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

SUBMISSION DATE / POSTED DATE

12-06-2023 / 14-06-2023

CITATION

Deng, Shuiguang; Zhao, Hailiang; Huang, Binbin; Zhang, Cheng; Chen, Feiyi; Deng, Yinuo; et al. (2023). Cloud-Native Computing: A Survey from the Perspective of Services. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.23500383.v1

DOI

10.36227/techrxiv.23500383.v1

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

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 paper surveys key issues during the life-cycle of cloud-native applications, from the perspective of services. Specifically, we elaborate the research domains by decoupling the life-cycle of cloud-native applications into four states: building, orchestration, operate, 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.**

***Index Terms*—Cloud-native applications, survey, service life-cycle management, research roadmap, microservice.**

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]. Service-Oriented Architecture (SOA) is the principle to design the software systems by (i) provisioning independent, reusable, and automated functions as reusable services and (ii) providing a robust and secure foundation for leveraging these services [2]. In recent years, the most influential variant of SOA is *the microservices 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 microservices approach is more and more appealing, as it allows teams and software organizations to be more productive to build continuously deployed systems with the support of DevOps [4] and continuous integration/continuous delivery (CI/CD) pipelines [5].

Accompanied with the development of microservices, a new terminology, *cloud-native*, or *cloud-native computing*, is attracting increasingly more attention in academia. In accordance with Cloud Native Computing Foundation (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 microservices architecture, cloud-native is also characterized by the following terminologies:

- *Containerization*. Containerization is a function isolation mechanism which leverages the Linux kernel to isolate resources, and 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]. Combing containerization with the microservices architecture, each part of an application, including processes, libraries, etc., is packaged into its own container. This facilitates reproducibility, transparency, and resource isolation.
- *Orchestration*. Orchestration is the automated configuration, management, and coordination of the inter-related microservices to build the elastic and scalable functionalities. Because microservices are deployed in the way of containers, orchestration reduces to the automation of the operational effort to manage the containers' life-cycle, including resource provisioning, deployment, scheduling, scaling (up and down), networking, load balancing, etc, in order to execute the applications' workflows or processes. Kubernetes [9], originated from Google's Borg cluster manager [10], is the most popular open-source container orchestration software.

As a conclusion, a cloud-native application can be viewed as a distributed, elastic and horizontal scalable system composed of inter-related microservices, which isolates state in a minimum of stateful components [11]. Applications are built with cloud-native technologies by the following steps: (i) 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); (ii) Using lightweight operating system virtualization technology, i.e., containerization, to pack each microservice into a container; (iii) Orchestrating these containers into a organic whole for functionalities with automatic configuration

S. Deng, H. Zhao, C. Zhang, F. Chen, Y. Deng, and J. Yin are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China. E-mail: dengsg, hliangzhao, coolzc, chenfeiyi, yinuo, zjuyjw@zju.edu.cn

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

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

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

[1]https://www.cncf.io/about/who-we-are/

[2]https://github.com/cncf/toc/blob/main/DEFINITION.md

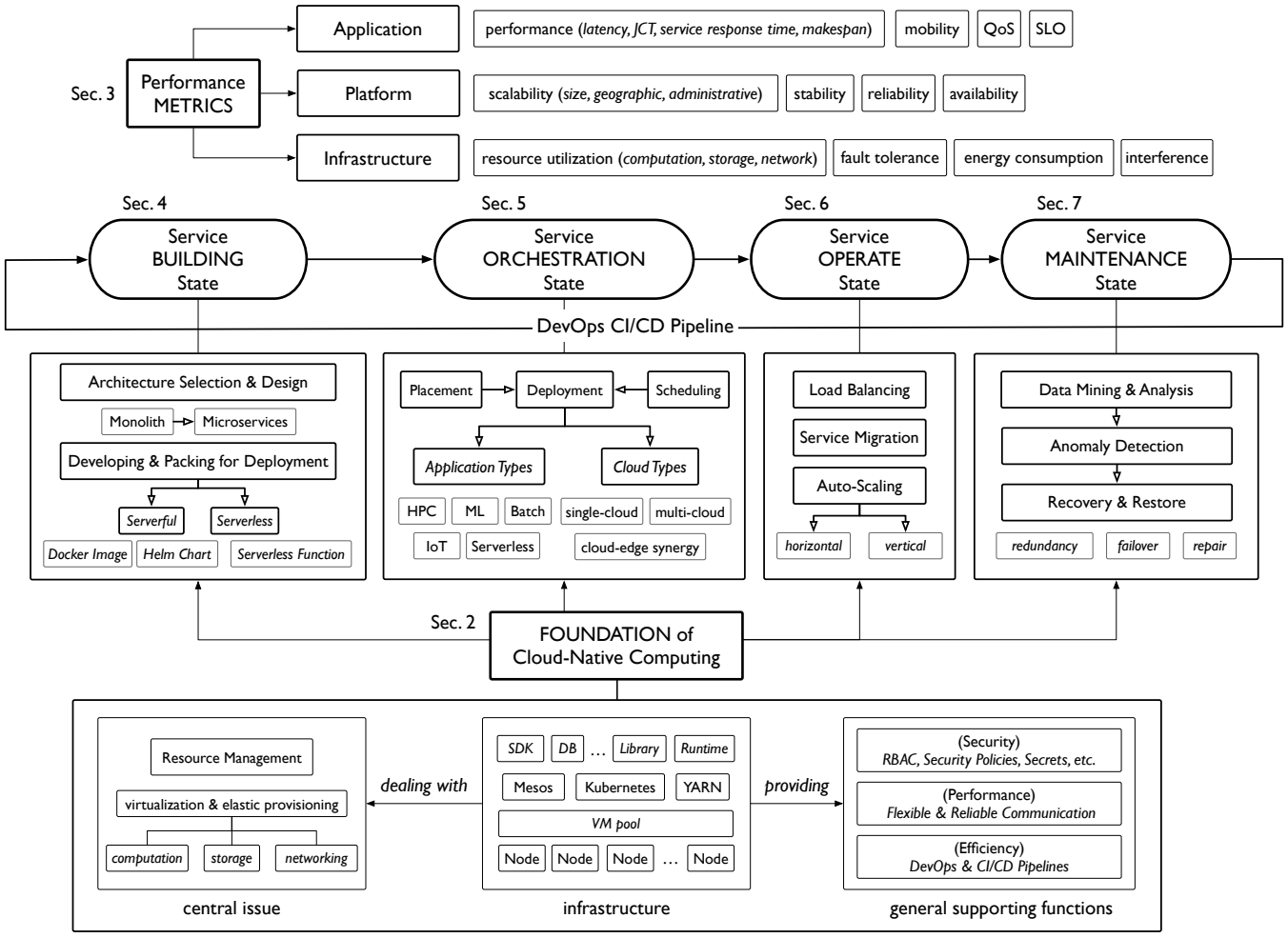

Fig. 1. The research roadmap for cloud-native computing, designed from the perspective of services.

and management throughout their life-cycle; (iv) Using DevOps and CI/CDs to deliver the cloud-native applications with reliability and scalability.

Considering that cloud-native is better known in industry, in this paper, we try to survey the past and present of cloud-native applications w.r.t. 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 researches, either theoretical or systematic, 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 service-oriented computing (SOC), into four states: *building*, *orchestration*, *operate*, and *maintenance*. As Fig. 1 shows, different key problems are emphasized in different states. In addition, we also collect the performance metrics which are frequently mentioned when building the cloud-native applications, and analyse them from three levels: infrastructure, platform, and software. Apart from the service life-cycle and the 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 resource management, i.e., resource virtualization and elastic 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. The following sections 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]. However, they mainly provide high-level opinions and ignore the review on 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, the author of [14] discusses the recent developments of architectural frameworks for intelligent and autonomous management for cloud-native networks. This paper comprehensively reviewed the technical trend toward cloud-native network design and network-cloud/edge convergence. Moreover, the scheduling of

TABLE I
SUMMARY OF IMPORTANT ACRONYMS.

ACRONYM	DEFINITION	ACRONYM	DEFINITION
SOC	Services-Oriented Computing	**SOA**	Service-Oriented Architecture
DevOps	Combination of development and operations	**CI/CD**	continuous integration & continuous delivery
CNCF	Cloud Native Computing Foundation	**API**	application programming interfere
SDN	software-defined network	**NFV**	network funciton virtualization
CNF	cloud-native network function	**VXLAN**	Virtual eXtensible Local Area Network
GRE	Generic Routing Encapsulation	**MPLS**	Multiprotocol Label Switching
LVM	logical volume manager	**RAID**	redundant array of independent disks
CNI	container network interface	**CSI**	container storage interface
SDK	software development kit	**TLS**	transport layer security
NAT	network address translation	**SaaS**	Software as a Service
PaaS	Platform as a Service	**IaaS**	Infrastructure as a Service
JCT	job completion time	**CRUD**	create, read, update and deletion
REST	Representational State Transfer	**HPC**	high-performance computing
GA	Genetic algorithm	**QoE**	Quality of Experience
QoS	Quality of Service	**SLO**	Service-Level Objective
ANN	Artificial Neural Network	**MARL**	Multi-Agent reinforcement learning
VM	Virtual Machine	**ISP**	Internet Service Provider

microservices and containers, which has a strong connection with the orchestration platform Kubernetes, is reviewed in [15] and [16]. However, in 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 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: Sec. II-VII introduce the foundation, performance metrics, and the four states of demonstrated in this roadmap. Sec. VIII discusses the issues, challenges, limitations, and opportunities of cloud-native. We conclude this paper in Sec. IX. We summarize the definitions of the acronyms that will be frequently used in this paper in Table I for ease of reference.

II. FOUNDATION OF CLOUD-NATIVE COMPUTING

In this section, we demonstrate the fundamental necessities in cloud-native. From the perspective of clusters, we demonstrate the central issue of the foundation and general function modules which play critical roles when building cloud-native applications.

A. The Hierarchical Structure of Infrastructure

A cluster is a set of computing nodes which are connected to each other through fast local area networks. On top of the physical nodes, OS-level virtualization techniques are utilized to create virtual machines (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 softwares. Cluster management is always tightly coupled with resource management and task scheduling. Kubernetes, as we have mentioned before, is widely adopted across industries and has become a *de facto* standard. Kubernetes has the ability of managing 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 Kubernetes, Mesos [17], Docker Swarm [18], and Hadoop YARN [19] are also widely used cluster managers.

- *Kubernetes*. Kubernetes, abbreviated as K8s, is the most influential open-source platform in cloud-native 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 centre 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 of 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 [20]. Istio extends K8s to establish a programmable, application-aware network by taking Envoy[7] as the proxy. The monitoring of nodes and applications can be realized with Prometheus[8] and

[3]https://kubernetes.io/docs/concepts/architecture/nodes/
[4]The architecture of Kubernetes 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/
[8]https://prometheus.io

Grafana[9] while logging can be managed with Kibana[10].

- *Mesos*. Mesos is implemented in a two-level architecture. The first level is a *master resource manager* that dynamically controls which resources each *framework scheduler* owns. Correspondingly, in the second level, each *framework scheduler*, such as hadoop, MPI, and Mesos Marathon, are responsible for scheduling tasks at the application level [17].
- *YARN*. YARN manages Hadoop clusters. It consists of clients, container, resource manager (RM), node manager (NM), and application master (AM). Here container is defined as a collection of physical resources such as RAM, CPU cores and disk on a single node. It is NM 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 AM negotiates resources with RM and requests Container from the NM [19].
- *Docker Swarm*. Docker Swarm is the native inbuilt orchestration tool for Docker, which is called "swarm mode". Docker Swarm is composed of Docker Node, 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 constrast, the Workers are nothing but the instances of Docker Engine for running Docker containers [21].

The following contents are mainly focusing on K8s since in cloud-native, many rigor progress 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* [22]. To take the full advantage of this hierarchy, KueEdge divides its components into two parts: CloudCore and EdgeCore. CloudCore (i) handles the communication between it and *api-server* of a K8s cluster and (ii) communicates with the edge nodes. EdgeCore is responsible for (i) communicating with CloudCore and (ii) manages the containers, 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* [23].

On top of cluster managers, development tools including SDKs and middlewares are developed as the building blocks for cloud-native applications. The hierarchical structure of hardwares, OS-level softwares, cluster managers, and middlewares construct the fundamental necessities for building cloud-native applications.

[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/

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 II.

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 and deployment. 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, as well as 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 virtual machines 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, a.k.a. "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 [24]. 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 solution does operate on an existing operating system, making it Type 2 by definition.

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 [25]. 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 Kubernetes, are based on Docker or other container engines [26], [27].

TABLE II
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

This is mainly due to several unique advantages containers possess: simplicity, low overhead, fast deployment, and ease to design and build.

2) Network Virtualization: Network virtualization is another important level of virtualization in cloud-native scenarios. 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 software-defined network (SDN), network function virtualization (NFV), Service Mesh, and overlay networks, to understand how they enable efficient and intelligent network management.

Software-defined Network (SDN) includes a set of techniques to decouple the data plane and control plane to enable programmatical a dynamical management of network forwarding devices like 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 [28].

Network Function Virtualization (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 [29] and router [30]. Furthermore, Cloud-native Network Function (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 like 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 is an inherent requirement 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 Logical Volume Manager (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 Redundant Array of Independent Disks (RAID). There has 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. 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 live 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, Container Storage Interface (CSI)[13] is proposed and introduced since Kubernetes v1.9. The CSI is a standard of exposing arbitrary low level storage system to containerized applications orchestrated by Cloud Orchestration systems like Kubernetes. 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 today's customized cloud-native deployments.

C. General Supporting Function Modules

In this section, we demonstrate the general function modules provided by the K8s ecosystem that play 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.

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

- *Role-Based Access Control (RBAC)*. RBAC[14] 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's possible to restrict certain privileges to only authorized entities.
- *Pod Security Policies*. K8s provides Pod Security Policies that restrict the behavior of containers running inside the pods. These policies 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.
- *Secrets Management*. K8s offers a facility, named as secret[15], for securely storing and managing sensitive data like passwords, certificates, keys, etc. The secret data is encrypted at rest and in transit, and the 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, etc.) 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. Additionally, 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 (Container Network Interface) APIs. Typical implementations include Flannel[16], Calico[17], Weave[18], etc. 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 [31]. Different implementations of CNI are compared in [32], [33] and [34]. K8s also has an inbuilt DNS such that *Pods* and *Services* can be discovered and visited though their domain names.

Note that the implementation of CNI is mandatory for building a working K8s cluster. However, to have a reliable, observable, and secure communication, the vanilla network is far from enough. Under the circumstances, 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 [20]. Fig. 2 illustrates why Service mesh is called a mesh. Service mesh provides dynamic discovery of services, intelligent load balancing across services, security features with encryption and authentication, and observability tracing by leveraging a

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

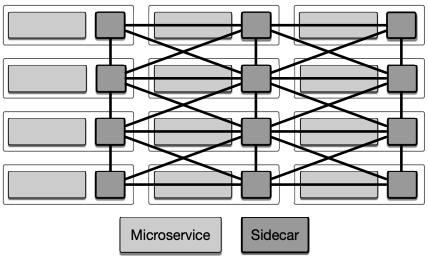

Fig. 2. The sidecar proxies along with each microservice form a mesh-like network.

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, 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 continuous delivery of new software versions, while guaranteeing their correctness and reliability [35]. With DevOps, development and operations teams work together across the entire cloud-native applications' lifecycle. DevOps is strongly connected with CI/CD, which is a multi-stage pipeline of continuous integration (*build* $\rightarrow$ *test* $\rightarrow$ *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.

In the K8s ecosystem, KubeSphere DevOps is a powerful CI/CD platform that has attracted many attentions from industry [36]. KubeSphere DevOps provides CI/CD pipelines based on Jenkins[19]. 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 provides different type services from different perspective. First, at application layer, various software are deployed as different service that can be accessed by customers over network. The performance metrics related to the application contextual are latency, response time, completion time, makespan and so on. Then, 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, infrastructure layer provides the raw computing, storage, and network resources

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

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 single performance metric or jointly considering multi-criteria. We describe these performance metrics in detail as follows.

A. Performance Metrics in Application Level

Various applications are encapsulated to be different containers and provides services to customers by deploying these containers to physical hosts in a cloud environment. Benefited from the convenience without installing and running the application programs. In application layer, the key problem is to guarantee the quality of service (QoS) for applications which can be measured by multiple performance metrics such as latency, response time, completion time, makespan. These performance metrics are very similar, and are often easily confused. But in fact, they have different meanings. In this paper, we distinguish these performance indicators and review these related work optimizing these indicators.

- *Latency*: it is usually referred to the time during between when something happens and when it is perceived. Some efforts presented in [37], [38] take latency as an important metric. For example, the authors in [38] extend Kubernetes mechanisms to support multi-tenant at the cost of introducing the moderate latency and throughput overheads.
- *Response Time*: it is referred to the time during 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 to optimize response time. For example, the authors presented in [39] extend Kubernetes mechanism to schedule pod according to dynamic network metrics, the goal of which is to reduce the application response time.
- *Job Completion Time (JCT)*: it is the time during from the start time of the entry task in job to the finish time of the last task. Different from response time, job completion time mainly concentrates on its processing time. Some efforts presented in [40]–[43] explicitly consider the job completion time.
- *Throughput*: it is the ratio of processed requests to the total number of arrived requests at the system. A higher throughput indicates that much more requests are processed in unit time and much less response time are 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 [44], [45].
- *Mobility*: the mobility of terminals also bring great challenges to service performance assurance. Specially, the service provision problem in the driver-less scenario has attracted extensive attention from academic and industry. Many works explore a series problems of service provision in terms of service placing, service scheduling, service migration, aiming at guaranteeing the service performance in the case of terminal moving. For instances, the authors presented in [37] explicitly investigate the influence of the terminal mobility on service performance, and design some strategies to guarantee the service performance.
- *SLO*: Service Level Objective (SLO) is a key performance indicator that defines the level of service. It typically defined 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 [46], [47] tries to guarantee SLO for an application.

B. Performance Metrics in Platform Level

Platform as a service provides all facilities and APIs to build and deliver various services conveniently, which efficiently avoid 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 perspective. We describe these metrics and review these related studies optimizing these indicators in detail.

- *Scalability*: it is the ability of a system enabling to dynamically adjust the amount of resources allocated to containerized applications according to their potential workload fluctuations, which ensures the applications supported with enough resources to minimize SLA violations. Scaling can be performed vertically, horizontally, or both [48]. At present, some efforts presented in [48]–[52] explicitly consider horizontal autoscaling, vertical autoscaling, and both. For example, in [53], it can create much more container replicas to meet more application resource requirement. In [54], it reallocates the amount of resource to the existing containerized application to best utilize the new hardware resource capacity.
- *Stability*: it is a system's ability to keep a quantity of required properties (e.g., queue length, waiting time, etc) within a bounded region when the system encounter some disturbances. To guarantee the stability of a system is a fundamental issue to ensure service performance. Therefore, some works investigate system stability problem and design various container deployment or placement approaches to guarantee service performance. For example, in [55], an adapted reinforcement learning algorithm is adopted to achieve horizontal and vertical elasticity of cloud application for increasing the flexibility to cope with varying workloads and guarantee performance stability. In [56], a two-step algorithm is designed to solve the container deployment problem in a geo-distributed computing environment. In the first step, a reinforcement learning approach is adopted to dynamically controls 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 geo-distributed computing resources. Its main goal is to satisfy Quality of Service requirements of latency-sensitive applications.

- *Reliability*: it is referred 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 very important performance metrics. 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 virtual machines in [57]. To maintain reliability and elasticity for the system, a dynamic scheduling algorithm is proposed to balance the workload of virtual machines in a cloud environment elastically based on resource provisioning and de-provisioning 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 problem to guarantee system reliability. The author presented in [58] 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.
- *Availability*: it is the proportion of time a system is in a functioning condition. Along with scalability, stability, reliability, availability is also a prevailing issues for platform service. To cope with possible failure caused by the mobility of parked vehicles and improve the service availability, the author presented in [59] design the dual cost and utility-aware heuristic algorithm to solve the problem of multi-replica task scheduling in a collaborative computing paradigm consisting parked vehicles.

C. Performance Metrics in Infrastructure Level

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

- *Resource Utilization*: It is an important performance metric used to describe the percentage of a system available resource, such computation resource, storage resource and network resource, that is occupied over an amount of available time (or capacity). In recent years, some efforts presented in [60]–[64] explore resource planning and resource scheduling problems with the maximization of the CPU and RAM utilization. Specifically, the authors presented in [61] extend the Kubernetes mechanisms to fairly allocate multi-resource (such as CPU, memory, and disk) for containerized workloads of multi tenants. In [65], a storage service orchestration platform is designed and implemented to support the stateful applications. In [66], 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 [67]–[69]. For example, the authors in [69] and [70] design a network-aware scheduler to automatically manage and deploy containerized applications, aiming at for reducing the network latency.
- *Interference*: In cloud native environments, various types of workloads are encapsulated in the form of containers. However, the isolation of container is weaker than that of virtual machine. Multiple containerized workloads (such as computing intensive, storage intensive) co-located on the same server can interfere with each other, which seriously affect system performance. The interference issue incurred by co-locating different type workloads become a pressing issue. Therefore, many works presented in [40], [44], [71]–[73] explicitly consider the inference between co-locate containerized jobs. For instance, the authors propose in [40] a container placement scheme that balances the resource contention on the worker nodes.
- *Energy Consumption*: It is the amount of energy used. The significant amount of energy consumed by data centers can incur high cost 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 [71], [74]–[77]. For example, the authors presented in [74] propose an energy-efficient container migration scheme to migrate containers for reducing the energy consumption. The authors presented in [71] design a scheduler to minimize energy consumption and interference.
- *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 costs for renting infrastructure resource is also very import performance metric. In recent years, some research efforts presented in [78] [79] [80] [81] 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.
- *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 has heavily negative effect to the system. Hence, it is import to design various fault tolerance mechanism to cope with this problem and to minimize the risk of failure. For instances, in [82], the authors develop a new container storage driver to solve the global failures and bundled

performance problem.

IV. Service Building

In service building state, the key steps are (i) architecture selection and (ii) code development & packing.

A. From Monolith to Microservices

Before the rise of the microservices 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:

- *Business Logic*. The application's core business logic. For example, in e-commerce websites, the logic of inventory and shipping management.
- *Database*. The data access objects responsible for the CRUD of data.
- *Interaction and Presentation*. The component responsible for handling HTTP requests and responding with either HTML or JSON/XML (for web services APIs) objects.
- *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, 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 increase the risk for availability since many dependent and tightly coupled components increase the impact of a single failure.

To solve the above problems, the microservices architecture is proposed and it becomes the dominant architectural style choice for service-oriented software [83]. With a microservices 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 demand for specific functions of an application [84]–[86]. The microservices architecture brings in many benefits, such as agility, flexible scaling, easy deployment, resilience, etc. When developing the cloud-native applications, the primary task is to select the 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. The most popular choice for deploying these container images on a container orchestration platform like Kubernetes 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 the required artifacts, including container images, Helm charts, documentation, etc., into an archive.

It worth mentioning that, if the cloud-native application is published through serverless functions, the developer only need to upload the code to the serverless platform, and the containerization and orchestration is automatically executed by the underlying middlewares 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 do not have to reserve and pay for a fixed amount of bandwidth or number of servers, as the service is auto-scaling. 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 Sec. V-A3.

V. Service Orchestration

Service orchestration is the automated configuration, management, and coordination of multiple microservices to deliver the end-to-end services. Since microservices are encapsulated in form of container, service orchestration is essentially container orchestration. As a popular open-source container orchestration tool, Kubernetes 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 [45], [88]. In the cloud native context, these two key technologies are investigated widely by a plenty of studies. 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 III.

- What type of applications are orchestrated in cloud native system?
- What computational architectures are used in the service orchestration?

A. Application Types

Different types application have significantly distinct characteristics, such as their Quality of Service requirements, the type, the structure and so on. These characteristics of the

TABLE III
REPRESENTATIVE WORKS IN SERVICE ORCHESTRATION.

WORK	APP. TYPE	RESOURCE			CLOUD TYPE			METRIC	IMPLEMENTATION
		CPU	Storage	B.W.	Single-Cloud	Multi-Cloud	Cloud-Edge		
[89]	ML	✓	✓	✓			✓	JCT	simulation
[90]		✓	✓	✓			✓	JCT	simulation
[91]		✓	✓	✓	✓			training time	simulation
[92]		✓	✓	✓	✓			JCT & makespan	system
[93]			✓		✓			makespan	system
[94]		✓			✓			JCT	simulation
[95]		✓			✓			latency & res. utilization	simulation
[**?**]		✓			✓			inference time	simulation
[96]		✓	✓		✓			response time	system
[97]	HPC	✓	✓		✓			response time	system
[98]		✓	✓	✓	✓			response time	simulation
[99]		✓	✓		✓			througput	simulation
[100], [101]		✓	✓	✓	✓			response time	system
[41]		✓	✓				✓	makespane	simulation
[102]		✓			✓			prediction accuracy	simulation
[103]	serverless	✓	✓	✓	✓			response time	simulation
[104]		✓	✓		✓			scalability	system
[105]		✓	✓	✓	✓			JCT	simulation
[106]		✓	✓		✓			latency & throughput	simulation
[107]			✓		✓			JCT	simulation
[108]		✓	✓		✓			JCT & res. utilization	system
[109]	batch job	✓	✓		✓			SLO & res. utilization	simulation
[110]		✓	✓	✓	✓			res. utilization	simulation
[111]		✓	✓	✓		✓		JCT	simulation
[112]		✓	✓	✓	✓			res. utilization & cost	simulation
[113]		✓	✓	✓	✓			JCT & res. utilization	simulation

applications have a significant impact on the service orchestration. A wide range of applications from High-Performance Computing (HPC), machine learning, batch, web service or serverless, are handled in 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.

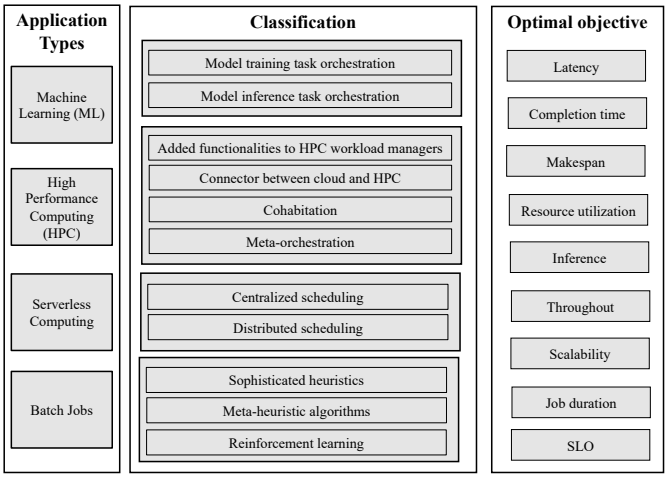

Fig. 3. Service orchestration under different application types.

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 consist of model training and model inference two stages. These two stage tasks are characterized by unique and complicated features. Specifically, the model training task is long-lived offline task, while the model inference task is short-lived online task. Moreover, these two stage tasks focus different performance metrics, in which the model training task focus on achieving high performance and the model inference task pay more attention on the response latency and inference accuracy. Their unique characteristics and different performance requirements impose some specific challenges to orchestrate model training tasks and model inference tasks. We comprehensively review and summarize these studies related to model training task orchestration and model inference task, respectively.

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 model and larger datasets, model training is an extremely time consuming and resource consuming task. Thus, it is urgent to parallelize model training task on multiple distributed workers. There are mainly four types of parallelism model training: data parallelism [89], [114]–[117], model parallelism [90], pipeline parallelism [91], [118]–[120] and mixed parallelism [92], [121]. Data parallelism refers to place multiple replicas of a model on multiple workers, and divide the datasets into many subsets to feed to these multiple workers. These multipe workers simultaneously perform the model training tasks, and synchronize their training results in the form of parameter servers or All-Reduce, etc. By data parallelism, the speed for model training can be greatly accelerated and the performance for model training can be enhanced. For example, in [89], a novel online preemptive scheduling framework is

designed to dispatch machine learning jobs to workers and parameter server for reducing the average job completion time. Analogously, in [93], the authors present a heterogeneity-aware scheduler which can efficiently collocate deep learning jobs on GPUs by exploiting the predicting information of GPU memory demand and job completion times. Its main goal is to improve the GPU resource utilization and reducing the makespan. However, with the increase of model complexity, a model with the large number of parameters cannot be launched on a single worker. Thus, model parallelism is proposed. Model parallelism refer to divide a model into multiple disjoint partitions and place these partitions on multiple workers. Since each worker only have 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 divides a model into multiple stages and place multiple stages on multiple workers; Beside, pipeline parallelism further divide the datasets into multiple micro-batches. Multiple workers can process multiple micro-batches simultaneously. Thus, pipeline parallelism training greatly reduce the training latency. For example, to achieve an efficient and task-independent model parallelism, the authors in [91] introduce a pipeline parallelism library to partition a deep neural network across multiple workers and split the datasets into mini-batches. Finally, mixed parallelism with data and model parallelism is proposed to reduce the training latency and resource consumption. For example, in [92], the authors design a job scheduling system which enable machine learning jobs to be implemented with data parallelism and model parallelism in clusters. The proposed system greatly reduces the job completion time and improves accuracy.

Model inference task orchestration. Model inference is the service ability to make predictions on new data based a trained model. Model inference tasks are usually deployed as online short-lived services (e.g., automatic driving, 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 [122] and MXNet Model Server [123], have implemented the orchestration function for model inference task. For academia, there are also a lot of studies about model inference task orchestration. These studies mainly can be classified two types: the individual model inference task orchestration and multiple model inference task orchestration. For the single model inference task orchestration, the authors in [124], [125] design some optimization techniques to efficiently orchestrate model inference task. However, the execution of a single model inference task not only fail to meet the requirement of application scenarios, but also causes the waste of the resources. Thus, many researchers further study multiple model inference task orchestration. They design different heuristic-based, modeling-based or prediction-based mechanisms to orchestrate multiple model inference tasks [94], [126], [127]. Specifically, the authors in [95] adopt a heuristic approach to select the requests to be co-located 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, the authors in [94] 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 smallest completion time, aiming at reducing the total delay. However, all of the above studies mainly exploit the heuristic method to orchestrate the 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 can not be applied to cope with dynamic colocation mechanisms for managing the inference workloads. With complexity and dynamics of model inference task, many researchers turn to learning-based methods such as multi-armed bandit, reinforcement learning, etc. For instance, in [44], the authors investigate the job completion time slowdown problem caused by the interference between co-located deep learning jobs. To address this problem, an interference-aware resource manager is designed to effectively co-locate heterogenous deep learning jobs for improving resource utilization and job throughput.

2) High Performance Computing: 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 tend to be required. HPC jobs are often submitted to an HPC cluster and wait to be scheduled by a HPC job scheduler. However, the existing HPC job schedulers lack micro-service support and container management capacities. Therefore, it is a challenge about how to efficiently support HPC workloads on Kubernetes. In recent years, there exist research efforts on efficiently orchestrating HPC jobs on cloud clusters [128]–[131]. These state-of-the-art studies on HPC job orchestration can be divided to four categories: added functionalities to HPC workload managers [132]–[134], connector between cloud and HPC [96], [135]–[137], cohabitation [99], [138], [139], meta-orchestration [140]–[142]. For added functionalities to HPC workload managers, it mainly extends HPC workload manager to support container orchestrator for HPC application. For example, in [96], [97], the authors investigate the problem of running HPC workloads efficiently on Kubernetes 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 shortcoming for added functionalities to HPC workload managers, the connector between cloud and HPC is proposed. The connector enable to bridge the gap between HPC and cloud systems and achieve HPC job orchestration on cloud platform. For example, in [98], a scheduler plug-in for Kubernetes is implemented to efficiently schedule the HPC applications on cloud platforms managed by Kubernetes. In [143], a workflow management system is implemented to schedule the containerized HPC applications such as nextflow, which greatly improve the numerical instabil-

ity incurred by variations across computational platforms. The benefit of the connector is non-intrusive and enable to exploit orchestration strategies of orchestration platforms. However, its disadvantage is that the network latency between cloud and HPC is high. Therefore, a HPC job orchestration approach, called cohabitaion, is proposed. The cohabitaion is to coexist HPC workloads manager and cloud orchestrators on an HPC cluster. For instances, in [99], the authors investigate the service orchestration problem towards HPC workloads. To address this problem, the authors modify the configuration and setup of Kubernetes to support HPC workloads, and evaluate the performance of HPC workloads. The cohabitaion has an advantage of fully exploiting the functionalities of orchestration platforms. However, the cohabitaion is extremely intrusive. Therefore, a HPC job orchestration approach, called meta-orchestration, is designed. The meta-orchestration approach is to implement an additional orchestrator on top of the cloud orchestrator and HPC workload manager. For example, in [144], a framework called Kube-batch is designed to enable HPC workloads execution on Kubernetes. In [100], an open source tool is designed to manage the full life cycle of HPC workloads in cloud architectures. In [101], a framework which is compatibility with Prometheus is proposed to automatically deploy the benchmarking workload for containerized HPC applications and analyze their performances. The advantage of the meta-orchestration approach is less intrusive. However, its disadvantage is to increase the complexity of the architecture and the efforts of maintenance.

3) Serverless Computing: Severless computing is a new execution model of cloud computing which is a integration of both function as a service (FaaS) and backend as a service (BaaS). Serverless computing is characterized by the automatic management and lightweight features. These characteristics of serverless computing enable developers to focus on the business logic, with no need worry about infrastructure provisioning and management. Benefiting from its advantages, serverless computing recently attracts a lot of attentions in both industry and academia. However, the inextricable dependencies between massive functions pose a great challenge to serverless orchestration. In recent years, there are a plenty of research efforts on serverless orchestration. In the industry community, several open-source platforms and serverless computing frameworks, such as Kubeless [145], OpenFaas [146], OpenWhisk [147] or Fission [148], are designed to support the serverless computing orchestration. These open source frameworks with different architectures enable to dynamically manage, scale, and provide different types of resources for serverless applicaitons. In the academic community, some research efforts presented in [102], [103] design diverse scheduling scheme for serverless applications. These strategies can roughly be divided two categories: centralized scheduling [104]–[106] and distributed scheduling [107], [149]. For the centralized scheduler, the authors in [102] present a double exponential smoothing approach to calculate the optimal number of pods for serverless applications. Analogously, in [103], a serverless computing frameworks, called Pigeon, is presented to schedule the FaaS funcation to pre-warmed containers. Moreover, the framework introduces a static pre-warmed container pool to cope with the burst function arrival. Both of novelty mechanism can greatly reducing the response time for serverless application and improve the system performance. Moreover, in [41], the authors investigate the influence of the composite property of services on scheduling scheme at 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 above these approaches are centralized. The centralized schedulers are vulnerable to a single point of failure and high communication overhead. To address these problem of the centralized scheduler, some distributed scheduling strategies are proposed. For example, the authors in [107] design a scheduler based on deep reinforcement learning to dynamically make decision on the number of functions and their resources, aiming at making a trade-off 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 optimization two aspects. For the works on system implementation, the authors in [108], design a cloud-native platform called Fluid which can co-orchestrate the data cache and deep learning jobs to improve the overall performance of multiple deep learning jobs. Analogously, in [109], 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 Kubernetes. For the works on optimization algorithm, there are mainly three kinds of methods to solve it: sophisticated heuristics, meta-heuristic algorithms [110]–[113] and reinforcement learning [150], [151]. The sophisticated heuristics, such fair scheduling [152], first-fit [153], simple packing strategies [154] are usually easy to understand and implement. However, it needs manual adjustment to gradually improve the algorithm. Therefore,the meta-heuristic algorithms, such genetic algorithm or ant colony algorithm, are proposed to orchestrate batch jobs. For example, in [110], the authors adopt the meta-heuristics optimization algorithm to schedule batch jobs for achieve higher resource utilization. In [111], 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 about how many instances as well as on which edge sites to place them for each microservice. Its main goal is to reduce service execution latency and improve the service availability. Analogously, in [112], a stochastic hybrid workflow scheduling algorithm is design to jointly schedule offline batch workflows and online stream workflows in cloud container services. Its main goal is to minimize the cost and improving resource utilization in cloud container services. Moreover, in [113], the authors 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 meta-heuristic algorithms can not 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 instances, in [150], the authors adopt a deep reinforcement learning algorithm to schedule independent batch jobs among multiple clusters adaptively. Analogously, the authors in [151] propose a graph learning approach to discover the insightful properties and patterns of batch jobs. Based on these characteristics, the batch jobs can be better scheduled in production cloud computing environment.

B. Cloud Types

Existing mainstream computing paradigms include single-cloud, multi-cloud and cloud-edge synergy. Different computing paradigms have different characteristics, which have an important impact on service orchestration. Plenty of research studies have investigated the service orchestration under three different computing paradigms. We categorize them by the mainstream computing paradigms and overview them. Fig. 4 outlines the structure of this section.

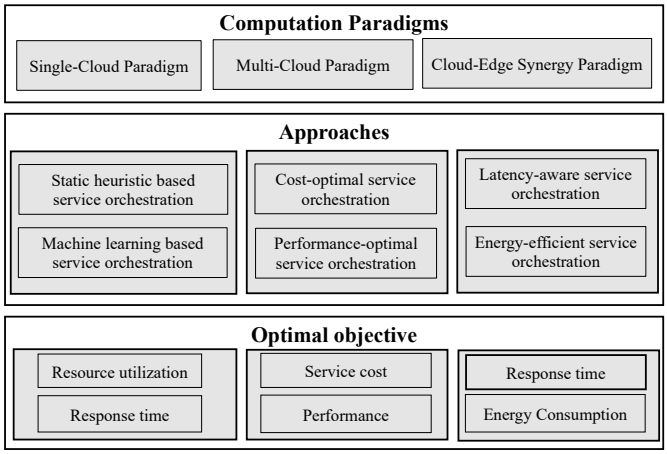

Fig. 4. Service orchestration under different computation paradigms.

1) Single-Cloud: The single-cloud paradigm is a new service model which delivers complex hardware and software services to external customers through the Internet [110]. In order to improve the utilization of cloud resource and reduce the response time of cloud service, the efficient cloud service orchestration is key. Currently, a large number of approaches are proposed to handle service orchestration [62], [63], [72], [78], [79], [155]–[159], [159]. These approaches can be classified three types: static heuristic based service orchestration [155], [156], machine learning based service orchestration [63], [72], [157]. For static heuristic based service orchestration, various heuristic algorithms including bin-packing algorithms, genetic algorithm, particle swarm optimization, etc., are adopted to orchestrate service in certain workload scenarios at a limited scale. For example, in [155], the authors 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 virtual machines while meeting deadline constraints. Analogously, in [156], the authors 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, recurrent Neural Network, etc., are accordingly employed to orchestrate service. For example, in [72], a containerized task scheduler employs K-means++ algorithms to characterize the workload features and identify their behavior, and accurately co-locate heterogeneous workloads in an interference-aware manner. This scheme greatly improves resource utilization and reduces the rescheduling rate. Moreover, in [63], a Kubernetes scheduler extension which adopt machine learning algorithm to predict Quality of Experience (QoE) and schedule the resource based on the predicted QoE, is designed to improve the average QoE and eliminate over-provisioning altogether in the cloud. Also, in [157], a self-adaptive Kubernetes 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 model for diverse and dynamic workloads and environments and predict multi-dimensional performance metrics. These schemes could further improve the quality of resource provisioning decisions in response to the changing workloads under complex environments.

2) Multi-Cloud: With the surge of cloud workloads, single-cloud paradigm cannot meet with their various requirements, such as resource requirement, cost requirement, reliability requirement and so on. Therefore, multi-cloud paradigm is proposed. The multi-cloud paradigm enables resources among different clouds to be shared to cope with a burst of incoming tasks. In addition, the multi-cloud paradigm can efficiently improve service reliability and reduce service cost. Although benefiting from these advantages of the multi-cloud paradigm, the heterogeneity of the underlying resources and services for different cloud systems brings some new challenges to service orchestration in multi-cloud paradigm. To cope with these new challenges, some related research works about service orchestration in multi-cloud paradigm [160]–[165] 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, the authors in [160]–[162] 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, the authors in [163], adopt a meta-heuristic algorithm to continuously make elastic container deployment plans in geographically distributed clouds, aim to maintain performance while minimizing the operating costs. Also, the authors in [165], propose a hybrid GA-based approach to

deploy a new type of composite application in multi-cloud. Its main goal is to optimize the 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, the authors in [164] turn to learning-based method. They adopt a deep reinforcement learning to dispatch the new arriving requests for applications in multi-cloud, 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 remote cloud to process commonly leads to significant propagation delays, bandwidth and energy consumption. It drives the centralized cloud to sink their 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 cloud-edge synergy paradigm. There are a plenty of researcher studies to investigate these challenges [45], [74], [161], [166]–[172]. Their optimization objectives mainly include response time and energy consumption. Base 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, the authors in [45], [161], [166], [167], [169], [173], adopt Markov decision process, reinforcement learning, deep reinforcement learning and heuristic methods to offload the containerized applications in cloud-edge synergy paradigm. Their main goals is to optimize latency. Specifically, in [45], a learning-based scheduling framework for edge-cloud systems is designed to dispatch service request and orchestrate multiple microservices instances, the goal of which is to improve the long-term system throughput rate. Analogously, in [168], 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, the authors in [74] adopt best-fit algorithms to place the containers, aiming to reduce the energy consumption. Also, in [71], a competent controller is presented to schedule containerized applications in edge-cloud system, aiming at minimizing the interference and the energy consumption.

VI. Service Operate

Service operation, which encompasses load balancing, service migration, and resource auto-scaling, is crucial for maintaining a high-performing and efficient system infrastructure. By integrating load balancing, service migration, and resource auto-scaling in the operational state, we can enable robust and efficient service management dynamically and at scale. Representative works are listed in Table IV for summarization.

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 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 Sec. VI-A1, we will provide a comprehensive analysis of load-balancing algorithms. Sec. 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. 5. Centralized load balancing can further be classified into static and dynamic algorithms based on whether the algorithm incorporates prior knowledge of the system. On the other hand, distributed load balancing employs multi-agent algorithms that offer greater flexibility and scalability than centralized algorithms.

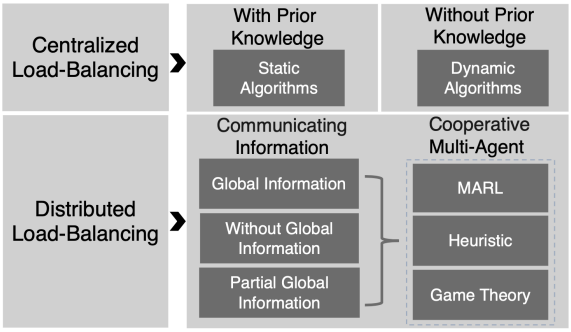

Fig. 5. Algorithms for load-balancing.

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, FIFO, MIN-MIN, and others, 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, Zhang et al. [174] 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. [156] 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

TABLE IV
REPRESENTATIVE WORKS IN SERVICE OPERATION.

WORK	METRIC					RESOURCE				ALGORITHM
	QoS	Throughput	Makespan	Response time	Revenue	CPU	RAM	B.W.	Storage	
[174]			✓			✓		✓		Heuristic
[156]		✓				✓	✓	✓	✓	Heuristic
[175]					✓	✓			✓	Game theory
[57], [176]			✓			✓				Heuristic
[177]	✓	✓				✓				LP
[178]	✓							✓	✓	Heuristic
[179]	✓					✓		✓		Approx. algorithm
[180]				✓		✓	✓			Heuristic
[181]	✓					✓				Epidemic
[182]	✓					✓		✓		Game theory
[183]			✓			✓				MARL
[184]				✓		✓				DQN
[185]		✓		✓		✓		✓		MARL
[186]				✓		✓	✓			Heuristic
[187], [188]				✓		✓	✓			-
[189]				✓		✓	✓	✓		Linear regression
[190]	✓				✓	✓		✓		MIP
[191]					✓	✓		✓		-
[192]				✓		✓		✓		-
[193]		✓		✓		✓		✓		-
[194]	✓			✓	✓	✓				RL
[195]		✓		✓		✓	✓	✓		-
[196]				✓			✓	✓		-
[197]	✓			✓			✓	✓		-
[198]				✓		✓		✓		-
[199], [200]		✓		✓	✓	✓		✓		Heuristic
[201]		✓		✓		✓	✓	✓		-
[202]		✓		✓		✓				PID
[203]				✓		✓				MAP
[204]		✓		✓		✓	✓			MPC
[205]				✓		✓	✓			PID
[206]	✓			✓		✓				Fuzzy logic
[207]	✓	✓			✓	✓		✓		FTRL
[208]	✓			✓		✓	✓			ILP
[209]	✓			✓		✓	✓	✓		Heuristic
[210]		✓		✓		✓		✓		RL

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.

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. Hongli et al. [175] propose a theoretical game algorithm that balances tasks by offloading them to edge servers while ensuring Service Level Objectives (SLOs). Another approach to dynamic load balancing revolves around resource allocation. Fatemeh et al. [57] design a heuristic algorithm-based optimization algorithm that evaluates overloaded, loaded, and balanced virtual machines to achieve load balancing. However, these efficiency-focused load-balancing algorithms may not be suitable for large-scale virtual machine environments.

To meet the requirements of reliability and elasticity in large-scale systems, Mohit et al. [176] propose a dynamic scheduling algorithm based on the last optimal k-interval virtual machines strategy that balances workload through resource provisioning and de-provisioning methods. This algorithm effectively balances the workload of virtual machines in a cloud environment through resource provisioning and de-provisioning methods. Regarding load balancing among microservices, Ruozhou et al. [177] introduce a graph-based model to analyze dependencies among microservices and adopted 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. Lemei et al. [178] 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, Jingzhou et al. [179] propose an approximation algorithm with a polynomial-time complexity that follows a two-step process to implement service deployment.

Multi-agent 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 et al. [180] design a distributed approach that focuses on migrating virtual machines (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, Harshitha et al. [181] 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. [182], 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's 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 multi-agent reinforcement learning (MARL), have gained popularity in this domain. Zhiyuan et al. [183] propose a 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. Ali et al. [184] design a multi-agent 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. Omar et al. [185] utilize a graph neural network (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 virtual machines and can be easily deployed. Additionally, the load balancer should provide a secure environment for the system and its users. Some popular load balancing solutions include Nginx [211], a widely-deployed reverse proxy server, and HAProxy [212], a fast and efficient reverse proxy software. Recent advancements in load balancing technology include Maglev [213], 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 [214] is another load balancer that supports uniform load distribution with per-connection consistency.

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. 6.

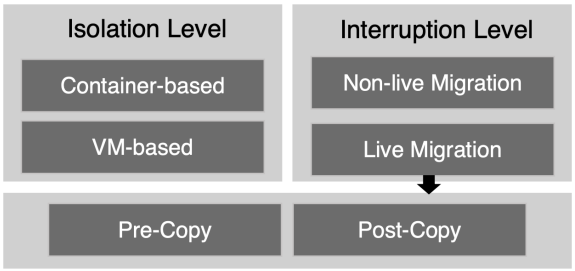

Fig. 6. Classification of service migration.

Live vs. non-live service migration. The main difference between live and non-live migration is as follows: non-live migration requires the system to be shut down during the migration process, while live migration involves migrating running systems. Non-live 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. Additionally, 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 [215].

Live migration predominantly employs two methods: pre-copy and post-copy [186]. The pre-copy 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 post-copy migration algorithm offers shorter overall migration time. However, this approach can cause page faults during the migration process, resulting in degraded performance and reduced stability of the virtual machine [216]. To address the challenges of post-copy migration, fault tolerance becomes necessary after a failure during the recovery of a virtual machine. 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 [187]. Additionally, an optimized post-copy 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 [188]. These advancements in live migration techniques aim to improve the efficiency, reliability, and performance of the migration process. The choice between pre-copy and post-copy methods depends on factors such as migration time, system stability, fault tolerance requirements, and the impact on the virtual machine's performance.

VM-based vs. container-based service migration. Existing works have optimized the VM-based migration process from various perspectives including modeling [189], [190] , prediction [189], [191] , latency [192], [193], energy consumption [194], etc. The dynamics of workloads make live migration modeling challenging. Jo et al. propose a machine learning-based model to enhance the prediction accuracy, considering critical characteristics of live migration [189]. Nguyen et al. 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 [190]. Maintaining uninterrupted uptime is crucial for live migration, particularly in large-scale systems with frequent infrastructure changes. Adam et al. propose a live VM migration scheme that minimizes the impact on users while addressing version updates and security concerns [191]. Bandwidth-Aware Compression (BAC) focuses on the trade-off between VM compression and transmission during migration [192]. The utilization of multi-page compression techniques enables an efficient migration scheme, reducing total migration latency while maintaining performance comparable to benchmarks. To enhance performance during live migration, Franck et al. propose a new Multi-Path TCP method over WAN, which significantly decreases round-trip latency and improves responsiveness and user engagement [193]. For energy consumption reduction and resource allocation in cloud-native environments, Basu et al. adopt a reinforcement learning algorithm to make optimal decisions regarding virtual machine migration [194]. 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 [195]–[198], [217]. Cloud-native platforms like K8s and Docker Swarm offer efficient handoff capabilities during container migration. To reduce handoff latency during migration, Lele et al. propose a framework that enables mobile users to offload their tasks to edge servers through seamless migration of container-based services [217]. In order to provide users with the freedom to choose cloud-native platforms, Thad et al. introduce a tool called CloudHopper, which facilitates the movement of containers between different platforms [195]. 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, Piush et al. present mWarp, a live container migration tool that efficiently remaps the physical memory of containers [196]. Bo proposes an efficient live migration system called Sledge, which integrates images and management context to reduce migration overhead and improve quality of service (QoS) with minimal downtime. The system employs a dynamic context-loading mechanism to minimize downtime during migration [197]. Although containers boot faster than VMs, their behavior during live migration under non-ideal conditions remains a question. Roberto et al. 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 [198]. 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 the upgrade during service operations, load balancing between clusters, and service deployment between cloud vendors. The most popular hypervisors used for migration in cloud-native are as follows.

- Kernel-based Virtual Machine (KVM) is a module in the Linux kernel used to virtualize 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 [24].
- 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, and FreeBSD, etc. It allows live migration between multiple hosts seamlessly [218].
- 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 [219].
- 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 [220].

C. Resource Auto-Scaling

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 *auto-scaling* method to efficiently utilize the elastic feature of resources according to its budget and profit. This section introduces *auto-scaling* in three categories, i.e., vertical scaling, horizontal scaling, and hybrid auto-scaling in Sec. VI-C1. Tools and systems are presented in Sec. VI-C2.

1) Algorithm Design and Analysis: According to different policies adopted by auto-scaling, we summarise auto-scaling into three categories as shown in Fig. 7.

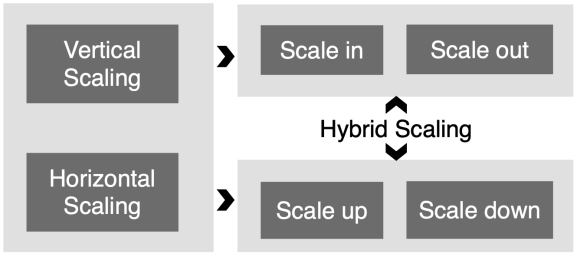

Fig. 7. Techniques for Resource Auto-Scaling.

Horizontal. Horizontal scaling is a widely adopted approach for auto-scaling in cloud-native environments, enabling applications to dynamically adjust the number of virtual machines (VMs) to scale resources. Researchers have explored cost-efficient methods [199], [200] and fast scaling approaches [201], [221] in this domain. Cost-effectiveness is a crucial consideration in implementing horizontal scaling in cloud-native systems. Romero et al. introduce INFaaS, which optimizes resource cost efficiency for machine learning inference applications with evolving dynamic requirements [199]. Zhang et al. propose a solution for auto-scaling in ML-as-a-Service using an LSTM network for workload prediction and a heuristic method for optimal instance provisioning decisions [200]. The ability to respond quickly with guaranteed response times is a key requirement in cloud-native platforms. Somma et al. present a fast resource provisioning method consisting of deploying containers responsible for application services and auto-scaling resource allocations among containers [221]. Shillaker et al. design Faaslets, a lightweight horizontal-scaling approach for containers in clusters [201]. In the context of virtualized network functions (VNFs) in cloud-native systems, auto-scaling techniques have a significant impact on on-the-fly provisioned infrastructure performance. Salhab et al. propose a framework to address resource provisioning on-demand for the 5G core network through auto-scaling of constrained resources [222]. Akhtar et al. design a horizontal scaling manager on virtualized infrastructure to balance traffic workload for security network functions [202]. 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.

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 [203]–[206]. To achieve cost-effective resource allocation in vertical scaling, Russo et al. propose MEAD, which utilizes a prediction algorithm based on Markovian Arrival Processes to handle bursty workloads, along with an auto-scaling module for resource allocation [203]. Lakew et al. address the resource allocation problem by employing a fine-grained vertical scaling technique that adapts to varying workloads in cloud-native systems [204]. Efficiency improvement in vertical scaling has also been a focus of research. Sfakianakis et al. introduce LatEst, a vertical scaling strategy that predicts bursts in serverless cloud systems and allocates resources efficiently within minimal time [205]. Tesfatsion et al. aim to increase resource usage and reduce energy consumption through long-term optimization using vertical scaling techniques [206]. In the context of avoiding overloaded Virtualized Network Functions (VNFs), Fei et al. 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 [207]. 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.

Hybrid. Hybrid scaling is a method commonly used in cloud-native networks that combines 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. 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 [223]. Avgeris et al. 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 [208]. In terms of minimizing resource costs, Mahmud et al. 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 [209]. Schuler et al. 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 [210]. 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 auto-scaling are listed as follows.

- HPA [224] is a fundamental horizontal-scaling strategy in k8s framework, with the target of re-allocating 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.
- AWS Lambda function scaling [225] 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.
- Knative Pod Autoscaler(KPA) [226] is an auto-scaling 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 auto-scaling 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. Auto-scaling 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.

- *Heterogeneous workloads.* Different workload has different resource demand for computing and bandwidth. The resource allocation granularity is a key for the performance of load balancing. Allocating too many resources leads to a waste while allocating too few resources causes congestion.
- *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.
- *Configuration management*. Migrating services often involves configuring multiple components (e.g., databases, web servers) to work together seamlessly. Keeping track of configurations and ensuring they are properly migrated can be challenging.

VII. Service Maintenance

Service maintenance collects and analyzes service and system indicators, adjusts and develop strategies, and performs fault recovery. This section is organized as Fig. 8 shows. After introducing the data collection, we describe the research status of data analysis of cloud and the evolution based on data analysis. We summarize and classify the representative works based on the main purpose of them, as shown in Table V.

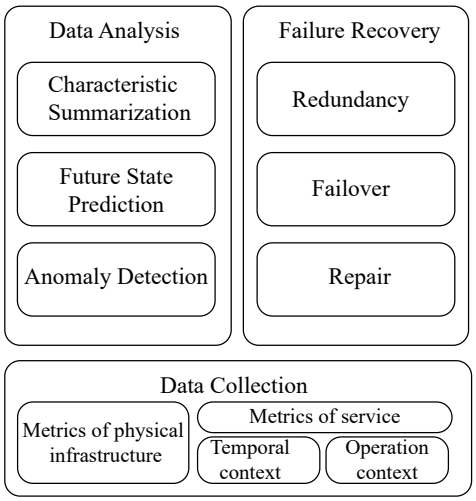

Fig. 8. Organization of service maintenance.

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: firstly, monitor the whole physical infrastructure of cloud environment. Secondly, monitor the performance of each service. For the former one, the healthy state of

TABLE V
REPRESENTATIVE WORKS IN SERVICE MAINTENANCE.

WORK	QOS	SERVER UTILIZATION	RISK CONTROL	COST AND PROFIT
[227]–[229]	✓		✓	
[230]		✓		✓
[231]	✓	✓		✓
[232]	✓	✓		
[233], [234]	✓	✓		
[58], [235]–[238]	✓		✓	
[239], [240]			✓	✓
[241]	✓		✓	
[242], [243]			✓	
[244]–[246]			✓	✓

disks, the usage of memory, processors, network bandwidth are usually focused on. There are lots of mature tools to detect the metrics of cloud infrastructures: such as DX [247], CA [248], etc. When analysing the performance of cloud services, there are mainly two kinds of contexts: the temporal context and the operation context [232]. The operation context means the metrics directly related to the service at each level, from software to the kernel. For example, the cache, processors, memory usage directly caused by the monitoring service. Besides, the temporal context is also important. The temporal context is the behavior of the services which have resource competition relationship with monitoring service. The challenge for collecting the operation context is that it is difficult to trace the the operation from layer to layer (i.e. from service layer to kernel layer). DTrace can solve this problem by instrumenting all code [249]. Besides, Ardelean et al. 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 [250] collects all the requests across multiple nodes. But it is not suitable for the cloud environment with billions users. To reduce the overhead of Magpie, a common way is to use bursty tracing [251], which just samples partial temporal context.

B. Data Analysis

Data analysis at service maintenance state is to mine the the property and regular patterns of running jobs from monitoring indexes and guide jobs allocation and scaling. Generally, there are three main research directions for data analysis: analyze and summarize the characteristic of running jobs and anomalous events to provide better understanding of cloud services, predict the future state of the cloud services, detect cloud service anomaly and find root cause automatically. We organize this section as following: in section VII-B1, we try to answer the two questions: what kinds of data is worth and suitable to analyze, 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 healthy state of disk, 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 challenges.

1) Summarize Cloud Services Characteristic: It is important to summarize the characteristic of cloud systems and the running cloud jobs, as it provides valuable insight to improve utilization of servers and quality of services. There are two important questions to summarize the characteristics. Firstly, what kind of data would contain valuable information and suitable to be analyzed? Secondly, researchers and cloud service providers are interested on what kind of characteristics? For the first question, Hauswirth et al. claimed the virtualization introduced by cloud native environment provides a significant challenge to understanding complete system performance, not found in traditionally compiled languages [252]. Thus, they proposes vertical profiling to provide profiling of all level of the execution stack. Vertical profiling can just apply to java-based applications. To make it more general, Ardelean et al. extend it to application based on any language [232]. Besides, code snippets [253], functional-level variance [254], control flow [255] are also suitable data. For the second question, characteristic summarizations usually concentrate on reducing the cost of providers, improving the quality of services and the healthy state of cloud systems. We list some of the most popular topics below: the different and similar impacts of different failures [227], the causes of failures [228], [229], the causes of low utilizations [230], pricing strategy analysis [231], the analysis of time-varying mixture load [232].

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, auto-scaling, 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, the forthcoming service failures.

Workload prediction. Workload prediction focus on predicting the future processor usage, memory usage and bandwidth usage, which can help to improve the quality of service(Qos) as well as improve the utilization of cloud servers. There are mainly two kinds of workload prediction methods: the statistical methods and neural-network based methods. For statistical methods, AR [256], MA [256], ARIMA [257], Bayesian models [258] are used to predict the future workloads. Among them, ARIMA is one of the most popular and classical methods, which assumes the value at present is affected by the trend information, the history values and

some noises. One of the problem of using ARIMA is that the workload of different jobs can have different regular patterns and it will lead to low accuracy to predict the workload by a single model. Thus, an adaptive statistic model [259] is proposed to solve this problem, which combines linear regression, ARIMA, and support vector regression.

Recently, it is 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 [234]. Thus, many researchers turn to neural-network based methods such as RNN [260], LSTM [261], etc. 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. But these recurrent network based methods give the same weights to the workload in observing window, while the history workload has different impact on predicting workload. Thus, a method combines LSTM and attention mechanism is proposed to put different weight on history workload [233]. Besides, another problem of using recurrent network based methods is the forgetting effect [262] when extracting long-term dependent information. Thus, a method [234] combines top-sparse auto-encoder and GRU is proposed to effectively extract the essential representations of workloads from the original high-dimensional workload data and predict highly variable workload accurately.

Healthy state of cloud servers prediction. Researchers in this domain mainly focus on disk dives 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 [58], 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 [263], Wilcoxon rank-sum test [264], support vector machines (SVM) [265] artificial neural network (ANN) [266], etc.

However, these methods have reported poor performance on real-world environment. Firstly, the status of disk drives corrupt gradually and is not only either good or bad [235] [58]. Thus, Aniello et al. propose a method firstly divide the healthy status of disks drives into seven levels by regression tree, and then use LSTM to predict the disk healthy status in the future [58]. The fine-grained disk drivers healthy status support more flexible data-restore mechanism, which can plan data restoring in advance according to the different prediction of fine-grained disk drivers healthy status. However, LSTM used in this work is recurrent networks and it is reported to be vulnerable to he highly variant interval between triggering events and hardware failures [236]. Thus, Sun et al. use the temporal Convolution 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 [236]. Secondly, the data imbalance between disk failure data and normal data hinders the models to predict accurately. Thus, Sun et al. also design a new loss function to prevent the gradient vanishing in front of the huge data imbalance [236]. Moreover, the above methods are based on offline training and can not adapt to the continuous update systems [237]. Thus, Xiao et al. proposes a method based on online random forest algorithm to maintain stable predicting accuracy for long-term usage.

Service failure prediction. Different from the above sections, service failure prediction focus on service Qos and predict 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 [238]. Thus, it is worthwhile to know the possible failure in advance. By doing so the cloud systems can take steps to prevent the predicting failures.

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

The rule based methods require experts define specific rules in advance. For example, PerfAugur [267] is designed to predict failures by specified features. Generally, these methods are accurate but just suitable for specific scenarios.

The statistic methods include ARMA [268], ARIMA [269], SVM [270], Hidden Semi-markov Model [271], etc. Among the classical statistic methods, Cavallo et al. [272] 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 [273]. However, Amin et al. [273] point out the traditional ARIMA model can not deal with the high volatility of quality of service (QoS) properly. Thus, they propose a model integrate GARCH and ARIMA to solve this problem.

However, statistic methods rely on some mathematics assumptions and can not work well on high dimension feature and dependent sequence data. Thus, many researchers apply the deep learning models to service failure prediction. Chen et al. [274] propose a deep learning methods based recurrent neural networks (RNN) to predict task-level failures. However, the drawback of RNN is that it will definitely forget the information in long distance, which will degrade the predicting accuracy. Although some modified recurrent neural networks, such as LSTM [275], can mitigate this effect, the weights put on each value in observing window is unequal and degrade as the distance goes farther. Thus, Gao et al. [276] propose a method based on bi-direction LSTM to further improve the accuracy.

3) Anomaly Detection: In the cloud native scene, researchers detect anomaly from different aspects: components anomalous usage (e.g. anomalous processor and memory usage, network attacks, disk drives failures), service anomalous Qos (e.g. high latency and low throughput). It is worth to notice 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 happens. The organization of this subsection is illustrated in Fig.9. There are three main challenges when detecting anomaly: labeled data obtaining issues, high variance of cloud environment, alert storm. The labeled data obtaining issues is caused by the requirements of expertise experience and lots of efforts to label the anomalous data. The high variance of cloud environment suggests that the detecting patterns the models learned from history data may become outdated frequently. Besides, there are lots of APIs and components in cloud environment. Lots of them are relevant to each other. When anomaly occurs to one part, the related parts will also be abnormal. Thus, when an anomaly occurs, the system always suffers from a alerting storm. This phenomena suggests the necessity of root cause finding.

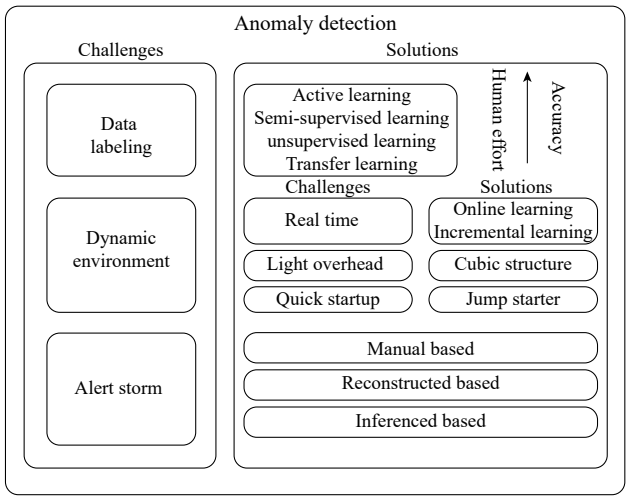

Fig. 9. Organization of anomaly detection.

Data labeling issue. The anomalous data labeling in cloud is different from labeling tasks in many other domains (e.g. image recognition), as it requires expertise experience to label the anomaly, which makes the labeled data rarer. There are mainly four kinds of methods to solve it: semi-supervised learning [277]–[280], unsupervised learning [281], [282], transfer learning [239], [283], [284], active learning [239], [240]. Semi-supervised learning is suitable for the datasets with small amount of labeled data. It firstly uses the labeled data to train a primary model. After that, it uses the primary model to assign pseudo label to the remaining unlabeled data. Then, it uses the data with pseudo label and data with label to retrain the primary model. If we not only have a dataset with small amount of labeled data, but also have experts to label some data in the process of model training, we can use the active learning to further improve the detecting accuracy. Active learning firstly uses the labeled data to train a primary model. After that, it uses the primary model to pick a small part of unlabeled data which need label most and label the remaining data with pseudo labels. Then, experts label the data picked by the primary model. Then, it retrains the primary model with newly labeled data, labeled data, pseudo labeled data. However, if we have none labeled data at all, we can use transfer learning or unsupervised learning. Both of them rely on assumptions: transfer learning assume there are some similarities between source dataset and target dataset, unsupervised learning assume the normal state has some unified and invariant latent regularity that can be learned by models. Autoencoder [285] is one of the most classical unsupervised learning models in anomaly detection. It firstly compresses the features into a smaller latent variable. After that, it reconstructs the features from the latent variable. In training process, it makes the reconstructed features as similar as possible to the original features. It assumes in the training process, autoencoder can learn the reconstructing patterns of normal data and when the autoencoder meets anomalous data in inferring process, it will fail to reconstruct. By then, autoencoder can detect anomalies. As each feature in cloud anomaly detection is time dependent, recently, some researchers combine the recurrent neural network with the autoencoder to capture the correlation between features as well as the time dependence [286], [287].

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

Cloud environment high variance issue. The cloud environment is highly variable. For example, Google and baidu are reported thousands of software changes are deployed everyday [241]. 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 to two streams: one is based on time dependent information, the other is based on the correlation between different metrics. The former one is to capture the time-dependent normal pattern from time series and compare the tested data with the normal pattern, while the latter one 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. [243] 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. [242] propose a method uses cuboid structure to store the relationship of different metrics, which significantly reduce the storing overhead. Besides, they also propose a incremental learning method suitable for cuboid structure, which significantly reduce the retraining overhead. However, it is reported the incremental retraining methods need lots of data points to converge [288] and still need more data to reach steady state [289], which requires the cloud system 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 [241] when the accuracy of old model is low and there is no enough data to apply the incremental learning. To reduce the initial time, Ma et al. [241] propose a quickly start method which can work

well without relying on as big amount of data as incremental learning and report high accuracy.

Alert storm issue. Cloud system is consisted of thousands of components with extraordinary complex dependency [244]. Besides, business transactions in a cloud native system usually have a much longer calling path with dozens of distributed microservices participating [245]. 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 [245]. 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, reconstruction based methods. Manual-rule based methods are based on expertise experience, which is accurate though human-work costly and easily-outdated. For example, Demirbaga et al. [246] defined three kinds anomaly cause: data locality (i.e. data needed is not at the same server as task), resource heterogeneity (i.e. jobs are scheduled to machine remaining few computing resource), network failure (i.e. network disconnection). They detect these anomaly by defining threshold of several metrics, when the defining combined metrics exceed the thresholds, then 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 [281]. They can just apply to the anomaly detecting methods based features reconstruction. This root cause finding methods work by compute the distances between every pair of reconstructed feature and original feature. 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 [245] use conditional dependence to establish the topology of cause and effect relationship among tasks and use the topology to find root cause. Sipple [290] uses 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 like Docker introduce almost zero overhead when launching a container, making it possible to re-spawn a failed container within seconds possible. Second, modern metric collection and processing systems like Prometheus [291] allow high integration into cloud-native systems, further enabling well-informed decisions. Third, with fast development of machine learning techniques, automatically-made decisions are getting yet 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, etc.), preferably in multiple locations, in order to minimize disruption even if one or several of them goes down. This method has already been in use before the container-based cloud-native era [292]. In addition to improved reliability, distributing data from multiple locations can lead to reduced latency to end users. Kang et al. [293] design and implement a custom controller in Kubernetes to select and use multiple replicas of Virtual Network Functions (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. [294] propose a novel encoding mechanism to save encode data before saving to multiple locations, instead of simply creating replicas. Furthermore, the authors mitigated increase in latency introduced by distributed storage by rethinking how consensus can be reached, to offer near-optimal latency versus cost trade-offs.

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 redundant deployment talked in the previous section. 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 et al. [295] 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 genetic algorithms to take latancies into consideration when making a failover decision. Jin et al. [296] build a SDN failover mechanism, FAVE, that is aware of physical link failure, to be used in virtualized SDN environments. Landa et al. [297] utilize TCP re-transmission metrics to declare network failure in CDN networks, and quick re-route traffic through redundant links to keep high availability.

3) Repair: Repair tries to fix the error instead of redirect 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. [298] consider the complexity in modern cloud deployments and identify that such complexity could lead to failure in deployments. They build AURA, which transform a deployment into a directed acyclic graph, so whenever an error happens, it is possible to respawn only a small portion of the entire deployment, thus keeping the repair process efficient.

D. Challenges and Research Opportunities

The 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 that whether to train only one model for all the servers or train 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 trade off between efficiency and accuracy.

The dynamic evolution of cloud environment. The cloud environment is dynamic. New missions arrives and old missions ends at every moment. The models generally becomes

outdated and need retraining frequently. Designing a light-weight retraining methods can bring huge benefit.

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 the open issues and possible research directions.

- *Hybrid multi-cloud 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 are applicaitons 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.
- *User-friendly servie shapes (forms).* As cloud-native being the foundation of today's most web applicaitons, 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 the infrastructure management. However, serverless computing is criticized by its long cold time, inefficient state management and other related issues. Better service shapes/forms for more wider application scenarios are required.
- *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.
- *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 multi-cloud scenarios. There is a string need for better security mechanisms that can effectively protect against cyber threats, especially as attacks become more sophisticated.

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. Conclusions

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, operate, 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.

Acknowledgment

This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB4500100, the National Science Foundation of China under Grants 62125206 and U20A20173, and the Key Research Project of Zhejiang Province under Grant 2022C01145.

References

[1] M. P. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.* IEEE, 2003, pp. 3–12.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: a research roadmap," *International Journal of Cooperative Information Systems*, vol. 17, no. 02, pp. 223–255, 2008.

[3] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA).* IEEE, 2016, pp. 44–51.

[4] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of devops concepts and challenges," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–35, 2019.

[5] F. Zampetti, S. Geremia, G. Bavota, and M. Di Penta, "CI/CD pipelines evolution and restructuring: A qualitative and quantitative study," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 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 Journal of Research and Development*, vol. 60, no. 4, pp. 12–1, 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 *2015 IEEE Conference on Communications and Network Security (CNS).* IEEE, 2015, pp. 559–567.

[8] "Docker: Modernize your applications, accelerate innovation," [n.d.]. [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 *Proceedings of the Tenth European Conference on Computer Systems*, 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," *Journal of Systems and Software*, vol. 126, pp. 1–16, 2017.

[12] N. Kratzke, "A brief history of cloud application architectures," *Applied Sciences*, vol. 8, no. 8, p. 1368, 2018.

[13] G. Gil, D. Corujo, and P. Pedreiras, "Cloud native computing for industry 4.0: Challenges and opportunities," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2021, pp. 01–04.

[14] 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.

[15] A. Senthuran and S. Hettiarachchi, "A review of dynamic scalability and dynamic scheduling in cloud-native distributed stream processing systems," *ICDSMLA 2019*, pp. 1539–1553, 2020.

[16] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys (CSUR)*, 2022.

[17] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for Fine-Grained resource sharing in the data center," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*. Boston, MA: USENIX Association, Mar. 2011. [Online]. Available: https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center

[18] N. Naik, "Building a virtual system of systems using docker swarm in multiple clouds," in *2016 IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, 2016, pp. 1–3.

[19] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013, pp. 1–16.

[20] W. Li, Y. Lemieux, J. Gao, Z. Zhao, and Y. Han, "Service mesh: Challenges, state of the art, and future research opportunities," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2019, pp. 122–1225.

[21] F. Soppelsa and C. Kaewkasi, *Native docker clustering with swarm*. Packt Publishing Ltd, 2016.

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

[23] W. Xu, "Test report on kubeedge's support for 100,000 edge nodes," https://kubeedge.io/en/blog/scalability-test-report/, 2022.

[24] "Kernel-based virtual machine." [Online]. Available: https://www.redhat.com/en/topics/virtualization/what-is-KVM

[25] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in *2015 IEEE International Conference on cloud engineering*. IEEE, 2015, pp. 386–393.

[26] "Docker: Modernize your applications, accelerate innovation," [n.d.]. [Online]. Available: https://www.docker.com/

[27] K. Kushwaha and N. Center, "How container runtimes matter in kubernetes?" 2017.

[28] D. B. Rawat and S. R. Reddy, "Software defined networking architecture, security and energy efficiency: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 325–346, 2016.

[29] J. Deng, H. Hu, H. Li, Z. Pan, K.-C. Wang, G.-J. Ahn, J. Bi, and Y. Park, "Vnguard: An nfv/sdn combination framework for provisioning and managing virtual firewalls," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE, 2015, pp. 107–114.

[30] 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 *2015 World Congress on Information Technology and Computer Applications (WCITCA)*. IEEE, 2015, pp. 1–6.

[31] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, and C. W. Mike Bursell, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks," https://datatracker.ietf.org/doc/rfc7348/, 2020.

[32] Y. Park, H. Yang, and Y. Kim, "Performance analysis of cni (container networking interface) based container network," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 2018, pp. 248–250.

[33] S. Qi, S. G. Kulkarni, and K. K. Ramakrishnan, "Assessing container network interface plugins: Functionality, performance, and scalability," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 656–671, 2021.

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

[35] 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, nov 2019. [Online]. Available: https://doi.org/10.1145/3359981

[36] T. K. Community, "Kubesphere devops: A powerful ci/cd platform built on top of kubernetes for devops-oriented teams," https://kubesphere.io/devops/, 2022.

[37] H. Chen, S. Deng, H. Zhu, H. Zhao, R. Jiang, S. Dustdar, and A. Y. Zomaya, "Mobility-Aware Offloading and Resource Allocation for Distributed Services Collaboration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2428–2443, 2022.

[38] C. Zheng, Q. Zhuang, and F. Guo, "A Multi-Tenant Framework for Cloud Container Services," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 359–369.

[39] Ł. Wojciechowski, K. Opasiak, J. Latusek, M. Wereski, V. Morales, T. Kim, and M. Hong, "NetMARKS: Network metrics-AwaRe kubernetes scheduler powered by service mesh," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–9.

[40] Y. Fu, S. Zhang, J. Terrero, Y. Mao, G. Liu, S. Li, and D. Tao, "Progress-based container scheduling for short-lived applications in a kubernetes cluster," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 278–287.

[41] S. Deng, H. Zhao, Z. Xiang, C. Zhang, R. Jiang, Y. Li, J. Yin, S. Dustdar, and A. Y. Zomaya, "Dependent Function Embedding for Distributed Serverless Edge Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2346–2357, 2021.

[42] 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 *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 2019, pp. 1–6.

[43] G. Zhang, R. Lu, and W. Wu, "Multi-resource fair allocation for cloud federation," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2019, pp. 2189–2194.

[44] 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 Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 88–100, 2021.

[45] Y. Han, S. Shen, X. Wang, S. Wang, and V. C. M. Leung, "Tailored learning-based scheduling for kubernetes-oriented edge-cloud system," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[46] T. Pusztai, A. Morichetta, V. C. Pujol, S. Dustdar, S. Nastic, X. Ding, D. Vij, and Y. Xiong, "Slo script: A novel language for implementing complex cloud-native elasticity-driven slos," in *2021 IEEE International Conference on Web Services (ICWS)*. IEEE, 2021, pp. 21–31.

[47] ——, "A novel middleware for efficiently implementing complex cloud-native slos," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021, pp. 410–420.

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

[49] R. Pinciroli, A. Ali, F. Yan, and E. Smirni, "Cedule+: Resource management for burstable cloud instances using predictive analytics," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 945–957, 2020.

[50] S. N. A. Jawaddi, M. H. Johari, and A. Ismail, "A review of microservices autoscaling with formal verification perspective," *Software: Practice and Experience*, 2022.

[51] Z. Wang, X. Tang, Q. Liu, and J. Han, "Jily: Cost-aware AutoScaling of heterogeneous GPU for DNN inference in public cloud," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2019, pp. 1–8.

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

[53] 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, 2020.

[54] J. Liu, S. Zhang, Q. Wang, and J. Wei, "Coordinating Fast Concurrency Adapting with Autoscaling for SLO-Oriented Web Applications," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[55] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 329–338.

[56] F. Rossi, V. Cardellini, F. L. Presti, and M. Nardelli, "Geo-distributed efficient deployment of containers with Kubernetes," *Computer Communications*, vol. 159, pp. 161–174, 2020.

[57] F. Ebadifard, S. M. Babamir, and S. Barani, "A dynamic task scheduling algorithm improved by load balancing in cloud computing," in *2020 6th International Conference on Web Research (ICWR)*. IEEE, 2020, pp. 177–183.

[58] A. De Santo, A. Galli, M. Gravina, V. Moscato, and G. Sperlì, "Deep learning for hdd health assessment: An application based on lstm," *IEEE Transactions on Computers*, vol. 71, no. 1, pp. 69–80, 2020.

[59] K. Nguyen, S. Drew, C. Huang, and J. Zhou, "Collaborative container-based parked vehicle edge computing framework for online task offloading," in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*. IEEE, 2020, pp. 1–6.

[60] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *IEEE INFOCOM 2019-IEEE conference on computer communications*. IEEE, 2019, pp. 505–513.

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

[62] M. F. Bestari, A. I. Kistijantoro, and A. B. Sasmita, "Dynamic Resource Scheduler for Distributed Deep Learning Training in Kubernetes," in *2020 7th International Conference on Advance Informatics: Concepts, Theory and Applications (ICAICTA)*. IEEE, 2020, pp. 1–6.

[63] M. Carvalho and D. F. Macedo, "QoE-Aware Container Scheduler for Co-located Cloud Environments," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 286–294.

[64] L. Toka, "Ultra-reliable and low-latency computing in the edge with kubernetes," *Journal of Grid Computing*, vol. 19, no. 3, pp. 1–23, 2021.

[65] A. Warke, M. Mohamed, R. Engel, H. Ludwig, W. Sawdon, and L. Liu, "Storage Service Orchestration with Container Elasticity," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2018, pp. 283–292.

[66] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini, and S. Cretti, "Foggy: a platform for workload orchestration in a fog computing environment," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 231–234.

[67] 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, 2021.

[68] 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. 183 879–183 890, 2020.

[69] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 351–359.

[70] 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, 2019.

[71] 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 of Things Journal*, vol. 7, no. 5, pp. 4228–4237, 2019.

[72] Z. Zhong, J. He, M. A. Rodriguez, S. Erfani, R. Kotagiri, and R. Buyya, "Heterogeneous task co-location in containerized cloud computing environments," in *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2020, pp. 79–88.

[73] Y.-G. Yim, H.-J. Jang, and H.-W. Jin, "QoS for best-effort batch jobs in container-based cloud," *Concurrency and Computation: Practice and Experience*, 2021.

[74] P. Chhikara, R. Tekchandani, N. Kumar, and M. S. Obaidat, "An efficient container management scheme for resource-constrained intelligent IoT devices," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12 597–12 609, 2020.

[75] M. Xu and R. Buyya, "Brownout approach for adaptive management of resources and applications in cloud computing systems: A taxonomy and future directions," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–27, 2019.

[76] J. R. Gunasekaran, P. Thinakaran, N. C. Nachiappan, M. T. Kandemir, and C. R. Das, "Fifer: Tackling resource underutilization in the serverless era," in *Proceedings of the 21st International Middleware Conference*, 2020, pp. 280–295.

[77] 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 *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2019, pp. 1–13.

[78] A. Chung, J. W. Park, and G. R. Ganger, "Stratus: Cost-aware container scheduling in the public cloud," in *Proceedings of the ACM symposium on cloud computing*, 2018, pp. 121–134.

[79] Z. Zhong and R. Buyya, "A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 2, pp. 1–24, 2020.

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

[81] M. Xu, A. N. Toosi, and R. Buyya, "A self-adaptive approach for managing applications and harnessing renewable energy for sustainable cloud computing," *IEEE Transactions on Sustainable Computing*, vol. 6, no. 4, pp. 544–558, 2020.

[82] I. Kim, K. J. Oh, and Y. I. Eom, "Overlit: New Storage Driver for Localization and Specialization," in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2019, pp. 1–4.

[83] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016, pp. 44–51.

[84] K. Gos and W. Zabierowski, "The comparison of microservice and monolithic architecture," in *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*. IEEE, 2020, pp. 150–153.

[85] L. De Lauretis, "From monolithic architecture to microservices architecture," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2019, pp. 93–96.

[86] F. Ponce, G. Márquez, and H. Astudillo, "Migrating from monolithic architecture to microservices: A rapid review," in *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*. IEEE, 2019, pp. 1–7.

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

[88] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys (CSUR)*, 2022.

[89] 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 Journal on Selected Areas in Communications*, 2022.

[90] A. Xu, Z. Huo, and H. Huang, "On the acceleration of deep learning model parallelism with staleness," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2088–2097.

[91] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[92] H. Wang, Z. Liu, and H. Shen, "Job scheduling for large-scale machine learning clusters," in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, 2020, pp. 108–120.

[93] 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 *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 695–705.

[94] X. Wu, H. Xu, and Y. Wang, "Irina: Accelerating dnn inference with efficient online scheduling," in *4th Asia-Pacific Workshop on Networking*, 2020, pp. 36–43.

[95] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: A gpu cluster engine for accelerating dnn-based video analysis," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 322–337.

[96] N. Zhou, Y. Georgiou, M. Pospieszny, L. Zhong, H. Zhou, C. Niethammer, B. Pejak, O. Marko, and D. Hoppe, "Container orchestration on hpc systems through kubernetes," *Journal of Cloud Computing*, vol. 10, no. 1, pp. 1–14, 2021.

[97] C. Misale, M. Drocco, D. J. Milroy, C. E. A. Gutierrez, S. Herbein, D. H. Ahn, and Y. Park, "It's a Scheduling Affair: GROMACS in the Cloud with the KubeFlux Scheduler," in *2021 3rd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. IEEE, 2021, pp. 10–16.

[98] C. Misale, D. J. Milroy, C. E. A. Gutierrez, M. Drocco, S. Herbein, D. H. Ahn, Z. Kaiser, and Y. Park, "Towards standard Kubernetes scheduling interfaces for converged computing," in *Smoky Mountains Computational Sciences and Engineering Conference*. Springer, 2021, pp. 310–326.

[99] 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 *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. IEEE, 2019, pp. 11–20.

[100] S. López-Huguet, J. D. Segrelles, M. Kasztelnik, M. Bubak, and I. Blanquer, "Seamlessly managing HPC workloads through Kubernetes," in *International Conference on High Performance Computing*. Springer, 2020, pp. 310–320.

[101] S. Choochotkaew, T. Chiba, S. Trent, T. Yoshimura, and M. Amaral, "AutoDECK: Automated Declarative Performance Evaluation and Tuning Framework on Kubernetes," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE, 2022, pp. 309–314.

[102] D. Fan and D. He, "Knative Autoscaler Optimize Based on Double Exponential Smoothing," in *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*. IEEE, 2020, pp. 614–617.

[103] W. Ling, L. Ma, C. Tian, and Z. Hu, "Pigeon: A dynamic and efficient serverless and faas framework for private cloud," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2019, pp. 1416–1421.

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

[105] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica, "The power of choice in {Data-Aware} cluster scheduling," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 301–316.

[106] S. Venkataraman, A. Panda, K. Ousterhout, M. Armbrust, A. Ghodsi, M. J. Franklin, B. Recht, and I. Stoica, "Drizzle: Fast and adaptable stream processing at scale," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 374–389.

[107] H. Wang, D. Niu, and B. Li, "Distributed machine learning with a serverless architecture," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1288–1296.

[108] R. Gu, K. Zhang, Z. Xu, Y. Che, B. Fan, H. Hou, H. Dai, L. Yi, Y. Ding, G. Chen, and Others, "Fluid: Dataset abstraction and elastic acceleration for cloud-native deep learning training jobs," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 2182–2195.

[109] 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. 105 192–105 204, 2021.

[110] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, and J. Murphy, "A WOA-based optimization approach for task scheduling in cloud computing systems," *IEEE Systems Journal*, vol. 14, no. 3, pp. 3117–3128, 2020.

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

[112] L. Ye, Y. Xia, L. Yang, and C. Yan, "SHWS: Stochastic Hybrid Workflows Dynamic Scheduling in Cloud Container Services," *IEEE Transactions on Automation Science and Engineering*, 2021.

[113] Y. Hu, H. Zhou, C. de Laat, and Z. Zhao, "Concurrent container scheduling on heterogeneous clusters with multi-resource constraints," *Future Generation Computer Systems*, vol. 102, pp. 562–573, 2020.

[114] 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 *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1569–1578.

[115] 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 Systems Journal*, 2021.

[116] W. Zheng, M. Tynes, H. Gorelick, Y. Mao, L. Cheng, and Y. Hou, "Flowcon: Elastic flow configuration for containerized deep learning applications," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.

[117] 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 *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–11.

[118] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.

[119] P. Li, E. Koyuncu, and H. Seferoglu, "Respipe: Resilient model-distributed dnn training at edge networks," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 3660–3664.

[120] Z. Luo, X. Yi, G. Long, S. Fan, C. Wu, J. Yang, and W. Lin, "Efficient pipeline planning for expedited distributed dnn training," *arXiv preprint arXiv:2204.10562*, 2022.

[121] X. Yi, Z. Luo, C. Meng, M. Wang, G. Long, C. Wu, J. Yang, and W. Lin, "Fast training of deep learning models over multiple gpus," in *Proceedings of the 21st International Middleware Conference*, 2020, pp. 105–118.

[122] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "Tensorflow-serving: Flexible, high-performance ml serving," *arXiv preprint arXiv:1712.06139*, 2017.

[123] "Multi model server: a tool for serving neural net models for inference," 2022. [Online]. Available: https://github.com/awslabs/multi-modelserver

[124] 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, 2021.

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

[126] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 220–233.

[127] D. Mendoza, F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "Interference-aware scheduling for inference serving," in *Proceedings of the 1st Workshop on Machine Learning and Systems*, 2021, pp. 80–88.

[128] N. Zhou, H. Zhou, and D. Hoppe, "Containerisation for high performance computing systems: Survey and prospects," *IEEE Transactions on Software Engineering*, 2022.

[129] A. Reuther, C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, M. Jones, P. Michaleas, A. Prout, A. Rosa *et al.*, "Scalable system scheduling for hpc and big data," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 76–92, 2018.

[130] M. A. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. Cunha, and R. Buyya, "Hpc cloud for scientific and business applications: taxonomy, vision, and research challenges," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–29, 2018.

[131] Y. Fan, "Job scheduling in high performance computing," *arXiv preprint arXiv:2109.09269*, 2021.

[132] Q. Wofford, P. G. Bridges, and P. Widener, "A layered approach for modular container construction and orchestration in hpc environments," in *Proceedings of the 11th Workshop on Scientific Cloud Computing*, 2020, pp. 1–8.

[133] S. Julian, M. Shuey, and S. Cook, "Containers in research: initial experiences with lightweight infrastructure," in *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, 2016, pp. 1–6.

[134] J. Higgins, V. Holmes, and C. Venters, "Orchestrating docker containers in the hpc environment," in *International Conference on High Performance Computing*. Springer, 2015, pp. 506–513.

[135] N. Zhou, Y. Georgiou, L. Zhong, H. Zhou, and M. Pospieszny, "Container orchestration on hpc systems," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 34–36.

[136] N. Zhou, "Containerization and orchestration on hpc systems," in *Sustained Simulation Performance 2019 and 2020*. Springer, 2021, pp. 133–147.

[137] N. Zhou, L. Zhong, D. Hoppe, B. Pejak, O. Marko, J. Cardona, M. Czerkawski, I. Andonovic, C. Michie, C. Tachtatzis *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*. CRC Press, 2022, pp. 255–272.

[138] F. Liu, K. Keahey, P. Riteau, and J. Weissman, "Dynamically negotiating capacity between on-demand and batch clusters," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 493–503.

[139] M. E. Piras, L. Pireddu, M. Moro, and G. Zanetti, "Container orchestration on hpc clusters," in *International Conference on High Performance Computing*. Springer, 2019, pp. 25–35.

[140] J. Carnero and F. J. Nieto, "Running simulations in hpc and cloud resources by implementing enhanced tosca workflows," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2018, pp. 431–438.

[141] E. Di Nitto, J. Gorroñogoitia, I. Kumara, G. Meditskos, D. Radolović, K. Sivalingam, and R. S. González, "An approach to support automated deployment of applications on heterogeneous cloud-hpc infrastructures," in *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2020, pp. 133–140.

[142] I. Colonnelli, B. Cantalupo, I. Merelli, and M. Aldinucci, "Streamflow: cross-breeding cloud with hpc," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1723–1737, 2020.

[143] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[144] "kube-batch," 2019. [Online]. Available: https://github.com/kubernetes-sigs/kube-batch

[145] ©Kubeless 2022 project authors, "The kubernetes native serverless framework," 2022, https://kubeless.io/.

[146] H. Govind and H. González-Vélez, "Benchmarking serverless workloads on kubernetes," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 704–712.

[147] K. Djemame, M. Parker, and D. Datsev, "Open-source serverless architectures: an evaluation of apache openwhisk," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2020, pp. 329–335.

[148] S. K. Mohanty, G. Premsankar, M. Di Francesco *et al.*, "An evaluation of open source serverless computing frameworks." *CloudCom*, vol. 2018, pp. 115–120, 2018.

[149] O. Mashayekhi, H. Qu, C. Shah, and P. Levis, "Execution templates: Caching control plane decisions for strong scaling of data analytics," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 513–526.

[150] J. Huang, C. Xiao, and W. Wu, "Rlsk: a job scheduler for federated kubernetes clusters based on reinforcement learning," in *2020 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2020, pp. 116–123.

[151] Z. Gu, S. Tang, B. Jiang, S. Huang, Q. Guan, and S. Fu, "Characterizing Job-Task Dependency in Cloud Workloads Using Graph Learning," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 288–297.

[152] "Hadoop: Fair scheduler," April 2019. [Online]. Available: https://hadoop.apache.org/docs/current/hadoopyarn/hadoop-yarnsite/FairScheduler.html

[153] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Transactions on Computers*, vol. 63, no. 11, pp. 2647–2660, 2013.

[154] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 455–466, 2014.

[155] S. Wang, Z. Ding, and C. Jiang, "Elastic scheduling for microservice applications in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 98–115, 2020.

[156] D. Zhao, M. Mohamed, and H. Ludwig, "Locality-aware scheduling for containers in cloud computing," *IEEE Transactions on cloud computing*, vol. 8, no. 2, pp. 635–646, 2018.

[157] L. Bulej, T. Bureš, P. Hnětynka, and D. Khalyeyev, "Self-adaptive K8S Cloud Controller for Time-sensitive Applications," in *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2021, pp. 166–169.

[158] G. Ambrosino, G. B. Fioccola, R. Canonico, and G. Ventre, "Container Mapping and its Impact on Performance in Containerized Cloud Environments," in *2020 IEEE International Conference on Service Oriented Systems Engineering (SOSE)*. IEEE, 2020, pp. 57–64.

[159] 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 Transactions on Parallel and Distributed Systems*, 2022.

[160] F. Rossi, V. Cardellini, and F. L. Presti, "Elastic deployment of software containers in geo-distributed computing environments," in *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2019, pp. 1–7.

[161] A. Das, S. Imai, S. Patterson, and M. P. Wittie, "Performance optimization for edge-cloud serverless platforms via dynamic task placement," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 41–50.

[162] N. Akhtar, A. Raza, V. Ishakian, and I. Matta, "Cose: Configuring serverless functions using statistical learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 129–138.

[163] Y. Aldwyan, R. O. Sinnott, and G. T. Jayaputera, "Elastic deployment of container clusters across geographically distributed cloud data centers for web applications," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 21, p. e6436, 2021.

[164] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Location-Aware and Budget-Constrained Service Brokering in Multi-Cloud via Deep Reinforcement Learning," in *International Conference on Service-Oriented Computing*. Springer, 2021, pp. 756–764.

[165] ——, "Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1954–1969, 2020.

[166] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "Fscaler: Automatic resource scaling of containers in fog clusters using reinforcement learning," in *2020 international wireless communications and mobile computing (IWCMC)*. IEEE, 2020, pp. 1824–1829.

[167] M. Yan, X. Liang, Z. Lu, J. Wu, and W. Zhang, "Hansel: Adaptive horizontal scaling of microservices using bi-lstm," *Applied Soft Computing*, vol. 105, p. 107216, 2021.

[168] A. Marchese and O. Tomarchio, "Network-Aware Container Placement in Cloud-Edge Kubernetes Clusters," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 859–865.

[169] 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 Transactions on Mobile Computing*, vol. 20, no. 3, pp. 939–951, 2019.

[170] T. Pusztai, S. Nastic, A. Morichetta, V. C. Pujol, P. Raith, S. Dustdar, D. Vij, Y. Xiong, and Z. Zhang, "Polaris scheduler: Slo-and topology-aware microservices scheduling at the edge," in *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2022, pp. 61–70.

[171] T. Pusztai, F. Rossi, and S. Dustdar, "Pogonip: Scheduling asynchronous applications on the edge," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 660–670.

[172] S. Nastic, T. Pusztai, A. Morichetta, V. C. Pujol, S. Dustdar, D. Vii, and Y. Xiong, "Polaris scheduler: Edge sensitive and slo aware workload scheduling in cloud-edge-iot clusters," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 206–216.

[173] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 712–725, 2018.

[174] 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 Generation Computer Systems*, vol. 115, pp. 497–516, 2021.

[175] H. Lu, G. Xu, C. W. Sung, S. Mostafa, and Y. Wu, "A game theoretical balancing approach for offloaded tasks in edge datacenters," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022, pp. 526–536.

[176] M. Kumar and S. C. Sharma, "Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment," *Computers & Electrical Engineering*, vol. 69, pp. 395–411, 2018.

[177] R. Yu, V. T. Kilari, G. Xue, and D. Yang, "Load balancing for interdependent iot microservices," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 298–306.

[178] L. Huang, S. Cheng, Y. Guan, X. Zhang, and Z. Guo, "Consistent user-traffic allocation and load balancing in mobile edge caching," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 592–597.

[179] J. Wang, G. Zhao, H. Xu, H. Huang, L. Luo, and Y. Yang, "Robust service mapping in multi-tenant clouds," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[180] J. O. Gutierrez-Garcia and A. Ramirez-Nafarrate, "Agent-based load balancing in cloud data centers," *Cluster Computing*, vol. 18, no. 3, pp. 1041–1062, 2015.

[181] H. Menon and L. Kalé, "A distributed dynamic load balancer for iterative applications," in *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2013, pp. 1–11.

[182] X. Xu, Q. Jiang, P. Zhang, X. Cao, M. R. Khosravi, L. T. Alex, L. Qi, and W. Dou, "Game theory for distributed iov task offloading with fuzzy neural network in edge computing," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 11, pp. 4593–4604, 2022.

[183] Z. Yao, Z. Ding, and T. Clausen, "Multi-agent reinforcement learning for network load balancing in data center," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 3594–3603.

[184] 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 Computing*, vol. 25, no. 1, pp. 119–140, 2022.

[185] O. Houidi, S. Bakri, and D. Zeghlache, "Multi-agent graph convolutional reinforcement learning for intelligent load balancing," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–6.

[186] A. Shribman and B. Hudzia, "Pre-copy and post-copy vm live migration for memory intensive applications," in *European Conference on Parallel Processing*. Springer, 2012, pp. 539–547.

[187] 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 *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 343–351.

[188] 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 *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. IEEE, 2019, pp. 16–24.

[189] C. Jo, Y. Cho, and B. Egger, "A machine learning approach to live migration modeling," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 351–364.

[190] N. T. Khai, A. Baumgartner, and T. Bauschert, "A multi-step model for migration and resource reallocation in virtualized network infrastructures," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2017, pp. 730–735.

[191] A. Ruprecht, D. Jones, D. Shiraev, G. Harmon, M. Spivak, M. Krebs, M. Baker-Harvey, and T. Sanderson, "Vm live migration at scale," *ACM SIGPLAN Notices*, vol. 53, no. 3, pp. 45–56, 2018.

[192] C. Li, D. Feng, Y. Hua, W. Xia, L. Qin, Y. Huang, and Y. Zhou, "Bac: Bandwidth-aware compression for efficient live migration of virtual machines," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[193] F. Le and E. M. Nahum, "Experiences implementing live vm migration over the wan with multi-path tcp," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1090–1098.

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

[195] T. Benjaponpitak, M. Karakate, and K. Sripanidkulchai, "Enabling live migration of containerized applications across clouds," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2529–2538.

[196] P. K. Sinha, S. S. Doddamani, H. Lu, and K. Gopalan, "mwarp: accelerating intra-host live container migration via memory warping," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 508–513.

[197] B. Xu, S. Wu, J. Xiao, H. Jin, Y. Zhang, G. Shi, T. Lin, J. Rao, L. Yi, and J. Jiang, "Sledge: Towards efficient live migration of docker containers," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 321–328.

[198] R. Torre, E. Urbano, H. Salah, G. T. Nguyen, and F. H. Fitzek, "Towards a better understanding of live migration performance with docker containers," in *European Wireless 2019; 25th European Wireless Conference*. VDE, 2019, pp. 1–6.

[199] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "{INFaaS}: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 397–411.

[200] C. Zhang, M. Yu, W. Wang, and F. Yan, "{MArk}: Exploiting cloud services for {Cost-Effective},{SLO-Aware} machine learning inference serving," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 1049–1062.

[201] S. Shillaker and P. Pietzuch, "Faasm: Lightweight isolation for efficient stateful serverless computing," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 419–433.

[202] N. Akhtar, I. Matta, A. Raza, and Y. Wang, "El-sec: Elastic management of security applications on virtualized infrastructure," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 778–783.

[203] G. R. Russo, V. Cardellini, G. Casale, and F. L. Presti, "Mead: Model-based vertical auto-scaling for data stream processing," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 314–323.

[204] E. B. Lakew, A. V. Papadopoulos, M. Maggio, C. Klein, and E. Elmroth, "Kpi-agnostic control for fine-grained vertical elasticity," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 589–598.

[205] Y. Sfakianakis, M. Marazakis, C. Kozanitis, and A. Bilas, "Latest: Vertical elasticity for millisecond serverless execution," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 879–885.

[206] S. K. Tesfatsion, L. Tomás, and J. Tordsson, "Optibook: Optimal resource booking for energy-efficient datacenters," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE, 2017, pp. 1–10.

[207] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 486–494.

[208] M. Avgeris, D. Dechouniotis, N. Athanasopoulos, and S. Papavassiliou, "Adaptive resource allocation for computation offloading: A control-theoretic approach," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 2, pp. 1–20, 2019.

[209] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 1, pp. 1–21, 2018.

[210] L. Schuler, S. Jamil, and N. Kühl, "Ai-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 804–811.

[211] "F5 nginx management suite." [Online]. Available: https://www.nginx.com/

[212] "Haproxy:the reliable, high performance tcp/http load balancer." [Online]. Available: https://www.haproxy.org/

[213] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 523–535.

[214] T. Barbette, C. Tang, H. Yao, D. Kostic, G. Q. Maguire Jr, P. Papadimitratos, and M. Chiesa, "A high-speed load-balancer design with guaranteed per-connection-consistency." in *NSDI*, 2020, pp. 667–683.

[215] M. E. Elsaid, H. M. Abbas, and C. Meinel, "Virtual machines pre-copy live migration cost modeling and prediction: a survey," *Distributed and Parallel Databases*, vol. 40, no. 2, pp. 441–474, 2022.

[216] M. Noshy, A. Ibrahim, and H. A. Ali, "Optimization of live virtual machine migration in cloud computing: A survey and future directions," *Journal of Network and Computer Applications*, vol. 110, pp. 1–10, 2018.

[217] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2020–2033, 2018.

[218] "Xen project brings the power of virtualization everywhere." [Online]. Available: https://xenproject.org/

[219] "Open source container-based virtualization for linux." [Online]. Available: https://openvz.org/

[220] "A project to implement checkpoint/restore functionality for linux." [Online]. Available: criu.org

[221] G. Somma, C. Ayimba, P. Casari, S. P. Romano, and V. Mancuso, "When less is more: Core-restricted container provisioning for serverless computing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 1153–1159.

[222] N. Salhab, R. Rahim, and R. Langar, "Nfv orchestration platform for 5g over on-the-fly provisioned infrastructure," in *IEEE INFOCOM 2019-*

IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2019, pp. 971–972.

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

[224] "Horizontal pod autoscaler walkthrough." [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

[225] "Scaling and concurrency in lambda." [Online]. Available: https://docs.aws.amazon.com/lambda/latest/operatorguide/scaling-concurrency.html

[226] "an open-source enterprise-level solution to build serverless and event driven applications." [Online]. Available: https://knative.dev/docs/serving/autoscaling/kpa-specific/

[227] R. Birke, I. Giurgiu, L. Y. Chen, D. Wiesmann, and T. Engbersen, "Failure analysis of virtual and physical machines: patterns, causes and characteristics," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 1–12.

[228] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[229] V. N. Padmanabhan, S. Ramabhadran, S. Agarwal, and J. Padhye, "A study of end-to-end web access failures," in *Proceedings of the 2006 ACM CoNEXT conference*, 2006, pp. 1–13.

[230] C. Lu, K. Ye, G. Xu, C.-Z. Xu, and T. Bai, "Imbalance in the cloud: An analysis on alibaba cluster trace," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 2884–2892.

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

[232] D. Ardelean, A. Diwan, and C. Erdman, "Performance analysis of cloud applications," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 405–417.

[233] 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 Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 1–18, 2019.

[234] 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 Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 923–934, 2020.

[235] 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 Transactions on Computers*, vol. 65, no. 11, pp. 3502–3508, 2016.

[236] X. Sun, K. Chakrabarty, R. Huang, Y. Chen, B. Zhao, H. Cao, Y. Han, X. Liang, and L. Jiang, "System-level hardware failure prediction using deep learning," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[237] J. Xiao, Z. Xiong, S. Wu, Y. Yi, H. Jin, and K. Hu, "Disk failure prediction in data centers via online learning," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3225058.3225106

[238] X. Zeng, S. Garg, M. Barika, S. Bista, D. Puthal, A. Y. Zomaya, and R. Ranjan, "Detection of sla violation for big data analytics applications in cloud," *IEEE Transactions on Computers*, vol. 70, no. 5, pp. 746–758, 2020.

[239] X. Zhang, J. Kim, Q. Lin, K. Lim, S. O. Kanaujia, Y. Xu, K. Jamieson, A. Albarghouthi, S. Qin, M. J. Freedman *et al.*, "Cross-dataset time series anomaly detection for cloud systems," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 1063–1076.

[240] T. Pimentel, M. Monteiro, A. Veloso, and N. Ziviani, "Deep active learning for anomaly detection," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[241] M. Ma, S. Zhang, J. Chen, J. Xu, H. Li, Y. Lin, X. Nie, B. Zhou, Y. Wang, and D. Pei, "{Jump-Starting} multivariate time series anomaly detection for online service systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 413–426.

[242] H. Peng, R. Yang, Z. Wang, J. Li, L. He, S. Y. Philip, A. Y. Zomaya, and R. Ranjan, "Lime: Low-cost and incremental learning for dynamic heterogeneous information networks," *IEEE Transactions on Computers*, vol. 71, no. 3, pp. 628–642, 2021.

[243] J. Xu, Y. Wang, P. Chen, and P. Wang, "Lightweight and adaptive service api performance monitoring in highly dynamic cloud environment," in *2017 IEEE International Conference on Services Computing (SCC)*, 2017, pp. 35–43.

[244] L. Li, X. Zhang, X. Zhao, H. Zhang, Y. Kang, P. Zhao, B. Qiao, S. He, P. Lee, J. Sun *et al.*, "Fighting the fog of war: Automated incident detection for cloud systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 131–146.

[245] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 492–502.

[246] U. Demirbaga, Z. Wen, A. Noor, K. Mitra, K. Alwasel, S. Garg, A. Y. Zomaya, and R. Ranjan, "Autodiagn: An automated real-time diagnosis framework for big data systems," *IEEE Transactions on Computers*, vol. 71, no. 5, pp. 1035–1048, 2021.

[247] "Dx unified infrastructure management," [n.d.]. [Online]. Available: https://community.broadcom.com/enterprisesoftware/communities/communityhomeblogs?CommunityKey=170eb4e5-a593-4af2-ad1d-f7655e31513b

[248] "Ca unified infrastructure management," [n.d.]. [Online]. Available: http://aspiretp.com/uim/

[249] "Dtrace," [n.d.]. [Online]. Available: http://dtrace.org/blogs/about

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

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

[252] M. Hauswirth, P. F. Sweeney, A. Diwan, and M. Hind, "Vertical profiling: Understanding the behavior of object-priented applications," *SIGPLAN Not.*, vol. 39, no. 10, p. 251–269, oct 2004. [Online]. Available: https://doi.org/10.1145/1035292.1028998

[253] L. Zheng, J. Zhai, X. Tang, H. Wang, T. Yu, Y. Jin, S. L. Song, and W. Chen, "Vapro: Performance variance detection and diagnosis for production-run parallel applications," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 150–162. [Online]. Available: https://doi.org/10.1145/3503221.3508411

[254] P. Su, S. Jiao, M. Chabbi, and X. Liu, "Pinpointing performance inefficiencies via lightweight variance profiling," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–19.

[255] I. Laguna, D. H. Ahn, B. R. De Supinski, S. Bagchi, and T. Gamblin, "Diagnosis of performance faults in largescale mpi applications via probabilistic progress-dependence inference," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1280–1289, 2014.

[256] P. Dinda, "Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 2, pp. 160–173, 2006.

[257] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using arima model and its impact on cloud applications' qos," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015.

[258] G. K. Shyam and S. S. Manvi, "Virtual resource prediction in cloud environment: A bayesian approach," *Journal of Network and Computer Applications*, vol. 65, pp. 144–154, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804516300091

[259] P. Singh, P. Gupta, and K. Jyoti, "Tasm: technocrat arima and svr model for workload prediction of web applications in cloud," *Cluster Computing*, vol. 22, no. 2, pp. 619–633, 2019.

[260] W. Zhang, B. Li, D. Zhao, F. Gong, and Q. Lu, "Workload prediction for cloud cluster using a recurrent neural network," in *2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI)*, 2016, pp. 104–109.

[261] J. Kumar, R. Goomer, and A. K. Singh, "Long short term memory recurrent neural network (lstm-rnn) based workload forecasting model for cloud datacenters," *Procedia Computer Science*, vol. 125, pp. 676–682, 2018, the 6th International Conference on Smart Computing and Communications. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050917328557

[262] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 11 106–11 115.

[263] G. Hamerly, C. Elkan *et al.*, "Bayesian approaches to failure prediction for disk drives," in *ICML*, vol. 1, no. 2001, 2001, pp. 202–209.

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

[265] 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." *Journal of Machine Learning Research*, vol. 6, no. 5, 2005.

[266] S. J. Russell, *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[267] S. Roy, A. C. Konig, I. Dvorkin, and M. Kumar, "Perfaugur: Robust diagnostics for performance anomalies in cloud services," in *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 2015, pp. 1167–1178.

[268] T. Chalermarrewong, T. Achalakul, and S. C. W. See, "Failure prediction of data centers using time series and fault tree analysis," in *2012 IEEE 18th International Conference on Parallel and Distributed Systems*. IEEE, 2012, pp. 794–799.

[269] M. Godse, U. Bellur, and R. Sonar, "Automating qos based service selection," in *2010 IEEE International Conference on Web Services*. IEEE, 2010, pp. 534–541.

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

[271] S. Fu and C.-Z. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, 2007, pp. 1–12.

[272] B. Cavallo, M. Di Penta, and G. Canfora, "An empirical comparison of methods to support qos-aware service selection," in *Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems*, 2010, pp. 64–70.

[273] A. Amin, A. Colman, and L. Grunske, "An approach to forecasting qos attributes of web services based on arima and garch models," in *2012 IEEE 19th International Conference on Web Services*, 2012, pp. 74–81.

[274] X. Chen, C.-D. Lu, and K. Pattabiraman, "Failure prediction of jobs in compute clouds: A google cluster case study," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. IEEE, 2014, pp. 341–346.

[275] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

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

[277] S. Mohseni, M. Pitale, J. Yadawa, and Z. Wang, "Self-supervised learning for generalizable out-of-distribution detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5216–5223.

[278] J. Sun, L. Yang, J. Zhang, F. Liu, M. Halappanavar, D. Fan, and Y. Cao, "Gradient-based novelty detection boosted by self-supervised binary classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8370–8377.

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

[280] 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 Recognition*, vol. 58, pp. 121–134, 2016.

[281] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 1409–1416.

[282] 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 *International Conference on Machine Learning*. PMLR, 2022, pp. 3621–3633.

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

[284] G. Michau and O. Fink, "Unsupervised transfer learning for anomaly detection: Application to complementary operating condition transfer," *Knowledge-Based Systems*, vol. 216, p. 106816, 2021.

[285] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 9428–9433.

[286] 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 Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 548–561, 2020.

[287] L. Shen, Z. Yu, Q. Ma, and J. T. Kwok, "Time series anomaly detection with multiresolution ensemble decoding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 11, 2021, pp. 9567–9575.

[288] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 187–196.

[289] 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 *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2828–2837.

[290] J. Sipple, "Interpretable, multidimensional, multimodal anomaly detection with negative sampling for detection of device failure," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9016–9025.

[291] "Prometheus - monitoring system & time series database," [n.d.]. [Online]. Available: https://prometheus.io/

[292] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "Racs: a case for cloud storage diversity," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 229–240.

[293] R. Kang, M. Zhu, F. He, and E. Oki, "Implementation of virtual network function allocation with diversity and redundancy in kubernetes," in *2021 IFIP Networking Conference (IFIP Networking)*. IEEE, 2021, pp. 1–2.

[294] M. Uluyol, A. Huang, A. Goel, M. Chowdhury, and H. V. Madhyastha, "{Near-Optimal} latency versus cost tradeoffs in {Geo-Distributed} storage," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 157–180.

[295] Y. Aldwyan and R. O. Sinnott, "Latency-aware failover strategies for containerized web applications in distributed clouds," *Future Generation Computer Systems*, vol. 101, pp. 1081–1095, 2019.

[296] H. Jin, G. Yang, B.-y. Yu, and C. Yoo, "Fave: Bandwidth-aware failover in virtualized sdn for clouds," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 505–507.

[297] R. Landa, L. Saino, L. Buytenhek, and J. T. Araújo, "Staying alive: Connection path reselection at the edge," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 233–251.

[298] I. Giannakopoulos, I. Konstantinou, D. Tsoumakos, and N. Koziris, "Cloud application deployment with transient failure recovery," *Journal of Cloud Computing*, vol. 7, no. 1, pp. 1–20, 2018.

Shuiguang Deng is currently a full professor at the College of Computer Science and Te chnology in Zhejiang University, China, where he received a BS and PhD degree both in Computer Science in 2002 and 2007, respectively. He previously worked at the Massachusetts Institute of Technology in 2014 and Stanford University in 2015 as a visiting scholar. His research interests include Edge Computing, Service Computing, Cloud Computing, and Business Process Management. He serves for the journal IEEE Trans. on Services Computing, Knowledge and Information Systems, Computing, and IET Cyber-Physical Systems: Theory & Applications as an Associate Editor. Up to now, he has published more than 100 papers in journals and refereed conferences. In 2018, he was granted the Rising Star Award by IEEE TCSVC. He is a fellow of IET and a senior member of IEEE.

Hailiang Zhao received the B.S. degree in 2019 from the school of computer science and technology, Wuhan University of Technology, Wuhan, China. He is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research interests include cloud & edge computing, distributed computing and optimization algorithms. He has published several papers in flagship conferences and journals including IEEE ICWS 2019, IEEE TPDS, IEEE TMC, etc. He has been a recipient of the Best Student Paper Award of IEEE ICWS 2019. He is a reviewer for IEEE TSC and Internet of Things Journal.

Binbin Huang is an Assistant Professor in the College of Computer Science at the University of Hangzhou Dianzi, in Hangzhou, China. She received her PhD degree in Computer Science and Technology from Beijing University of Posts and Telecommunications in 2014. His research interests include cloud computing, mobile edge computing, and reinforcement learning.

Cheng Zhang received the MS degree in electrical engineering from Zhejiang University, China, in 2013, Currently, he is working toward the PhD degree in computer science and technology at Zhejiang University. His research interests include edge computing and edge intelligence.

Feiyi Chen received the B.S. degree in 2021 from the school of computer science and engineering, Sun Yat-sen University (SYSU), Guangzhou, China. She is currently pursuing the master degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. Her research interests include cloud computing, edge computing, and distributed systems.

Yinuo Deng received the B.S. degree in 2022 from the School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing, China. He is currently a M.S. student in computer science of technology at College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research interests include cloud computing, networking, and distributed systems.

Jianwei Yin received the Ph.D. degree in computer science from Zhejiang University (ZJU) in 2001. He was a Visiting Scholar with the Georgia Institute of Technology. He is currently a Full Professor with the College of Computer Science, ZJU. Up to now, he has published more than 100 papers in top international journals and conferences. His current research interests include service computing and business process management. He is an Associate Editor of the IEEE Transactions on Services Computing.

Schahram Dustdar is a Full Professor of Computer Science (Informatics) with a focus on Internet Technologies heading the Distributed Systems Group at the TU Wien. He was founding co-Editor-in-Chief of ACM Transactions on Internet of Things (ACM TIoT). He is 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, as well as on the editorial board of IEEE Internet Computing and IEEE Computer. Dustdar is recipient of multiple awards: TCI Distinguished Service Award (2021), IEEE TCSVC Outstanding Leadership Award (2018), IEEE TCSC Award for Excellence in Scalable Computing (2019), ACM Distinguished Scientist (2009), ACM Distinguished Speaker (2021), IBM Faculty Award (2012). He is an elected member of the Academia Europaea: The Academy of Europe, where the chairman of the Informatics Section for multiple years. He is an IEEE Fellow (2016), an Asia-Pacific Artificial Intelligence Association (AAIA) President (2021) and Fellow (2021). He is an EAI Fellow (2021) and an I2CICC Fellow (2021). He is a Member of the IEEE Computer Society Fellow Evaluating Committee (2022 and 2023).

Albert Y. Zomaya is the Peter Nicol Russell Chair Professor of Computer Science and Director of the Centre for Distributed and High-Performance Computing at the University of Sydney. To date, he has published > 600 scientific papers and articles and is (co-)author/editor of > 30 books. A sought-after speaker, he has delivered > 250 keynote addresses, invited seminars, and media briefings. His research interests span several areas in parallel and distributed computing and complex systems. He is currently the Editor in Chief of the ACM Computing Surveys and processed in the past as Editor in Chief of the IEEE Transactions on Computers (2010-2014) and the IEEE Transactions on Sustainable Computing (2016-2020). Professor Zomaya is a decorated scholar with numerous accolades including Fellowship of the IEEE, the American Association for the Advancement of Science, and the Institution of Engineering and Technology (UK). Also, he is an Elected Fellow of the Royal Society of New South Wales and an Elected Foreign Member of Academia Europaea. He is the 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 (2011), IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing (2011), IEEE Computer Society Technical Achievement Award (2014), ACM MSWIM Reginald A. Fessenden Award (2017), the New South Wales Premier's Prize of Excellence in Engineering and Information and Communications Technology (2019), and the Research Innovation Award, IEEE Technical Committee on Cloud Computing (2021).