chen, J. Hu, G. Min, A. Y. Zomaya, and T. El-Ghazawi, "Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 923–934, Apr. 2020.

[236] C. Xu, G. Wang, X. Liu, D. Guo, and T.-Y. Liu, "Health status assessment and failure prediction for hard drives with recurrent neural networks," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3502–3508, Nov. 2016.

[237] X. Sun et al., "System-level hardware failure prediction using deep learning," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.

[238] J. Xiao, Z. Xiong, S. Wu, Y. Yi, H. Jin, and K. Hu, "Disk failure prediction in data centers via online learning," in *Proc. 47th Int. Conf. Parallel Process.* New York, NY, USA: Association for Computing Machinery, 2018, pp. 1–10, doi: 10.1145/3225058.3225106.

[239] X. Zeng et al., "Detection of SLA violation for big data analytics applications in cloud," *IEEE Trans. Comput.*, vol. 70, no. 5, pp. 746–758, May 2021.

[240] X. Zhang et al., "Cross-dataset time series anomaly detection for cloud systems," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2019, pp. 1063–1076.

[241] T. Pimentel, M. Monteiro, A. Veloso, and N. Ziviani, "Deep active learning for anomaly detection," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–8.

[242] M. Ma et al., "$Jump - Starting$ multivariate time series anomaly detection for online service systems," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2021, pp. 413–426.

[243] H. Peng et al., "Lime: Low-cost and incremental learning for dynamic heterogeneous information networks," *IEEE Trans. Comput.*, vol. 71, no. 3, pp. 628–642, Mar. 2022.

[244] J. Xu, Y. Wang, P. Chen, and P. Wang, "Lightweight and adaptive service API performance monitoring in highly dynamic cloud environment," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2017, pp. 35–43.

[245] L. Li et al., "Fighting the fog of war: Automated incident detection for cloud systems," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2021, pp. 131–146.

[246] P. Wang et al., "CloudRanger: Root cause identification for cloud native systems," in *Proc. 18th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput.*, Washington, DC, USA, May 2018, pp. 492–502.

[247] U. Demirbaga et al., "AutoDiagn: An automated real-time diagnosis framework for big data systems," *IEEE Trans. Comput.*, vol. 71, no. 5, pp. 1035–1048, May 2022.

[248] *Dx Unified Infrastructure Management*. [Online]. Available: https://community.broadcom.com/enterprisesoftware/communities/communityhomeblogs?CommunityKey=170eb4e5-a593-4af2-ad1d-f7655e31513b

[249] *CA Unified Infrastructure Management*. [Online]. Available: http://aspiretp.com/uim/

[250] *DTrace*. [Online]. Available: http://dtrace.org/blogs/about

[251] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *Proc. OSDI*, vol. 4, 2004, p. 18.

[252] M. Arnold, M. Hind, and B. G. Ryder, "Online feedback-directed optimization of Java," *ACM SIGPLAN Notices*, vol. 37, no. 11, pp. 111–129, Nov. 2002.

[253] M. Hauswirth, P. F. Sweeney, A. Diwan, and M. Hind, "Vertical profiling: Understanding the behavior of object-priented applications," *ACM SIGPLAN Notices*, vol. 39, no. 10, pp. 251–269, Oct. 2004, doi: 10.1145/1035292.1028998.

[254] L. Zheng et al., "Vapro: Performance variance detection and diagnosis for production-run parallel applications," in *Proc. 27th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.* New York, NY, USA: Association for Computing Machinery, Apr. 2022, pp. 150–162, doi: 10.1145/3503221.3508411.

[255] P. Su, S. Jiao, M. Chabbi, and X. Liu, "Pinpointing performance inefficiencies via lightweight variance profiling," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2019, pp. 1–19.

[256] I. Laguna, D. H. Ahn, B. R. D. Supinski, S. Bagchi, and T. Gamblin, "Diagnosis of performance faults in LargeScale MPI applications via probabilistic progress-dependence inference," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1280–1289, May 2015.

[257] P. A. Dinda, "Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 2, pp. 160–173, Feb. 2006.

[258] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud Applications' QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, Oct. 2015.

[259] G. K. Shyam and S. S. Manvi, "Virtual resource prediction in cloud environment: A Bayesian approach," *J. Netw. Comput. Appl.*, vol. 65, pp. 144–154, Apr. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804516300091

[260] P. Singh, P. Gupta, and K. Jyoti, "TASM: Technocrat ARIMA and SVR model for workload prediction of web applications in cloud," *Cluster Comput.*, vol. 22, no. 2, pp. 619–633, Jun. 2019.

[261] W. Zhang, B. Li, D. Zhao, F. Gong, and Q. Lu, "Workload prediction for cloud cluster using a recurrent neural network," in *Proc. Int. Conf. Identificat., Inf. Knowl. Internet Things (IIKI)*, Oct. 2016, pp. 104–109.

[262] J. Kumar, R. Goomer, and A. K. Singh, "Long short term memory recurrent neural network (LSTM-RNN) based workload forecasting model for cloud datacenters," *Proc. Comput. Sci.*, vol. 125, pp. 676–682, Jan. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050917328557

[263] H. Zhou et al., "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proc. AAAI Conf. Artif. Intell.*, 2021, vol. 35, no. 12, pp. 11106–11115.

[264] G. Hamerly and C. Elkan, "Bayesian approaches to failure prediction for disk drives," in *Proc. ICML*, vol. 1, 2001, pp. 202–209.

[265] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan, "Improved disk-drive failure warnings," *IEEE Trans. Rel.*, vol. 51, no. 3, pp. 350–357, Sep. 2002.

[266] J. F. Murray, G. F. Hughes, K. Kreutz-Delgado, and D. Schuurmans, "Machine learning methods for predicting failures in hard drives: A multiple-instance application," *J. Mach. Learn. Res.*, vol. 6, no. 5, pp. 1–34, 2005.

[267] S. J. Russell, *Artificial Intelligence a Modern Approach*. London, U.K.: Pearson, 2010.

[268] S. Roy, A. C. König, I. Dvorkin, and M. Kumar, "PerfAugur: Robust diagnostics for performance anomalies in cloud services," in *Proc. IEEE 31st Int. Conf. Data Eng.*, Apr. 2015, pp. 1167–1178.

[269] T. Chalermarrewong, T. Achalakul, and S. C. W. See, "Failure prediction of data centers using time series and fault tree analysis," in *Proc. IEEE 18th Int. Conf. Parallel Distrib. Syst.*, Dec. 2012, pp. 794–799.

[270] M. Godse, U. Bellur, and R. Sonar, "Automating QoS based service selection," in *Proc. IEEE Int. Conf. Web Services*, Jul. 2010, pp. 534–541.

[271] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure prediction based on log files using random indexing and support vector machines," *J. Syst. Softw.*, vol. 86, no. 1, pp. 2–11, Jan. 2013.

[272] S. Fu and C.-Z. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *Proc. SC ACM/IEEE Conf. Supercomputing*, Nov. 2007, pp. 1–12.

[273] B. Cavallo, M. Di Penta, and G. Canfora, "An empirical comparison of methods to support QoS-aware service selection," in *Proc. 2nd Int. Workshop Princ. Eng. Service-Oriented Syst.*, May 2010, pp. 64–70.

[274] A. Amin, A. Colman, and L. Grunske, "An approach to forecasting QoS attributes of web services based on ARIMA and GARCH models," in *Proc. IEEE 19th Int. Conf. Web Services*, Jun. 2012, pp. 74–81.

[275] X. Chen, C.-D. Lu, and K. Pattabiraman, "Failure prediction of jobs in compute clouds: A Google cluster case study," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, Nov. 2014, pp. 341–346.

[276] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.

[277] J. Gao, H. Wang, and H. Shen, "Task failure prediction in cloud data centers using deep learning," *IEEE Trans. Services Comput.*, vol. 15, no. 3, pp. 1411–1422, May 2022.

[278] S. Mohseni, M. Pitale, J. Yadawa, and Z. Wang, "Self-supervised learning for generalizable out-of-distribution detection," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 4, 2020, pp. 5216–5223.

[279] J. Sun et al., "Gradient-based novelty detection boosted by self-supervised binary classification," in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, no. 8, Jun. 2022, pp. 8370–8377.

[280] A. Daneshpazhouh and A. Sami, "Entropy-based outlier detection using semi-supervised approach with few positive examples," *Pattern Recognit. Lett.*, vol. 49, pp. 77–84, Nov. 2014.

[281] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognit.*, vol. 58, pp. 121–134, Oct. 2016.

[282] C. Zhang et al., "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 1409–1416.

[283] W. Chen, L. Tian, B. Chen, L. Dai, Z. Duan, and M. Zhou, "Deep variational graph convolutional recurrent network for multivariate time series anomaly detection," in *Proc. Int. Conf. Mach. Learn.*, vol. 162, 2022, pp. 3621–3633.

[284] T. Wen and R. Keyes, "Time series anomaly detection using convolutional neural networks and transfer learning," 2019, *arXiv:1905.13628*.

[285] G. Michau and O. Fink, "Unsupervised transfer learning for anomaly detection: Application to complementary operating condition transfer," *Knowl.-Based Syst.*, vol. 216, Mar. 2021, Art. no. 106816.

[286] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 1, 2019, pp. 9428–9433.

[287] Y. Zuo, Y. Wu, G. Min, C. Huang, and K. Pei, "An intelligent anomaly detection scheme for micro-services architectures with temporal and spatial data analysis," *IEEE Trans. Cognit. Commun. Netw.*, vol. 6, no. 2, pp. 548–561, Jun. 2020.

[288] L. Shen, Z. Yu, Q. Ma, and J. T. Kwok, "Time series anomaly detection with multiresolution ensemble decoding," in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, 2021, pp. 9567–9575.

[289] H. Xu et al., "Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications," in *Proc. World Wide Web Conf.*, 2018, pp. 187–196.

[290] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2019, pp. 2828–2837.

[291] J. Sipple, "Interpretable, multidimensional, multimodal anomaly detection with negative sampling for detection of device failure," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9016–9025.

[292] *Prometheus Monitoring System & Time Series Database*. [Online]. Available: https://prometheus.io/

[293] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A case for cloud storage diversity," in *Proc. 1st ACM Symp. Cloud Comput.*, Jun. 2010, pp. 229–240.

[294] R. Kang, M. Zhu, F. He, and E. Oki, "Implementation of virtual network function allocation with diversity and redundancy in kubernetes," in *Proc. IFIP Netw. Conf. (IFIP Netw.)*, Jun. 2021, pp. 1–2.

[295] M. Uluyol, A. Huang, A. Goel, M. Chowdhury, and H. V. Madhyastha, "$Near - Optimal$ latency versus cost tradeoffs in $Geo - Distributed$ storage," in *Proc. 17th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2020, pp. 157–180.

[296] Y. Aldwyan and R. O. Sinnott, "Latency-aware failover strategies for containerized Web applications in distributed clouds," *Future Gener.*

Comput. Syst., vol. 101, pp. 1081–1095, Dec. 2019.

[297] H. Jin, G. Yang, B.-y. Yu, and C. Yoo, "FAVE: Bandwidth-aware failover in virtualized SDN for clouds," in *Proc. IEEE 12th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2019, pp. 505–507.

[298] R. Landa, L. Saino, L. Buytenhek, and J. T. Araújo, "Staying alive: Connection path reselection at the edge," in *Proc. 18th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2021, pp. 233–251.

[299] I. Giannakopoulos, I. Konstantinou, D. Tsoumakos, and N. Koziris, "Cloud application deployment with transient failure recovery," *J. Cloud Comput.*, vol. 7, no. 1, pp. 1–20, Dec. 2018.

ABOUT THE AUTHORS

Shuiguang Deng (Senior Member, IEEE) received the B.S. and Ph.D. degrees in computer science from the College of Computer Science and Technology, Zhejiang University, Hangzhou, China, in 2002 and 2007, respectively.

He was a Visiting Scholar with the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2014, and Stanford University, Stanford, CA, USA, in 2015. He is currently a Full Professor with the College of Computer Science and Technology, Zhejiang University. He has published more than 100 papers in journals and refereed conferences. His research interests include edge computing, service computing, cloud computing, and business process management.

Dr. Deng is a Fellow of the Institution of Engineering and Technology (IET). In 2018, he was granted the Rising Star Award by IEEE Technical Community on Services Computing (TCSVC). He serves as an Associate Editor for IEEE TRANSACTIONS ON SERVICES COMPUTING, *Knowledge and Information Systems*, *Computing*, and *IET Cyber-Physical Systems: Theory & Applications*.

Cheng Zhang received the M.S. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2013, where he is currently working toward the Ph.D. degree in computer science and technology.

His research interests include edge computing and edge intelligence.

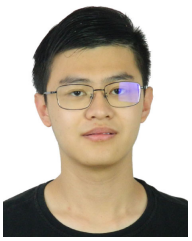
Feiyi Chen received the B.S. degree from the School of Computer Science and Engineering, Sun Yat-sen University (SYSU), Guangzhou, China, in 2021. She is currently working toward the master's degree at the College of Computer Science and Technology, Zhejiang University, Hangzhou, China.

Her research interests include cloud computing, edge computing, and distributed systems.

Hailiang Zhao received the B.S. degree from the School of Computer Science and Technology, Wuhan University of Technology, Wuhan, China, in 2019. He is currently working toward the Ph.D. degree at the College of Computer Science and Technology, Zhejiang University, Hangzhou, China.

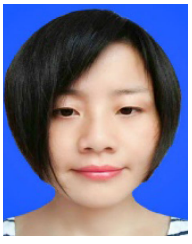
He has published several papers in flagship conferences and journals, including IEEE International Conference on Web Services (ICWS) 2019, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (TPDS), and IEEE TRANSACTIONS ON MOBILE COMPUTING (TMC). His research interests include cloud and edge computing, distributed computing, and optimization algorithms.

Mr. Zhao was a recipient of the Best Student Paper Award of IEEE ICWS 2019. He is a reviewer for IEEE TRANSACTIONS ON SERVICES COMPUTING (TSC) and IEEE INTERNET OF THINGS JOURNAL.



Yinuo Deng received the B.S. degree from the School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing, China, in 2022. He is currently working toward the M.S. degree in computer science of technology at the College of Computer Science and Technology, Zhejiang University, Hangzhou, China.

His research interests include cloud computing, networking, and distributed systems.



Binbin Huang received the Ph.D. degree in computer science and technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2014.

She is currently an Assistant Professor with the College of Computer Science, Hangzhou Dianzi University, Hangzhou, China. Her research interests include cloud computing, mobile edge computing, and reinforcement learning.

Jianwei Yin received the Ph.D. degree in computer science from Zhejiang University (ZJU), Hangzhou, China, in 2001.

He was a Visiting Scholar with the Georgia Institute of Technology, Atlanta, GA, USA. He is currently a Full Professor with the College of Computer Science, ZJU. He has published more than 100 papers in top international journals and conferences. His current research interests include service computing and business process management.

Dr. Yin is an Associate Editor of IEEE TRANSACTIONS ON SERVICES COMPUTING.

Schahram Dustdar (Fellow, IEEE) is currently a Full Professor of computer science (informatics) with a focus on Internet technologies heading the Distributed Systems Group at the Technische Universität Wien (TU Wien), Vienna, Austria.

Dr. Dustdar is an elected member of the Academia Europaea: The Academy of Europe, where he has been the Chairperson of the Informatics Section for multiple years. He has been an Asia-Pacific Artificial Intelligence Association (AAIA) President since 2021 and a Fellow since 2021. He has been an EAI Fellow since 2021 and an I2CICC Fellow since 2021. He was a member of the IEEE Computer Society Fellow Evaluating Committee from 2022 to 2023. He was a recipient of multiple awards, including the TCI Distinguished Service Award in 2021, the IEEE Technical Community on Services Computing (TCSVC) Outstanding Leadership Award in 2018, the IEEE TCSC Award for Excellence in Scalable Computing in 2019, the ACM Distinguished Scientist in 2009, the ACM Distinguished Speaker in 2021, and the IBM Faculty Award in 2012. He was the Founding Co-Editor-in-Chief of *ACM Transactions on Internet of Things* (ACM TIoT). He is the Editor-in-Chief of *Computing* (Springer). He is an Associate Editor of IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON CLOUD COMPUTING, *ACM Computing Surveys*, *ACM Transactions on the Web*, and *ACM Transactions on Internet Technology* and on the Editorial Board of IEEE INTERNET COMPUTING and the IEEE Computer Society.

Albert Y. Zomaya (Fellow, IEEE) is currently the Peter Nicol Russell Chair Professor of computer science and the Director of the Centre for Distributed and High-Performance Computing, The University of Sydney, Sydney, NSW, Australia. He has published more than 600 scientific papers and articles and is a (co)author/editor of more than 30 books. As a sought-after speaker, he has delivered more than 250 keynote addresses, invited seminars, and media briefings. His research interests span several areas in parallel and distributed computing and complex systems.

Dr. Zomaya is an elected fellow of the Royal Society of New South Wales and an Elected Foreign Member of Academia Europaea. He is a decorated scholar with numerous accolades, including the Fellowship of the American Association for the Advancement of Science and the Institution of Engineering and Technology, U.K. He was a recipient of the 1997 Edgeworth David Medal from the Royal Society of New South Wales for outstanding contributions to Australian Science, the IEEE Technical Committee on Parallel Processing Outstanding Service Award in 2011, the IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing in 2011, the IEEE Computer Society Technical Achievement Award in 2014, the ACM MSWIM Reginald A. Fessenden Award in 2017, the New South Wales Premier's Prize of Excellence in Engineering and Information and Communications Technology in 2019, and the Research Innovation Award from the IEEE Technical Committee on Cloud Computing in 2021. He is the Editor-in-Chief of *ACM Computing Surveys* and was the Editor-in-Chief of IEEE TRANSACTIONS ON COMPUTERS from 2010 to 2014 and IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING from 2016 to 2020.