Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

A multi-dimensional extensible cloud-native service stack for enterprises

Jian Lin*, Dongming Xie, Jinjun Huang, Zinan Liao and Long Ye

Abstract

With the widespread acceptance of the cloud-native concept and the emergence of a large number of dedicated cloud-native applications, the service stacks of cloud-native applications have received extensive attention in the industry. To analyze the extensibility problems of service stacks, a cloud-native light-cone model is proposed, which focuses on the dimensions of application, infrastructure, tenant and workflow, and provides a perspective view that reflects the concerns of stakeholders. Based on this model, various challenges in designing extensible cloud-native service stacks are identified by classification. To solve these challenges, a holistic architecture and a set of key technologies are designed, involving unified runtime abstraction, cluster bootstrapped creation, application-specific controllers, etc. Furthermore, the OMStack (Oriental Mind Stack) is implemented, which integrates these technologies and provides a group of PaaS and SaaS services for container cluster (OMCC), artificial intelligence (OMAI), big data (OMBD) and so on. Experimental analysis and production applications demonstrate the practicality, efficiency and reliability of the proposed architecture, stack and services.

Keywords: Cloud-native, Container cluster, Service stack, Kubernetes, PaaS, SaaS

Introduction

In recent years, the practices of cloud-native computing have been widely accepted by cloud service providers and information technology enterprises. On the basis of the original capabilities of cloud computing like resource pooling, on-demand provisioning and elastic scaling [1, 2], the concept of cloud-native computing further emphasizes that the full stack of systems and the full life-cycle of applications are naturally designed for cloud environments. It aims to provide automation, resiliency, manageability and observability to the users [3].

Many dedicated cloud-native applications are developed to serve different businesses. Several cloud-native service stacks [4–6] are also proposed to provide orchestration and maintenance capabilities for upper-layer applications. Compared to earlier cloud computing practices of running traditional applications on virtualized infrastructures, the cloud-native practices deliver higher resource utilization and enable more business flexibility. Thus, it gets particular attention from enterprises with on-premise clusters.

As can be seen from the history of cloud-native computing, its typical architectures and technologies are abstracted and summarized from best practices, rather than deduced from pure theories. This is reasonable in computer engineering, but it is also worthwhile to consider a more systematic approach to analyzing and building cloud-native service stacks. In this way, the research challenges can be analyzed in a structured manner, and the service software can be designed in a consolidated manner. This is the direction this paper hopes to explore.

To realize the value of the systematic approach, it is necessary to build production systems and serve real businesses. Instead of large and comprehensive service stacks for cloud service providers, the cloud-native service stack for enterprises is the focus of this paper, while mainstream applications including artificial intelligence and big data are considered as the primary workloads. To generalize the capabilities in many dimensions that the

*Correspondence: linjian@orientalmind.com

Oriental Mind (Wuhan) Computing Technology Co., Ltd., Wuhan, China

service stack needs to achieve, the "extensibility" feature is highlighted.

The contributions of this paper include:

(1) A cloud-native light-cone model is proposed to characterize the issues of extensibility in cloud-native service stacks. Multiple dimensions including application, infrastructure, tenant and workflow are covered. Based on this model and its perspective view, challenges in designing extensible cloud-native service stacks are identified and analyzed.

(2) A multi-dimensional extensible architecture for cloud-native service stacks is proposed, and a set of key technologies are designed to solve the challenges of multiple aspects. The main technologies include unified runtime abstraction, cluster bootstrapped creation, application-specific controllers, etc.

(3) An implementation of the architecture – OMStack (Oriental Mind Stack) is developed, which provides a group of PaaS and SaaS services for container cluster, artificial intelligence, big data and so on. Its practicality, efficiency and reliability have been proven in experimental analysis and production applications.

Background

The field of cloud-native computing has gradually formed a recognized technical system. The dimensional analysis and modeling research on the field is also emerging.

Cloud-native technical stacks

Cloud-native technical stacks are the foundation for building service capabilities. Despite the lack of standardized definitions, the main components and basic characteristics of the stacks have been identified and accepted by the practices of the industry. Three main perspectives are worth noting.

In terms of workload abstraction, the serverless execution model [7] is emphasized. It means the developers of applications only need to pay attention to the business logic, not the capacity planning, allocation or maintenance of the underlying resources. In the technical implementation, FaaS (Function as a Service) and CaaS (Container as a Service) are two major routes. A FaaS service hosts users' application functionalities and allocates scalable resources for them at fine granularity. Its disadvantage is that the application needs to be rewritten and the functionality is constrained. A CaaS service encapsulates and runs applications within containers. It has better compatibility with legacy business, but the scalability, efficiency and resiliency are also subject to the existing applications.

In terms of runtime management and resource scheduling, containerization is the main technical route. Compared to hypervisor-based virtualization technologies used by IaaS stacks, containerization technology is more lightweight and efficient. In the cloud-native context, service stacks need not only container engines for a single node, but also container orchestrators that manage the compute, storage and network resources for a cluster. Kubernetes [4] is such a widely used container orchestrator. Containerization technology also meets the needs of the microservice architecture for functional decomposition, service discovery and runtime management, so it is friendly to existing enterprise applications.

In terms of software engineering paradigm, the concepts of CI/CD (continuous integration, continuous delivery) and DevOps (combining software development and IT operations) are often emphasized by cloud-native practices. Its purpose is to improve the development efficiency and product quality of cloud-native services. Under this paradigm, automated workflows are leveraged to drive the building, verification and deployment of applications. The boundary between development and production environments becomes blurred, while automated traffic policies implement the iterations of services in a real-time and secure manner. Saving operation and maintenance costs is an important consideration for enterprises adopting this paradigm.

Dimensional analysis models

Architecture modeling is a systematic way of analyzing existing systems and designing new systems. Among the modeling methodologies, dimensional analysis is a concise and effective way.

In computer networking, the hourglass model [8] is a classic model that uses the dimensions of hierarchy and function to describe the relationship between protocols. The bridging role of the network layer (Internet protocol) is an important insight from the model. This model has also been extended to related fields such as grid computing [9]. Inspired by the hourglass model, Lin et al. proposed an analysis model for consolidated cluster systems [10] that adopts the dimensions of resource consolidation and runtime coordination. This model defines a set of key features on each dimension in order to locate the deficiencies in existing systems, and guide the improvement and innovation of new systems.

In software engineering, the 4+1 view model [11] provides general guidance for software architecture design and implementation. It uses the views of logical, process, development, physical and scenarios to clarify the concerns in each phase of a software life-cycle. Based on this model, many formal methods like UML diagrams are available for each view [12], so the conceptual model can further guide engineering practice, involving systematically analyzing existing software and automatically generating new code. Inspired by

this model, derived models [13, 14] have also been proposed in some subfields of software engineering.

In cloud computing, dimensional analysis is widely adopted by academia and industry. Among the models, the conceptual reference model proposed by NIST [15] as well as the taxonomy on multiple dimensions has universal guiding significance. It organizes the concepts of the taxonomy in four levels: role, activity, component and sub-component. Typical service deployment and consumption modes are defined by the model. This model focuses on external functional views instead of internal technical details. Other models and taxonomies [16] usually extend the reference model in depth (underlying implementation) or breadth (subdivision components). Refined formal specifications like TOSCA (Topology and Orchestration Specification for Cloud Applications) [17] are also proposed for describing and constructing cloud applications.

Specific to the field of cloud-native computing, there are relatively few studies on analysis models. Kratzke et al. proposed the ClouNS reference model [18] and analyzed a group of cloud-native applications, architectures and methodologies [19] according to the model. The main dimensions the model focuses on are infrastructure viewpoint and service category. Through hierarchical refinement, the underlying implementations of different applications and service stacks are classified. Thus, their design trade-offs can be identified, and potential innovation opportunities can be found. Besides, the maturity assessment model [3] for cloud-native service stacks is also concerned by enterprise users.

Cloud-native light-cone model

In order to characterize the issues in designing cloud-native service stacks, and analyze the technical challenges in a systematic way, a new model – the cloud-native light-cone model is proposed.

Definition

The cloud-native light-cone model is a dimensional analysis model. It focuses on describing and guiding the designs of cloud-native service stacks. It defines the dimensions of analysis from a spatiotemporal perspective in software engineering. Four dimensions including application, infrastructure, tenant and workflow are concerned by this model. As shown in Fig. 1, these dimensions form a 4D coordinate system that looks like a light cone in the theory of relativity. The axis scales qualitatively represent the design options for each dimension, while the points in the space represent the product orientations or design decisions of cloud-native services and service stacks.

This model is inspired by the end-to-end idea of the hourglass model but with obvious extensions. The application and infrastructure dimensions are the top and bottom ends of a technical stack, while the tenant and workflow dimensions are the spatial and temporal organization units of a business pipeline. The qualitative scales of each dimension are not constrained by this model, but can be chosen flexibly by users. A typical group of scales and their instances in existing service stacks are listed in Table 1. The object indexed by a larger number approximately contains the object indexed by a smaller number in each dimension.

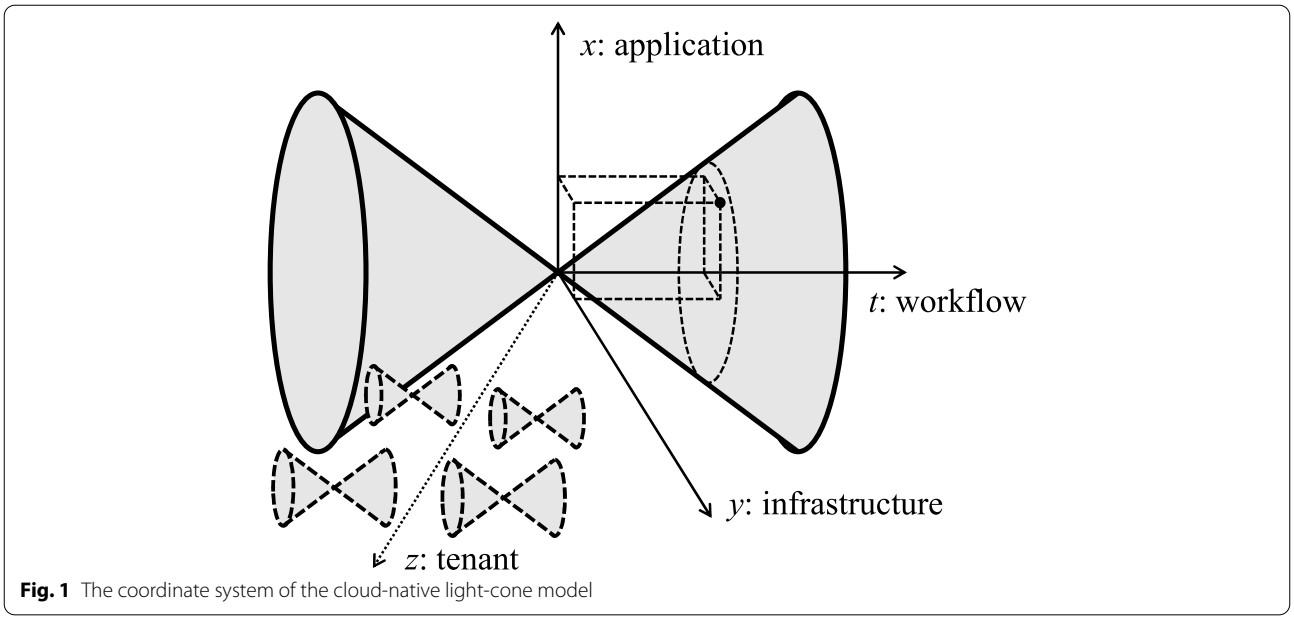

Fig. 1 The coordinate system of the cloud-native light-cone model

Table 1 The dimensions of the cloud-native light-cone model

Dimension	Example of scale		
	Index	Object	Instance
x: application	A1	process (task)	a mapper task of a Spark MR job
	A2	process group (job)	a Spark MR job
	A3	application framework	a group of Spark runtime services
	A4	application cluster	a Kubernetes cluster with Spark-operator
y: infrastructure	I1	device	a GPU
	I2	node	a VM with a GPU
	I3	IaaS stack instance	an OpenStack cluster
	I4	IaaS source	a public cloud
z: tenant	T1	user	a personal (sub)account on a public cloud
	T2	user group	a user group on a public cloud
	T3	virtual organization	a project on a public cloud
	T4	organization	an enterprise account on a public cloud
t: workflow	W1	intra-job	an Allreduce-based AI training
	W2	inter-job	an ensemble-learning-based AI training
	W3	inter-service	an AI pipeline from training to inference
	W4	inter-cluster	an AI pipeline for CI/CD

This model also provides a perspective view that reflects the concerns of different stakeholders. As shown in Table 2, pairwise combinations of the four dimensions constitute six kinds of concerns. They are associated with six typical roles in three groups of stakeholders (end-user, proprietor and developer) of a service stack. Each role's responsibility related to the four dimensions is explained in the table. It illustrates the guidance value of this model for the design, application and evaluation of cloud-native service stacks.

Properties

Extensibility is the main property of concern that can be derived directly from the structural features of the cloud-native light-cone model. The specific meanings of extensibility are explained as follows.

(1) Extensibility in depth: it means the whole or part of a service stack corresponding to a point in the coordinate system can support extensive types of instances, where the instances belong to the objects constrained by the multi-dimensional scales of the point. This property reflects the service stack's functional strength at a single point.

(2) Extensibility in breadth: it means the whole or part of a service stack corresponding to a point in the coordinate system has the potential to extend its range of capabilities to those defined by the nearby points in the coordinate system. This property reflects the service stack's capability coverage and application scope.

Although this model does not reflect other properties of a service stack directly, they can be further derived from the functions produced by the combination of

Table 2 The perspective view of the cloud-native light-cone model

Stakeholder		Concern	Explanation
Group	Role		
service stack end-user	manager	$x + z$	manage and coordinate various business resources, involving things (application) and people (tenant)
	ordinary user	$x + t$	organize individual work (application) uniformly and effectively in a structured way (workflow)
service stack proprietor	service operator	$y + z$	use the service stack on the infrastructure to serve the users (tenant)
	system maintainer	$y + t$	ensure the stability and availability of the business (workflow) on the infrastructure
service stack developer	software architect	$x + y$	design effective abstractions to map applications to underlying resources (infrastructure)
	product manager	$z + t$	design business processes (workflow) and user experience for tenants to improve efficiency and usability

dimensions. The following sections on architecture and designs will explain them.

Discussion

To better understand and apply the cloud-native light-cone model, it is necessary to explain how it relates to classic models and existing formal methods.

The original intention and positioning of this model are similar to the 4+1 view model, that is, to propose a macroscopic model for guiding system analysis and design. It evolves rather than replaces the classic model. On the one hand, it spotlights the scenario of cloud-native services. A specific target field makes the selection of dimensions more focused and relevant. On the other hand, while referring to the view division based on various stakeholders, this model places more emphasis on the spatiotemporal dimensions closely related to the runtime of cloud services, so as to better serve the effective output of enterprises.

The functionality of this model is a complement to classic formal methods like UML and TOSCA. It extends existing methodologies rather than conflicts with them. UML and TOSCA focus more on implementability. They pursue a consistent mapping from logical design to physical implementation. In contrast, our model focuses more on comprehensibility. It is a thinking framework that mainly serves the upstream of a software life-cycle for high-level problem analysis and architecture design.

Challenges of extensible cloud-native service stack

By using the cloud-native light-cone model, the challenges of designing an extensible cloud-native service stack are analyzed. To make the research problems more focused, this paper mainly studies the requirements of mainstream application scenarios in modern enterprises, involving artificial intelligence, big data, etc. All challenges are classified according to the dimensions of the model. The challenges in dimension x and y are mainly related to extensibility in depth, while those in dimension z and t are mainly related to extensibility in breadth.

Extensibility dimension *x*: diverse application modes

The key to enabling extensibility in the application dimension is to support diverse application modes. Its main challenges include diverse scheduling modes and runtime environment dependency.

Diverse scheduling modes

Mainstream enterprise applications usually require two main scheduling modes: microservice and batch job. For each application framework, there are also diverse scheduling (sub)modes for specific workload types. For the microservice mode, service stacks need to deal with issues including service discovery, traffic routing and high availability. For the batch job mode, service stacks need to support the schematized behaviors of scheduling stateful runtime among tasks such as gang-scheduling. Container-based service stacks are microservice friendly, but they generally do not provide deep adaptation for batch jobs. Although open-source adaptors like Kubeflow [20] and Volcano [21] are available, they provide either application-specific high-level abstraction or fine-grained widgets. A universal batch job management mechanism and a universal abstraction for diverse scheduling modes are important for an extensible stack.

Runtime environment dependency

An application's runtime environment dependency involves hardware and software. On the hardware side, artificial intelligence and big data applications often require compute accelerators (e.g. GPU, FPGA) and high-speed network cards (e.g. InfiniBand, RoCE). However, not all container orchestrators provide native abstractions to manage these special devices. Especially when dealing with diverse models and specs, simple resource matching mechanisms like label selector appears to be insufficient. On the software side, although the container technology aims at solving the problem of dependencies in a self-contained way, some complicated cases are not covered. For example, the versions of CUDA and OFED libraries are coupled with the versions of the underlying GPU and InfiniBand drivers. A service stack needs to decide whether to put the libraries inside or outside containers, and how to make sure the versions are compatible with those of the drivers on hosts.

Extensibility dimension *y*: heterogeneous infrastructures

Extensibility issues related to the infrastructure dimension mainly stem from heterogeneity. The resources of compute, storage and network have their own distinct challenges.

Heterogeneous IaaS

A container-based service stack usually uses bare-metal or virtual-machine-based infrastructure as its host environment. To provide a consistent execution environment for upper-layer applications, some of the software-level heterogeneity issues should be handled and hidden by the service stack. For example, the IaaS APIs are different among cloud providers, so tailored adaptation is necessary to enable dynamic resource provisioning and multi-cloud resource access. A typical issue in this aspect is the mechanism of public network accessibility and load balancing, in which the implementation is subject to complicated factors like network address translation and security policy. Conversely, some hardware-level

heterogeneity issues such as the instruction set architecture are often perceived by the upper-layer application developers, especially for an environment combining clouds and edges. In this case, a service stack should provide architecture-aware orchestration mechanisms.

Storage locality

The separation architecture of compute and storage [22] is popular in cloud-native environments. In this architecture, independent storage clusters are built using generic underlying storage systems like the object storage, while multiple compatible interfaces like the HDFS API are exported to compute clusters. This architecture is friendly to dynamic, elastic and stateless containerized compute workloads. However, for small/medium-scale on-premise clusters in enterprises, the separation architecture has potential disadvantages including the high sensitivity of network performance, the low utilization of storage resources, and the complexity of construction and maintenance. It is worthwhile to introduce classic local storage mode into containerized environments to solve these problems. The main challenges involve the dynamic allocation and reclamation of persistent storage space, the performance optimization of disk and network access, and the affinity between compute and storage containers.

Virtualization of high-performance hardware

Multiplexing of high-performance hardware like GPU and InfiniBand is important for cost savings. Although some devices provide native multiplexing or virtualization mechanisms, potential problems remain with their applications in cloud-native environments. Firstly, device plugins of container orchestrators are required to enable virtual devices in container clusters, but the ecosystem of device plugins is immature. For example, the existing InfiniBand plugin cannot constrain RDMA traffics on virtual devices [23]. Secondly, the configuration for using high-performance hardware in applications is generally complicated, while virtualization introduces further cumbersome factors. For example, a virtual function of a RoCE device has its additional index, which results in poor environmental portability for applications in containers. Besides, the security isolation and performance isolation among workloads sharing the same device are relatively weak due to the limitation of the container mechanism.

Extensibility dimension *z*: multi-tenant on-demand clusters

In the tenant dimension, the extensibility issues are reflected in the provisioning of independent virtual execution environments on-demand for multiple tenants in a secure and efficient manner.

Framework on-demand provisioning

On-demand provisioning is an important feature of cloud computing. Even in a single enterprise, users are always making continuous requests for resources in accordance with diversified business needs. According to the application dimension, cloud-native service stacks can provide on-demand resources for tenants at different scales: process group (job), application framework or application cluster. Application-specific stacks (e.g. a Kubeflow on Kubernetes cluster for artificial intelligence) usually only support the job-scale provisioning. When the tenant dimension is introduced, on-demand provisioning at the application framework or application cluster scale becomes necessary because it can provide fully controllable virtual execution environments for different tenants. Fine-grained resource allocation and access control for the users and groups inside a tenant (organization) are feasible in the virtual execution environments in a framework-native way. The challenge is how to implement it for diverse frameworks with different characteristics and requirements.

Security isolation

Security is a fundamental requirement in production systems, which involves factors such as runtime isolation, data privacy and traffic policy. In an enterprise-oriented multi-tenant environment, flexible security strategies can be adopted at different layers. The isolation between tenants (organizations) usually needs to be mandatory and physical, while that between users inside a tenant is usually just optional or logical. At the container level, achieving strong isolation by using the weak isolation techniques of containerization is a challenge. At the container cluster level, the multi-tenancy feature is not available at a product level in mainstream orchestrators [24]. Thus, cloud-native service stacks need to fill these gaps. The designs should balance security with additional overhead, resource utilization and application transparency.

Resource utilization

The percentage of resources used by effective workloads is an important indicator of a software stack, especially for that in an on-premise cluster. Multi-tenancy introduces additional issues on resource utilization. Typical issues are as follows. Firstly, the isolation of virtual execution environments among tenants limits the elastic scheduling of free resources, while the per-tenant components also take up more resources than shared components. Secondly, when introducing the separation architecture of compute and storage in small/

medium-scale clusters, the storage utilization may be relatively low compared with that in the traditional coupling architecture. Thirdly, to fully utilize expansive high-performance hardware, the trade-off between multiplexing and security is inevitable. A cloud-native service stack should deal with the above problems in a systematic way.

Extensibility dimension *t*: automated workflows

The workflow dimension focuses on achieving connectivity and automation across the entire life-cycle of business on the cloud. This dimension is the embodiment of the cloud-native software engineering paradigm.

Inter-job workflow

Batch job schedulers generally handle intra-job workflows, while common workflow engines in the cloud-native ecosystem generally handle workflows over containers or pods rather than batch jobs. For artificial intelligence and big data applications, higher-level workflows are often required to meet the needs of comprehensive businesses. For example, an artificial intelligence workflow involves model development, training, serving and verification, where different steps have diverse scheduling and interaction modes. To handle this kind of business, the workflow mechanism needs to be extensible for both universality (application adaptability) and scalability (task scale). Meanwhile, usability is another value that workflow mechanisms can bring. Application-specific workflow plans and routine steps can be made implicit in the upper layers of a service stack, while the lower layers are responsible for generic capabilities.

CI/CD workflow

When the concepts of continuous integration and continuous delivery are introduced in a service stack, the workflow designs will expand at multiple angles. In terms of environmental coverage, a CI/CD workflow may cover multiple clusters involving development and production environments. In terms of objects to be managed, a CI/CD workflow deals with not only executable tasks, but also entities like data assets and software artifacts. Although the general CI/CD toolchains in software engineering are relatively mature, adaptive designs are required when applying them in specific application scenarios. For example, a CI/CD workflow for artificial intelligence needs to manage the logics of models' compilation for specific inference devices, encapsulation for specific runtime environments, documentation of meta-information, etc. Besides, the programmable mechanism is helpful for a CI/CD workflow, which enhances the configurability of complicated automation pipelines.

Multi-dimensional extensible architecture

Guided by the cloud-native light-cone model, a multi-dimensional extensible architecture is designed. It focuses on addressing the challenges of implementing extensible cloud-native service stacks. The composition of this architecture and the relationship among main components are shown in Fig. 2. Intuitively, it inherits and extends the idea of the hourglass model. The architecture defines the service stack as a bridging layer logically. It emphasizes that the service components should treat the applications and infrastructures in a hierarchical view like the protocol stack of computer networking. The introduction of tenant and workflow emphasizes two meanings: (1) The virtual execution environments for tenants define the spatial division of a service stack, while the orchestration and scheduling of workflows define the temporal organization of a service stack. (2) The workflows carry the inputs of business logic, while the tenants receive the outputs of business logic.

Referring to the perspective view in Table 2, this architecture also embodies six main functions of a service stack by connecting four dimensions:

(1) $x + z$: business management. Enterprise tenants typically have business management policies consistent with their administrative regulations. The service stack should provide flexible mechanisms to allow tenants to implement their policies.

(2) $x + t$: runtime abstraction. The business logic carried by workflows will be abstracted into application runtime entities of appropriate granularities so that either the service stack or the users can manage the business logic in a uniform manner respectively.

(3) $y + z$: environment allocation. The infrastructures will be partitioned or packaged as multiple virtual execution environments to serve different tenants or businesses. The service stack ensures a tenant-specific resource view.

(4) $y + t$: resource maintenance. The infrastructure as the critical basis needs to be operated and maintained effectively. The service stack should meet the proprietor's need for stability and availability to host workflows in production.

(5) $x + y$: architecture adaption. The infrastructures will be utilized in an organized way according to the applications' requirements. Architecture adaptation interfaces between hardware and software at the appropriate level will be implemented by the service stack.

(6) $z + t$: logic orchestration. The workflows stand for the business processes requested by tenants, which are scheduled by the service stack to ensure that the orchestration logic meets the tenants' expectations as in the physical world.

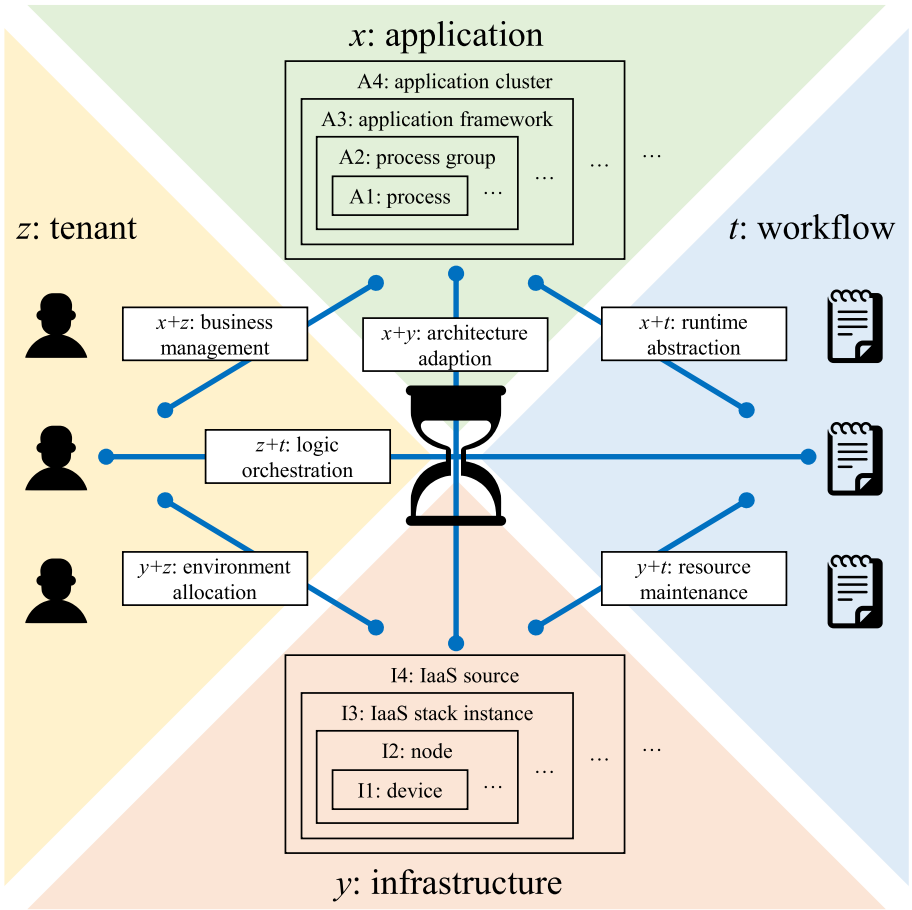

Fig. 2 The composition of the multi-dimensional extensible architecture

Designs of key technologies

Based on the multi-dimensional extensible architecture, a group of key technologies are designed. They aim to solve the concrete challenges so that an extensible cloud-native service stack for enterprises can be implemented systematically. Representative designs are introduced in this section. The correspondence between the designs of key technologies and the challenges they address is presented in Fig. 3.

Unified runtime abstraction

The unified runtime abstraction [25] is designed for job management. It includes two layers: the universal interface and extensible drivers. The universal interface is an abstract data structure describing the runtime entities as well as their resource requirements and scheduling policies of a job. It supports either using the fine-grained "process group" structure to describe any job in a free way or using the templated "application job" structure to describe jobs of classic application frameworks in a concise way. Taking the artificial intelligence job as an example, the templated structure abstracts typical distribution modes like PS-Worker, Allreduce and MPI as standard options of scheduling policy. The universal interface can be expressed and stored in JSON.

Extensible drivers are employed to support diverse application modes. For the microservice mode, the driver is implemented using the mechanisms of native Kubernetes and Istio [26]. For the batch job mode, the generic driver is implemented by synthesizing the mechanisms of Volcano, the network and affinity policies of Kubernetes, as well as several improved designs implementing the schematized scheduling behaviors. Some of the specific drivers for classic application frameworks are implemented using the operator mechanism of Kubernetes.

To solve the runtime environment dependency problem, the runtime abstract provides fields for environment matching. To support special devices with quantitative specifications (e.g. GPU's model and memory), and also to match the versions of drivers on hosts, a resource expression matching mechanism is designed to extend the simple label selector mechanism. It implements

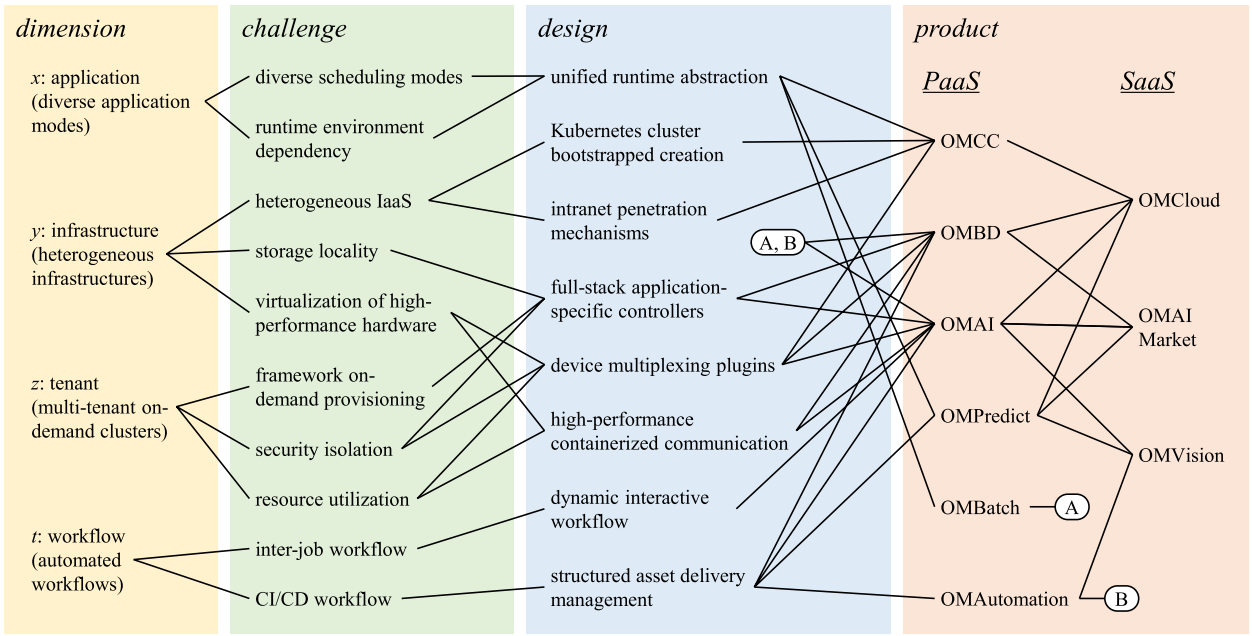

Fig. 3 The designs of key technologies and the challenges they address

arithmetic, string and set operations, so complicated matching conditions can be expressed.

Kubernetes cluster bootstrapped creation

The Kubernetes cluster bootstrapped creation mechanism [27] is designed for the life-cycle management of containerized application clusters. It can create independent Kubernetes clusters dynamically for different tenants. The mechanism includes the universal abstraction and extensible bootstrapped drivers. The universal abstraction describes the requirements of a Kubernetes cluster. Advanced configuration items involving network and storage infrastructures are supported.

Extensible bootstrapped drivers are responsible for creating Kubernetes clusters on bare-metal- or virtual-machine-based IaaS clusters. Two environments are related to a driver: the host environment where the driver runs, and the target environment where the new Kubernetes cluster is created. "Bootstrap" means that a driver is a self-contained job in a container running in the host environment. The self-contained job depends on only the standard IaaS APIs of the target environment, but not any other components of it. The implementation of drivers leverages some IaC (Infrastructure as Code) toolchains like Terraform [28]. The complexity of programming in the IaC DSL (domain-specific language) is hidden in the drivers, while the support for high-performance devices is extended in specially developed plugins.

Support for multiple heterogeneous cloud environments demonstrates the extensibility of this mechanism. The design of extensible providers in the IaC toolchain allows the target environment to be set on either a private cloud based on OpenStack with Magnum, or IaaS clusters of mainstream public clouds. This enables a cloud-native service stack to support hybrid cloud and multi-cloud environments [29] for performance and availability.

Intranet penetration mechanisms

To solve the network accessibility issues in complicated network environments, a group of intranet penetration mechanisms are employed. Traditionally, VPN (virtual private network) and cloud load balancer are common mechanisms for accessing the intranets inside containerized clusters from the Internet. They offer the benefits of reliability and performance, but they are subject to the capabilities of cloud providers and are costly. Two new flexible mechanisms are designed. One is the container-cluster-oriented reverse proxy mechanism, and another is the MQTT (message queuing telemetry transport)-based message broker. They require only the virtual IP service from cloud providers, and provide lightweight channels for intranet penetration at a low cost.

The container-cluster-oriented reverse proxy mechanism includes a load gateway and a group of Istio services. The load gateway works as the entry of traffic which supports dynamic virtual IP binding, while the Istio

services are responsible for the routing of traffic within the intranet. The MQTT-based message broker contains peer services in different network environments, which are designed for transmitting control messages on unreliable connections. Typical application scenarios of these mechanisms include: implementing a hybrid cloud where the cloud-native service is in a private cluster but the target environments are on public clouds, implementing an application across clouds and edges in which the edge sides have their own private networks, etc.

Full-stack application-specific controllers

The full-stack application-specific controller is a comprehensive technology to implement on-demand provisioning of application frameworks or application clusters. Compared with common Kubernetes operators [30], the concept of "full-stack" emphasizes that the technology covers not only the runtime entities of frameworks/clusters, but also the underlying resources including storage and network.

A typical example of these controllers is the HDFS-on-Kubernetes provisioner. It is a full-stack operator that aims to provision multiple on-demand storage clusters, and also addresses the challenges of storage locality and network performance sensitivity. This provisioner includes the designs of CRD (custom resource definition)-based workload abstraction, endpoint provisioning mechanism for port allocation and reclamation, and volume provisioning mechanism for disk partitioning and mounting. When a request for an HDFS cluster is received, the provisioner will find a group of hosts with enough disk space, partition the disks using LVM (logical volume manager), and then create the corresponding PV (persistent volume) objects for the service pods. To avoid the overhead of the overlay network, the host network is used in this scenario. The allocation records of the ports on hosts are managed by the provisioner so that different containerized HDFS clusters can share the same port space without conflicts.

To achieve network isolation, the controllers can also use the MAC-based VLAN and route policies to restrict the security risks among different application frameworks/clusters. As a general entry for a set of related objects, the Kubernetes operator-based controller design also facilitates the implementation of multi-tenant access control.

Device multiplexing plugins

The device multiplexing plugins are designed for sharing the high-performance hardware with multiple jobs to improve resource utilization. The main devices that need to be multiplexed include compute accelerators and high-speed network cards. On the GPU side, a new GPU device plugin for Kubernetes with its scheduler extender is developed. Compared to the existing solutions [31, 32], it supports matching GPU by model and allocating resources by both computing power and memory, where the computing power is abstracted as "milli-GPU". To enforce performance isolation, system call interception and external resource monitoring mechanisms are adopted. On the RDMA side, new device plugins are developed by improving the open-source solution for InfiniBand [33]. The SR-IOV-based virtualization is used to implement mandatory resource constraints, while compatibility design is also made to multiplex other RDMA devices like RoCE.

In order to solve the complexity issues of configuring high-performance devices in containerized environments, several utilities are developed. An RDMA device information detector is developed for fetching the additional indexes of virtualized devices, which helps the applications in containers to bind network interfaces. An RDMA job launcher is developed for setting essential system parameters like the pinned memory size in a secure manner. Other utilities like the multi-network IP address mapper, the shared memory setup tool and the GPU monitor agent are also available. For the application-specific container images supplied by a service stack, these utilities can be integrated and executed implicitly to promote the usability and portability of the stack.

High-performance containerized communication

The device multiplexing plugins for high-performance hardware solve the feasibility and utilization issues. Further, performance optimization needs to be considered. Several designs for high-performance containerized communication are proposed. Representative technologies are introduced as follows.

To accelerate distributed big data jobs, an RDMA-based remote shuffle service [34] is designed. Its basic idea inherits the existing work [35] in the industry, while its innovation is mainly in the cloud-native design and RDMA communication. This shuffle service runs as a container cluster. Benefiting from the provisioning mechanism of the full-stack operator, the local disk-based storage can be partitioned and mounted automatically. An RDMA-based shuffle client integrated with the Spark framework is developed to enable high-throughput communication with the shuffle service. The client and service use control messages to manage the allocation of pinned communication buffers.

To adapt the separation architecture of compute and storage, and provide better performance in this case, a containerized storage access layer [36] is proposed. This layer runs in the compute cluster as a cache service. The core implementation is based on Alluxio [37], and the

containerized provisioning is also based on the full-stack operator. To accelerate upper-layer applications that do not support RDMA, the IPoIB network is introduced. The multi-network IP address mapper will help applications to find this network plane for data access.

Dynamic interactive workflow

For typical application scenarios with inter-job workflows, it is necessary to design general underlying mechanisms and dedicated upper-layer routines to balance universality and usability. The key point of co-designing these two types of components is how to effectively combine manual operations with automated processes. The dynamic interactive workflow design is proposed to address this issue. At the lower layer, an enhanced workflow engine is designed. It supports scheduling logic based on dynamic DAG (directed acyclic graph) so that the manual interactions can be inserted dynamically, and complex conditional concurrency can be implemented. At the upper layer, dedicated interactive tools need to be customized to make the underlying orchestration transparent to the users.

One case of the dynamic interactive workflow design is the guided automatic learning technology [38] for the artificial intelligence model development. The underlying workflow connects the phases of online labeling, data pre-processing, model training, model serving and verification. In the model training phase, a set of batch jobs will be launched concurrently according to the algorithms of automatic hyperparameter tuning and neural network architecture search. The jobs may be created or terminated dynamically when triggered by certain conditions. This design allows not only to execute a workflow linearly, but also to jump between steps following certain rules. This kind of manual intervention is used to push the model's evolution to the desired direction. Historical execution paths created by dynamic interactive workflows can be accurately recorded by the provenance mechanism [39].

Structured asset delivery management

The structured asset delivery management mechanism [40] is designed to maintain the data assets and software artifacts involved in workflows of artificial intelligence and big data application frameworks. It aims to produce deliverable assets with complete encapsulation and documentation. The service consists of a DSL for asset meta-information and a continuous delivery engine for asset building. It allows the users to define the deliverable assets in a declarative language, through which the workflow will be created and scheduled automatically, and the encapsulation and documentation will be generated according to the meta-information of input objects.

The DSL for asset meta-information is based on JSON. It describes a deliverable asset using five sets of fields: general, inputs, outputs, dependencies and extensions. The general fields include the basic meta-information of the asset like its name, type and version, which determines the application framework and job type for building the asset. The inputs and outputs provide essential information to create the job for building the asset. The dependencies are used to build the DAG for workflow orchestration. The extensions are used for creating documents and other subsidiary artifacts. The asset meta-information in this DSL can be manually written by the users or automatically generated by the service stack through graphical configuration.

The continuous delivery engine for asset building has two main responsibilities: maintaining the meta-information database and driving the workflow engine. Its input is parsed from the asset meta-information in the DSL. The workflow is often inter-service or inter-cluster, especially for the final steps of encapsulation, documentation and delivery, so the engine supports distributed asynchronous workflows. The aforementioned provenance mechanism is also employed to trace the relationship among assets.

Product implementation of the OMStack

The multi-dimensional extensible architecture as well as its key technologies has been implemented in a newly proposed service stack – OMStack (Oriental Mind Stack). This service stack consists of a series of production-grade cloud services. The relationship between the services and their key designs is shown in Fig. 3.

OMStack overview

OMStack is a cloud-native service stack providing various types of PaaS and SaaS services. It uses containerization technology as a common base, and high-performance computing technology as a characteristic ability. It aims at serving the mainstream workloads widely used by modern enterprises such as artificial intelligence and big data.

The overall architecture of OMStack is shown in Fig. 4. The component layout of OMStack is guided by the dimensional analysis using the cloud-native light-cone model. Different PaaS services cover four scales (A1–A4 in Table 1) of the application dimension, while the infrastructures used by the stack are organized into four scales (I1–I4 in Table 1) of the infrastructure dimension. The SaaS layer includes an integrated portal and application services for end-users. The dependencies between components are represented by dotted lines in the figure.

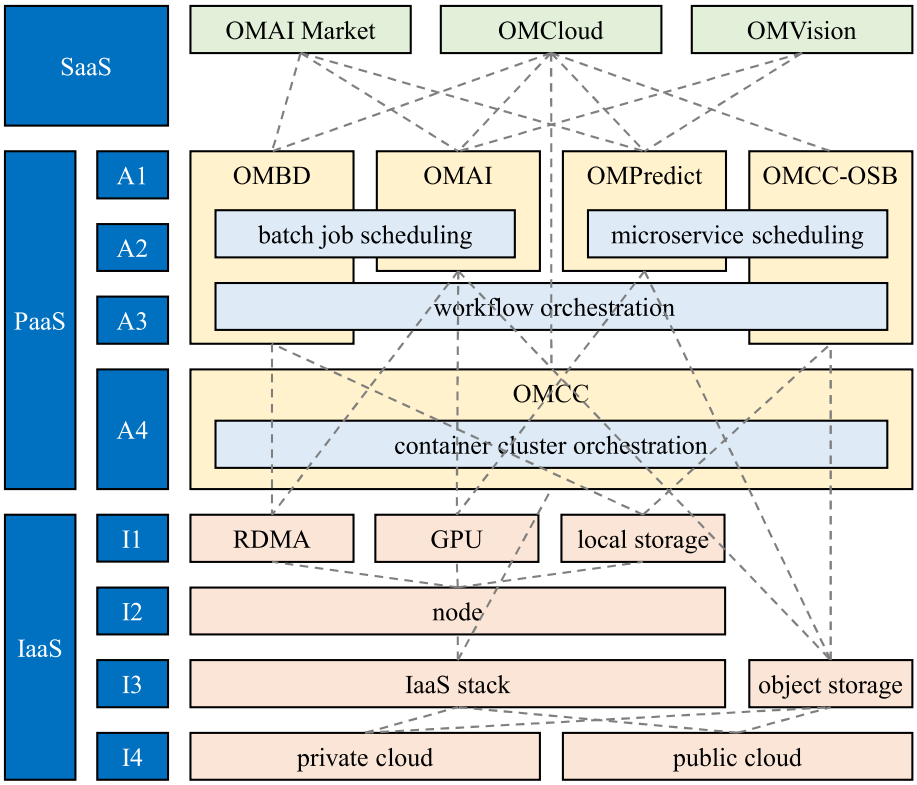

Fig. 4 The overall architecture of OMStack

PaaS services

Two layers of PaaS services are implemented. The lower layer is a general-purpose container cluster orchestration service covering the application cluster (A4) scale, while the upper layer is a group of domain-specific container application services covering the process (A1) to the application framework (A3) scales. These container application services involve supporting both batch jobs and microservices, where container-based scheduling is their core duty.

OMCC provides a subsidiary service named OMCC-OSB. It supports on-demand provisioning for common service middleware such as databases, web servers and cache services via the OSB (open service broker) interface. In an enterprise environment, it provides users with an easy-to-use service catalog that helps reduce the costs of operation and maintenance.

OMCC – container cluster service

OMCC [27] takes the managed Kubernetes cluster as the first-class entity, and provides rich capabilities inside and outside container clusters. Inside a container cluster, OMCC supports workload management, service orchestration and automatic maintenance for containerized applications, which significantly expands the functionality of traditional dashboards. Outside of container clusters, OMCC supports on-demand provisioning, elastic scaling and multi-cloud deployment for Kubernetes clusters, which can be considered as an extension of the IaaS layer. OMCC is a bridging layer that hides the complexity of the infrastructure, and offers consistent and universal capabilities for upper-layer services.

OMBD – big data service

OMBD is a big data analysis and processing service that supplies application frameworks of multiple paradigms including batch (Hadoop, Spark), streaming (Flink, Spark Streaming), NoSQL database (HBase), data warehouse (Hive), etc. On compute, it supports either scheduling jobs in a shared framework instance or provisioning independent framework instances for different businesses. On storage, it can provide on-demand containerized HDFS clusters, while external storage services are also supported. It allows the compute and storage clusters to run on either separated or shared infrastructure. The

high-performance RDMA network is adopted to implement efficient communication for distributed jobs.

OMAI – artificial intelligence service

OMAI [25] serves the full life-cycle of artificial intelligence including algorithm development, model training and online inference. Diverse AI engines like TensorFlow, PyTorch and MindSpore are integrated. Training modes based on custom algorithms, preset algorithms and guided automatic learning are ready for users with different knowledge backgrounds. Scheduling of large-scale distributed jobs with multiple distribution modes is supported by the scheduler with specific policies, and accelerated by the optional RDMA communication. Object storage is the standard storage service that enables data sharing in the whole pipeline, while other storage services are also compatible via the CSI (container storage interface) mechanism.

OMPredict – AI inference service

OMPredict is a professional AI inference service. It is well suited for enterprises or cloud service providers with diverse and variable AI inference requests. It supports running either model files in the FaaS mode or model images in the CaaS mode. Container-based elastic scaling mechanism is designed, which supports "scale to zero" to save resources for idle services. Access control and usage measurement are implemented in a universal gateway layer so that a complete commercial model service can be easily implemented using native models. Models produced by OMAI can be encapsulated and deployed in OMPredict automatically by the structured asset delivery mechanism.

OMBatch – batch processing service

OMBatch is a general-purpose batch processing service that implements the unified runtime abstraction and serves the scheduling of batch jobs. It is a basis of OMAI and OMBD, and it also works as an inter-job workflow engine within a service. OMBatch addresses the key issues of mapping stateful semantics like gang-scheduling to container-based microservices.

OMAutomation – automated workflow service

OMAutomation is a workflow orchestrator supporting inter-service and inter-cluster task orchestration. It serves many cases of service coordination in OMStack, such as the structured asset delivery and the implicit incremental training. When building a business-specific system with OMStack, OMAutomation can work as a general coordinator for upper-layer applications and lower-layer services.

SaaS services

The SaaS layer supplies business-oriented application services to present the functionalities of OMStack to the end-users. It represents the value extension of our model and architecture to the application level.

OMCloud – integration portal

OMCloud is a portal that integrates the user interfaces of all the OMStack services. Under the unified graphical interface, a set of universal mechanisms are designed to achieve standardized modular integration of services, which involve a user system, a billing system, a messaging service, a monitor service and so on. In on-premise clusters for enterprises, OMCloud can be easily integrated with existing IT systems due to its open designs.

OMAI Market – AI market service

OMAI Market provides an on-demand service trading platform for the supply and demand sides of artificial intelligence models. Neural network models from OMAI and statistical learning models from OMBD can be packaged as pay-as-you-go services with both auto-generated GUIs and RESTful APIs via this platform. Elastic model serving is supported by leveraging OMPredict. Usability designs including model accuracy assessment and JSON parsing guidance are also available.

OMVision – machine vision service

OMVision is an image and video analysis platform. It supports intelligent capabilities such as target detection, entity recognition and event discovery by using machine vision algorithms. Benefiting from the cloud-native design, algorithms can be dynamically deployed and elastically scaled on both cloud and edge sides according to requirements. Implicit incremental training and transfer learning for various scenarios are the key features enabled by the underlying OMAI and OMPredict.

Evaluation

To demonstrate the practicality, efficiency and reliability of OMStack, a set of performance experiments are performed and analyzed, representative results of which are presented in this section. OMStack has also been deployed and used in many production applications. This section will introduce some typical cases.

Performance experiments

Three typical experiments are introduced to illustrate the performance characteristics of OMStack. They

cover the life-cycle of interface invocation, job scheduling and job execution. The test environment is an OMStack cluster deployed on an OpenStack-based VM cluster with 120 nodes. Each virtual node has 16× vCPU, 64GB memory, and a 200GB data disk. The data plane network is a 10G ethernet supporting RoCE. The software environment involves CentOS 7.8, Kubernetes 1.19, Spark 3.0.2 and HiBench 8.0.

Service interface invocation

This experiment focuses on the response time of calling the REST APIs of OMAI concurrently. Each "view" interface synchronously returns a query result from an 100,000-record table, while each "create" interface asynchronously returns a state code for generating a single object. The components handling the requests involve OMCC, Kubernetes and PostgreSQL in addition to OMAI. A multi-thread client calls each interface with 10, 50 and 100 concurrent invocations, and the total number of requests is 1,000 for each interface in each case. The results are shown in Fig. 5. It shows that the response time increases linearly with the concurrency. For most interfaces, the response time of 10-concurrency is less than 200ms, and that of 100-concurrency is less than 2s. This reflects the high efficiency of OMStack components under high pressure.

Large-scale job scheduling

This experiment focuses on the time overheads for the main parts in the life-cycles of concurrent jobs in OMAI. The test load is a logistic regression training algorithm. The recorded times include creation time (in Kubernetes and PostgreSQL), scheduling time (in Kubernetes), and execution time (in containers). Sufficient CPUs and memory are prepared so that all the concurrent jobs do not need to be queued for resources. The results are shown in Fig. 6, where time is the average time of all concurrent jobs. It shows that creation and scheduling times are roughly linearly related to the number of jobs. This behavior is consistent with the Kubernetes modeling analysis given by [41]. The execution time remains basically constant, which reflects the scalability of OMStack in the number of jobs.

Distributed job execution

This experiment compares the execution time of a typical big data job with different configurations. The test load is the TeraSort benchmark in HiBench [42], while the objects of comparison involve the containerized environment provided by OMBD vs. the traditional VM environment, as well as the remote shuffle service vs. the built-in shuffle mechanism in Spark. The results are shown in Fig. 7. In terms of containerization, OMBD has acceptable overheads for some cases with small data sizes, but it shows an advantage in the 1TB case. Besides the performance fluctuation of Java applications, it is probably because containers provide better performance isolation [43]. In terms of shuffle, our design shows a significant advantage at 1TB data.

Application cases

The full stack or parts of OMStack has been applied in many scenarios in industry and academia to serve real businesses. It provides cloud-native services involving container clusters, big data, and artificial intelligence to support upper-layer applications.

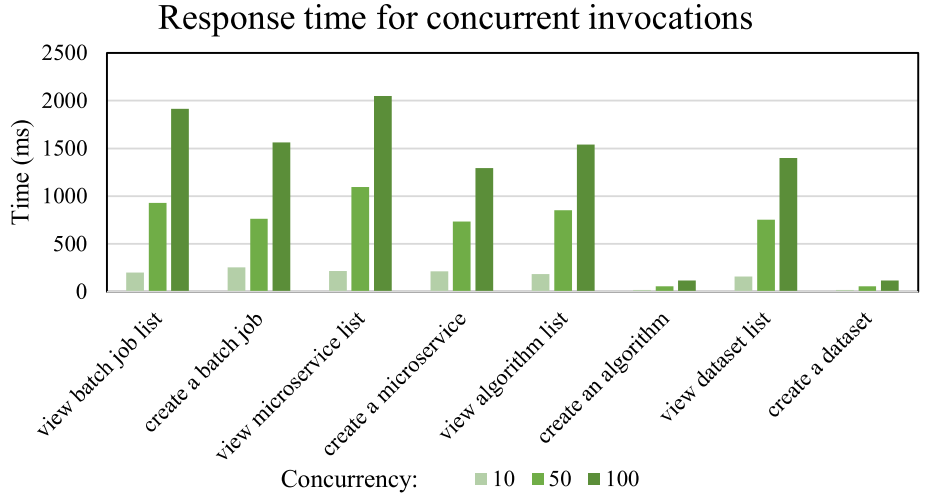

Fig. 5 Test results of service interface invocation

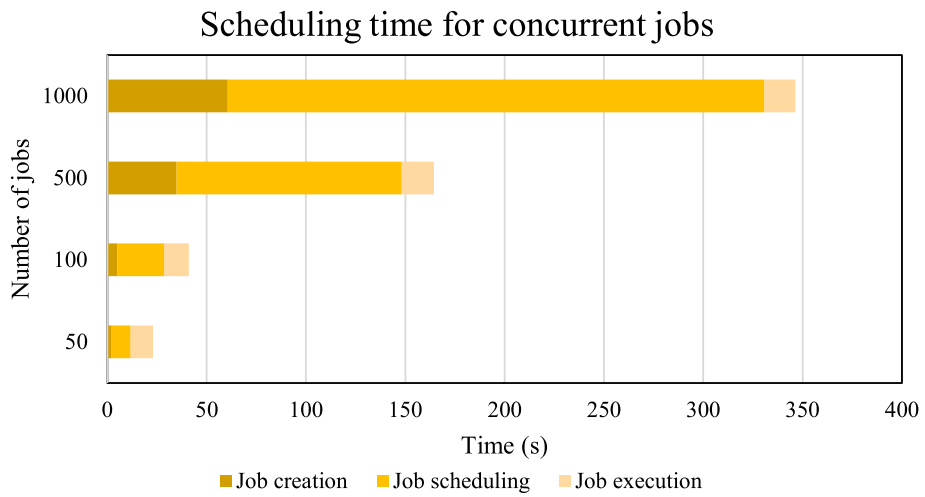

Fig. 6 Test results of large-scale job scheduling

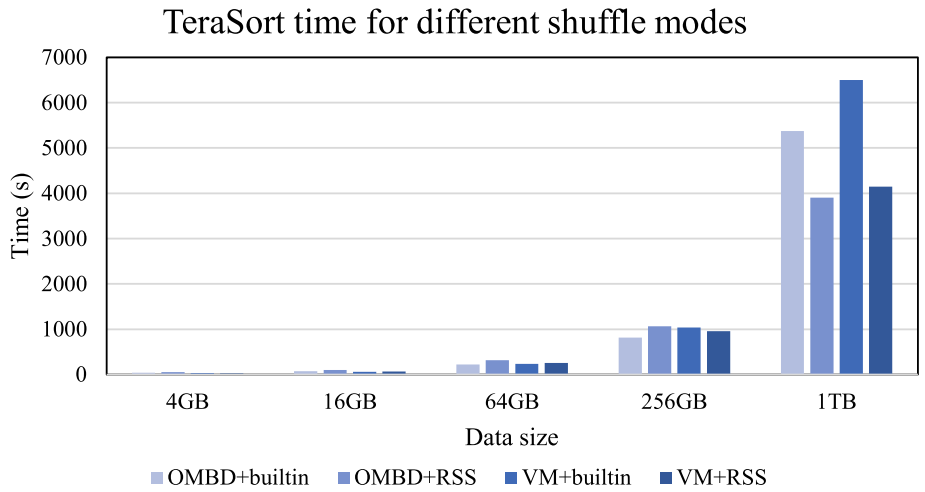

Fig. 7 Test results of distributed job execution

AI service in Big Earth Data

OMStack supports the construction of the artificial intelligence cloud service in the SDG (Sustainable Development Goals) Big Data Platform of the Big Earth Data program [44]. Scientists in fields such as remote sensing and physical geography use OMAI to develop and train models to facilitate their research. To lower the threshold for the non-computer professionals to use AI algorithms, a variety of domain-oriented models are preset, which are managed by the structured asset delivery mechanism so that the release and iteration can be automated. Guided automatic learning processes for geographic image processing and analysis are designed. Dedicated labeling tools and image pre-processing algorithms are developed to serve typical applications such as surface semantic segmentation and object detection in a pure GUI way. On the system side, the device multiplexing mechanism for high-performance accelerators improves the utilization of GPUs. The public cloud service of AI [45] has been launched and serves many institutions, which verifies the stability and reliability of OMStack.

iPaaS platform for enterprises

An iPaaS (Integration Platform as a Service) platform is designed based on OMStack. It aims to provide self-service provisioning and management of common PaaS-layer services for developers in the private/hybrid clouds of enterprises. OMCC and its affiliated OMCC-OSB are the core of the iPaaS platform. Benefiting from the unified runtime abstraction and full-stack application-specific

controllers, various services with different distributed architectures and scheduling modes can be organized and orchestrated uniformly. The service catalog already covers dozens of services including web applications, big data frameworks, message middleware, etc. Based on the DevOps concept, the automation designs in OMStack significantly reduce the cost of IT operations. The iPaaS platform has been adopted by several enterprise customers involving telecom operators, software manufacturers and hospitals.

Smart industrial inspection solution

Our team proposes a smart industrial inspection solution using the components in OMStack. This allows traditional industries to benefit from the advantages of cloud-native and artificial intelligence. The core of the solution is customized based on OMVision. Algorithms developed for specific inspection scenarios are executed by the inference service of OMVision and dynamically scheduled on demand. The implicit incremental training and transfer learning technologies enable the accuracy of models to be automatically improved in use based on user feedback. The intranet penetration mechanism can solve the problem that some businesses need to access edge-side networks or enterprise intranet from the Internet securely. This solution has been deployed for customers in multiple industries including machinery manufacturing, road maintenance and railway vehicles.

Related work

On the architecture analysis of cloud computing, besides the dimension- or taxonomy-based methods introduced in the section of background, formalized or quantitative methods have also been proposed. Binz et al. [46] proposed an enterprise topology graph model to match the architecture of cloud infrastructure with the organization structure for optimizing operational costs. Andrikopoulos et al. [47] designed a quantitative estimation method to analyze the CAP (Consistency–Availability–Partition tolerance) properties of cloud-native applications. Halabi et al. [48] designed a quantitative evaluation method using relative matrices to study the security properties of cloud service providers. Szalay et al. [49] proposed a quantitative model to formalize the state placement problem of cloud-native applications, which can guide the architecture optimization for cloud databases. Salmon et al. [50] applied classic models like End-User Computing Satisfaction (EUCS) to characterize the multifaceted properties of cloud services. Chemashkin et al. [51] used a control theory model to characterize Kubernetes operators, which aims to guide the state-space design of application-specific controllers. These subdomain-focused methods are instructive for designing a complete formalized and quantitative analysis methodology for cloud-native service stacks.

On the architecture design of cloud-native service stacks, theoretical studies and engineering practices are emerging. Balalaie et al. [52] reported the experience of migrating traditional architectures to the cloud-native architecture, which provides a common pattern for building DevOps-enabled microservice stacks. Pahl et al. [53] summarized a group of architectural definitions, principles and patterns for cloud service stacks, and proposed a reference architectural style. Moreno et al. [54] proposed a complete cloud architecture for enterprises to serve the full life-cycle of big data and artificial intelligence applications. Kosińska et al. [55] designed AMoCNA, a cloud-native framework with autonomic computing features for provisioning and scheduling diverse applications to improve manageability. Gundu et al. [56] aimed at solving load balancing challenges in multi-cloud and hybrid IT infrastructures in order to enhance the scalability of cloud computing architectures. Moreover, Goniwada [3] summarized and detailed many classic designs of overall and partial architectures in cloud-native systems. These studies and practices provide useful inputs for the model and architecture designs of this paper.

Conclusion

Cloud-native computing is popular in recent years. This paper aims at designing a practical architecture and a service stack to improve the existing designs in the cloud-native ecosystem. A cloud-native light-cone model extending the classic hourglass model is proposed, which highlights the extensibility features of a service stack on four dimensions: application, infrastructure, tenant and workflow. Guided by the model, the challenges in designing an extensible service stack are categorized and analyzed by dimension. Key issues such as diverse scheduling modes and framework on-demand provisioning are identified. To solve these challenges, a multi-dimensional extensible cloud-native architecture and a set of key technologies are designed by using a systematic approach based on the above model. These designs revolve around six main functions: business management, runtime abstraction, environment allocation, resource maintenance, architecture adaption and logic orchestration. Representative technologies including the Kubernetes cluster bootstrapped creation and full-stack application-specific controllers are developed to improve these functions.

These designs are not limited to theoretical and technical ideas, but are implemented as production-grade software and applied in real businesses. OMStack is proposed to realize the multi-dimensional extensible cloud-native architecture. A group of PaaS and SaaS services

integrating the key technologies are implemented in OMStack to serve enterprise applications. Typical services include the OMCC container cluster service, the OMBD big data service and the OMAI artificial intelligence service. The functionality and performance of these services and the underlying technologies have been verified by experiments. OMStack has supported many projects in industry and academia. Its value has been proven in practice.

In the future, the quantitative analysis method based on the cloud-native light-cone model will be explored so that the model's guidance to practice will be enhanced. The AIOps technique will be studied to improve the full-stack autonomy. In addition, more cloud-native services, especially IaaS services, will be designed and implemented to promote the completeness of OMStack.

Acknowledgements
We would like to thank the Oriental Mind team for the contributions to the high-quality implementation of this software stack.

Authors' contributions
Jian Lin is the leader of the OMStack project. He put forward the main ideas of architectural modeling and analysis, and wrote the main part of this manuscript. Dongming Xie is the software architect of OMStack. He designed many key technologies and wrote the part of application cases. Jinjun Huang, Zinan Liao and Long Ye are the software architects and chief developers of the sub-systems of OMStack. Jinjun Huang is responsible for the performance experiments. All the authors reviewed and approved this manuscript.

Funding
This research is supported in part by the Strategic Priority Research Program of the Chinese Academy of Sciences (XDA19020400) and the 13th "3551 Optics Valley Talent Schema" (2020).

Availability of data and materials
The experiment results of this paper are available from the corresponding author on reasonable request.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 1 June 2022 Accepted: 12 November 2022
Published online: 24 November 2022

References

1. Gundu SR, Panem CA, Pratik P (2022) Cloud Computing and its Service Oriented Mechanism. Akinik Publications, New Delhi
2. Bulla CM, Bhojannavar SS, Danawade VM (2013) Cloud Computing: Research Activities and Challenges. Int J Emerg Trends Technol Comput Sci 2(5):206–214
3. Goniwada SR (2022) Cloud Native Architecture and Design: A Handbook for Modern Day Architecture and Design with Enterprise-Grade Examples. Apress, New York
4. Cloud Native Computing Foundation. Kubernetes. https://kubernetes.io. Accessed 1 June 2022
5. QingCloud. KubeSphere. https://kubesphere.io. Accessed 1 June 2022
6. Red Hat. OpenShift. https://openshift.com. Accessed 1 June 2022
7. Hendrickson S, Sturdevant S, Harter T, Venkataramani V, Arpaci-Dusseau AC, Arpaci-Dusseau RH (2016) Serverless Computation with OpenLambda. In: Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud). USENIX Association, Denver, pp 1–7
8. Saltzer JH, Reed DP, Clark DD (1984) End-to-end Arguments in System Design. ACM Trans Comput Syst (TOCS) 2(4):277–288
9. Foster I, Kesselman C (1998) The Grid: Blueprint for a New Computing Infrastructure, 1st edn. Morgan Kaufmann, San Francisco
10. Lin J, Zha L, Xu Z (2013) Consolidated Cluster Systems for Data Centers in the Cloud Age: a Survey and Analysis. Front Comput Sci 7(1):1–19
11. Kruchten PB (1995) The 4+1 View Model of Architecture. IEEE Softw 12(6):42–50
12. Choi H, Yeom K (2002) An Approach to Software Architecture Evaluation with the 4+1 View Model of Architecture. In: Proceedings of 9th Asia-Pacific Software Engineering Conference (APSEC). IEEE, Gold Coast, pp 286–293
13. Thramboulidis K (2010) The 3+1 SysML View-Model in Model Integrated Mechatronics. J Softw Eng Appl 3(2):109–118
14. Hamdaqa M, Tahvildari L (2014) The (5+1) Architectural View Model for Cloud Applications. In: Proceedings of 24th Annual International Conference on Computer Science and Software Engineering (CASCON). IBM Corp., Markham, pp 46–60
15. Liu F, Tong J, Mao J, Bohn R, Messina J, Badger L, Leaf D (2011) NIST Cloud Computing Reference Architecture. NIST Spec Publ 500–292:1–28
16. Polash F, Abuhussein A, Shiva S (2014) A Survey of Cloud Computing Taxonomies: Rationale and Overview. In: Proceedings of the 9th International Conference for Internet Technology and Secured Transactions (ICITST). IEEE, London, pp 459–465
17. Brogi A, Soldani J, Wang P (2014) TOSCA in a Nutshell: Promises and Perspectives. In: European Conference on Service-Oriented and Cloud Computing. Springer, Manchester, pp 171–186
18. Kratzke N, Peinl R (2016) ClouNS – a Cloud-Native Application Reference Model for Enterprise Architects. In: Proceedings of the IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW). IEEE, Vienna, pp 1–10
19. Kratzke N, Quint PC (2017) Understanding Cloud-Native Applications after 10 Years of Cloud Computing - a Systematic Mapping Study. J Syst Softw 126:1–16
20. Bisong E (2019) Kubeflow and Kubeflow Pipelines. Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, New York, pp 671–685
21. Cloud Native Computing Foundation. Volcano. https://volcano.sh. Accessed 1 June 2022
22. Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I (2011) Disk-Locality in Datacenter Computing Considered Irrelevant. In: Proceedings of the 13th Workshop on Hot Topics in Operating Systems (HotOS). USENIX Association, Napa, pp 1–5
23. Link C, Sarran J, Grigoryan G, Kwon M, Rafique MM, Carithers WR (2019) Container Orchestration by Kubernetes for RDMA Networking. In: Proceedings of the IEEE 27th International Conference on Network Protocols (ICNP). IEEE, Chicago, pp 1–2
24. Baliyan DS. Introduction to Multi-Tenancy in Kubernetes. https://www.cncf.io/blog/2021/12/20/introduction-to-multi-tenancy-in-kubernetes/. Accessed 1 June 2022
25. Lin J, Xie D, Yu B (2020) Research on Cloud Service Adaptation of Deep Learning. Softw Guide 19(6):1–8
26. Istio authors. Istio. https://istio.io. Accessed 1 June 2022
27. Xie D, Huang L, Huang J, Lin J (2022) Design and Implementation of Container Cluster Service for Multi-Cloud. Softw Guide 21(6):169–175
28. Brikman Y (2019) Terraform: Up & Running: Writing Infrastructure as Code. O'Reilly Media, Sebastopol
29. Gundu SR, Panem CA, Anuradha T (2020) Hybrid IT and Multi Cloud an Emerging Trend and Improved Performance in Cloud Computing. SN Comput Sci 1(256):1–6
30. Dobies J, Wood J (2020) Kubernetes Operators: Automating the Container Orchestration Platform. O'Reilly Media, Sebastopol
31. Gu J, Song S, Li Y, Luo H (2018) GaiaGPU: Sharing GPUs in Container Clouds. In: Proceedings of the IEEE 8th International Conference on Big Data and Cloud Computing (BDCloud). IEEE, Melbourne, pp 469–476
32. Alibaba Cloud. GPU Sharing Scheduler Extender in Kubernetes. https://github.com/AliyunContainerService/gpushare-scheduler-extender. Accessed 1 June 2022

33. Mellanox. RDMA Shared Device Plugin. https://github.com/mellanox/k8s-rdma-shared-dev-plugin. Accessed 1 June 2022
34. Lin J, Hong Z (2022) RDMA-based Big Data Transmission System, Method. Device and Storage Medium. China Patent 202210047977.3. State Intellectual Property Office, Beijing
35. Bansal M, Yang B. Zeus: Uber's Highly Scalable and Distributed Shuffle as a Service. https://databricks.com/session_na20/zeus-ubers-highly-scalable-and-distributed-shuffle-as-a-service. Accessed 1 June 2022
36. Huang L, Yu B, Xie D, Lin J (2021) Alluxio-based Big Data Job Operation System and Method. China Patent 202111092499.X. State Intellectual Property Office, Beijing
37. Li H (2018) Alluxio: A virtual distributed file system. PhD thesis, University of California, Berkeley
38. Xie D, Xia J, Yi Q, Lin J (2020) Deep Learning Guide Device and Method. China Patent 202010675467.1. State Intellectual Property Office, Beijing
39. Lin J, Xie D (2020) OMProv: Provenance Mechanism for Objects in Deep Learning. In: Proceedings of the 1st Intelligent Data – From Data to Knowledge Workshop (DOING). Springer, Lyon, pp 98–109
40. Lin J, Yu B (2021) Data Asset Meta-Information Processing System and Method. China Patent 202110023049.9. State Intellectual Property Office, Beijing
41. Medel V, Tolosana-Calasanz R, Bañares JÁ, Arronategui U, Rana OF (2018) Characterising Resource Management Performance in Kubernetes. Comput Electr Eng 68:286–297
42. Huang S, Huang J, Dai J, Xie T, Huang B (2010) The HiBench Benchmark Suite: Characterization of the MapReduce-based Data Analysis. In: Proceedings of the IEEE 26th International Conference on Data Engineering Workshops (ICDEW). IEEE, Long Beach, pp 41–51
43. Xavier MG, Neves MV, De Rose CAF (2014) A Performance Comparison of Container-based Virtualization Systems for MapReduce Clusters. In: Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP). IEEE, Turin, pp 299–306
44. Guo H (2017) Big Earth Data: A New Frontier in Earth and Information Sciences. Big Earth Data 1(1–2):4–20
45. Big Earth Data Science Engineering Program. Deep Learning Cloud System of the SDG Big Data Platform. https://sdg.casearth.cn/en/onlineTools/AI. Accessed 1 June 2022
46. Binz T, Fehling C, Leymann F, Nowak A, Schumm D (2012) Formalizing the Cloud through Enterprise Topology Graphs. In: Proceedings of the IEEE 5th International Conference on Cloud Computing (CLOUD). IEEE, Honolulu, pp 742–749
47. Andrikopoulos V, Strauch S, Fehling C, Leymann F (2013) CAP-Oriented Design for Cloud-Native Applications, Communications in Computer and Information Science, vol 367. Springer International Publishing, Cham, pp 215–229
48. Halabi T, Bellaiche M (2017) Towards Quantification and Evaluation of Security of Cloud Service Providers. J Inf Secur Appl 33:55–65
49. Szalay M, Mátray P, Toka L (2021) State Management for Cloud-Native Applications. Electronics 10(4):423
50. Salmon M, Parmar A (2022) Cloud Computing at Unitec. Tech. rep, United Institute of Technology
51. Chemashkin FY, Drobintsev PD (2021) Kubernetes Operators as a Control System for Cloud-Native Applications. Tech. rep., Peter the Great St. Petersburg Polytechnic University
52. Balalaie A, Heydarnoori A, Jamshidi P (2016) Microservices Architecture Enables DevOps: An Experience Report on Migration to a Cloud-Native Architecture. IEEE Software 33(3):42–52
53. Pahl C, Jamshidi P, Zimmermann O (2018) Architectural Principles for Cloud Software. ACM Trans Internet Technol 18(2):1–23
54. Moreno C, González RAC, Viedma EH (2019) Data and Artificial Intelligence Strategy: A Conceptual Enterprise Big Data Cloud Architecture to Enable Market-Oriented Organisations. Int J Interact Multimedia Artif Intell 5(6):7–14
55. Kosińska J, Zieliński K (2020) Autonomic Management Framework for Cloud-Native Applications. J Grid Comput 18(4):779–796
56. Gundu SR, Panem CA, Anuradha T, Gad R (2022) Emerging Computational Challenges in Cloud Computing and RTEAH Algorithm based Solution. J Ambient Intell Humanized Comput 13:4249–4263

Publisher's Note