IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 July 2022, accepted 8 August 2022, date of publication 16 August 2022, date of current version 24 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3198971

RESEARCH ARTICLE

Band-Area Resource Management Platform and Accelerated Particle Swarm Optimization Algorithm for Container Deployment in Internet-of-Things Cloud

MINGXUE OUYANG[1], JIANQING XI[1], WEIHUA BAI[2], AND KEQIN LI[3], (Fellow, IEEE)
[1]School of Software Engineering, South China University of Technology, Guangzhou 510006, China
[2]School of Computer Science, Zhaoqing University, Zhaoqing 526061, China
[3]Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

Corresponding authors: Jianqing Xi (jianqingxi@163.com), Weihua Bai (bandwerbai@gmail.com), and Keqin Li (lik@newpaltz.edu)

This work was supported in part by the Science and Technology Plan Project of Guangdong Province, China, under Grant 2014B010112007 and Grant 2016B010124010; in part by the Guangdong Province Educational Science Planning Project under Grant 2019KTSCX199; in part by the Zhaoqing Science and Technology Special Fund Project under Grant 2020G1004; in part by the Zhaoqing University Science and Technology Projects under Grant zlgc201933; and in part by the Teaching Reform Project of University Public Computer Course of Guangdong Province under Grant 2021-GGJGJ-012.

ABSTRACT The method of building and deploying applications through the combination of container virtualization technology and a microservices framework has been widely used in Internet-of-Things clouds. However, there are gaps and a lack of coordination mechanisms between the Internet-of-Things and cloud computing. This study constructs a resource management platform, which is based on application container virtualization technology and combined with the microservices framework. The platform provide a support environment for the construction and deployment of Internet-of-Things cloud applications. However, there is no unified specification for the microservices templates. Therefore, a new service model called tool service was designed. The invocation relationship between services is studied, and developers can combine services through the invocation relationship between services to form a service function chain. However, container-based service deployment remains an unresolved issue. The deployment method of a container involves the quality of service of end users and the profit of cloud providers. To balance the profits of both parties, it is necessary to minimize the service response time and improve the resource utilization of the cloud data center. To address this problem, an accelerated particle swarm optimization strategy is proposed to realize service deployment. Through the invocation relationship between services, the execution containers are aggregated, so as to reduce the service transmission overhead and improve resource utilization. Compared with the experimental results of existing deployment strategies, the proposed optimization strategy has significantly improved performance parameters such as service transmission overhead, container aggregation, and resource utilization.

INDEX TERMS Accelerated particle swarm optimization, cloud computing, container, Internet-of-things, microservices, multi-objective optimization.

I. INTRODUCTION

Cloud computing is a distributed computing model with two salient characteristics: virtualization [1] and on-demand self-service [2]. Virtualization is the abstraction of computing and software resources, and provides various services to end users, such as hardware, networks, and application levels. Computing resources usually provide users with various computing services in the form of virtual machines (VMs) or containers, whereas the application level is provided to

The associate editor coordinating the review of this manuscript and approving it for publication was Wei-Wen Hu.

end users using specific application patterns or models, such as Microsoft Azure. On-demand self-service refers to the automatic and self-service use of computing resources (such as network, CPU, and storage) and software for users with immediate needs in a specific time period, without manual interaction with service providers.

In recent years, the Internet-of-Things (IoT) cloud has provided specific application services for many application fields by integrating IoT and cloud computing technology, which is widely used by researchers and enterprises. However, in the IoT cloud environment, with the rapid growth of end users and exponential growth of IoT devices, the IoT cloud has encountered unprecedented challenges [3]. The integration of IoT applications with the existing enterprise business system on the cloud architecture into a complete enterprise application system is a problem that cloud service providers must consider. However, there are gaps and a lack of coordination mechanisms between the IoT and cloud, and service providers usually complement and integrate IoT and cloud by offering IoT cloud platforms [4].

Because microservices architecture and IoT cloud application systems have similar architectural goals, the microservice approach has been widely used in the construction and deployment of IoT cloud application systems [5], [6], [7]. In contrast to traditional monolithic applications, the microservices approach develops monolithic applications into a set of fine-grained services through strong decoupling, which are usually distributed in a cloud data center (CDC) in the form of VM and container deployments. A lightweight communication mode is adopted between services, and an application system is constructed using a combination of services [8]. In the IoT cloud, it is an effective method to integrate the existing enterprise management system and IoT applications using a virtualization platform based on an application container [5]; that is, the virtualization platform can integrate the microservices method into the development of IoT cloud systems and deploy microservices using application container technology. Therefore, this study proposes a resource management platform based on an application container. The application container includes users, application services (such as microservices), messages, documents, and a set of operational rules [13]. End users can add the required enterprise business systems and IoT applications to the application container in a self-service manner, according to their own business requirements. The application system in the platform can interact through cooperation between services, and then use the computing resources of the CDC through the coordination of the application container.

The application container virtualization platform can provide a running support environment for enterprise applications built using microservices. However, the deployment and resource allocation of microservice instances are key issues [9]. At present, the microservices deployment method based on OS-level containers, such as the Docker, Linux Container (LXC), and Solaris Zones, is a trend. Compared with VM, this virtualization method is lightweight, and the computing resources of the system are packaged through namespace, which requires less storage resources and startup time. Service providers can easily deploy and expand microservices [10]. However, designing an effective service deployment and resource allocation algorithm must consider the quality of service (QoS) for end users and the profits of cloud providers, such as reducing the response time of services and improving CDC resource utilization. Based on this, we balance the interests of users and cloud providers by optimizing the service transmission overhead and CDC resource utilization. In the process of service deployment, the influence of the invocation relationship between services on service deployment is considered, and services are combined through the invocation relationship between services to form a service function chain (SFC) [11]. Aggregate the execution containers of the services in the SFC to the same physical node or the same data center as much as possible, the goal is to reduce the data transmission overhead of the service and improve CDC resource utilization.

In the IoT cloud, for the multi-objective optimization problem of microservices deployment and CDC resource allocation, it is difficult to obtain the optimal solution using traditional optimization methods (such as linear programming and convex optimization). Heuristic strategy has been widely used in combinatorial optimization problems in dynamic IoT clouds, which can reduce the complexity of the solution. Heuristic strategies include particle swarm optimization (PSO), artificial fish swarm (AF), ant colony optimization (ACO), bee algorithm (BA), genetic algorithm (GA), grey wolf algorithm (WA), and evolutionary game strategy. An improved PSO is the accelerated particle swarm optimization (APSO) strategy [12]. To make the algorithm converge faster and reduce the randomness of the algorithm, the APSO strategy adopts only the global optimal solution of the swarm in the iterative process. Here, in order to reduce the service transmission overhead and improve CDC resource utilization, we propose a multi-objective model of execution container deployment based on the invocation relationship between services. An improved ASPO algorithm is used to solve the multi-objective optimization problem.

The main contributions of this study are as follows:

-First, this study uses the application container virtualization technology to establish a resource management platform between IoT and cloud computing. The core module of the platform is a Band-area application container (BAC) [13]. BAC contains documents, fine-grained services(such as microservices), users, messages, and a set of operational rules. End users can add the required application services to BAC according to business requirements to integrate enterprise business systems and IoT applications.

-Second, to adapt to the interface characteristics of BAC, we propose a service model called ‘‘tool service’’ based on a microservices framework. Three invocation relationship models between services are studied: pipeline relationship (PR), production-consumer relationship (PCR), and random handshake relationship (RHR). Developers can combine

tool services into service function chains (SFCs) through the invocation relationship between services to build new applications.

-Third, this study establishes a service container deployment model that including service transmission overhead, execution container aggregation, and resource load balancing of the CDC.

-Fourth, based on the problem model, a multi-objective optimization model of execution container deployment was built. The model optimizes multiple objectives, such as the service transmission overhead, execution container aggregation value and resource load balancing of the CDC, and deploys services with the constraint that the demand of execution container is less than the total amount for the server resources.

-Finally, an improved accelerated particle swarm optimization algorithm is built to solve the multi-objective optimization problem of the execution container deployment of services. The algorithm adopts crowding distance to obtain the Pareto solution set, and selects the global optimal solution through the weighted sum function value of the three optimization objectives.

The remainder of this study is orgnized as follows. Section II describes related work. Section III presents the Band-area resource management platform. Section IV describes our system model. Section V discusses the deployment strategy of service execution containers in detail. Section VI discusses the experiments conducted, the results of which validate the effectiveness of the proposed deployment strategy. Finally, Section VII presents conclusions and future work.

II. RELATED WORK

A. OVERVIEW OF IoT CLOUD PLATFORM

At present, IoT cloud platforms have been applied in various fields such as government, enterprise, medical treatment, shopping, smart homes, transportation, and agriculture, providing end users with services such as large-scale sensor equipment, networks, application system construction and deployment, computing and storage. The IoT Platform as a Service has received extensive attention from various fields of applications and researchers. It is meaningful to combine IoT and the cloud into one framework to close the gap between IoT and cloud computing [14]. According to different architectural methods, the IoT cloud platform includes frameworks for application-based container virtualization and other technical frameworks.

1) FRAMEWORKS FOR APPLICATION-BASED CONTAINER VIRTUALIZATION

Typical application virtual containers are based on Microsoft Azure and the Google App Engine (GAE). The technical framework integrates the data generated by IoT application systems and devices into a cloud platform through the interface of the application container, and then uses the computing resources provided by the infrastructure in the cloud. For example, Microsoft's Lab of Things platform [15], which is mainly used for academic research, involves many fields such as smart homes and medical care. The platform structure consists of the client component HomeOS and Azure application container. HomeOS is used to share data sent by IoT devices, and the related application services are deployed in Azure. Nimbits [16] platform is based on a limited embedded system platform that mainly solves the problem of edge computing. The important data of devices are pushed to servers in the CDC through GAE, and some cloud application services are deployed in the Google App Engine.

2) OTHER TECHNICAL FRAMEWORKS

The technical framework is an IoT cloud platform developed for specific application fields. For example, the Oracle IoT [17] platform is applied to the processing of big data generated by sensor devices, including four components: Open, Insight, Secure, and Oracle. Open is responsible for the network connection of the sensing device, Insight calculates the business value of the device data, Secure is responsible for the security of the device, data, and network connection, and Big Data are predicted, analyzed, and integrated by Oracle.

In contrast to the above technical framework, the resource management platform based on the Band-area application container proposed in this study focuses on cooperation between application containers and aims at business. Through the interaction between tool services, cooperation between services and Band-area application containers, and coordination between application containers, IoT applications and existing enterprise applications are finally integrated into a complete enterprise application system.

B. RELEVANT DEPLOYMENT STRATEGIES

Microservice instance deployment and CDC resource utilization are important issues for IoT clouds, which are related to balancing profits between end users and cloud providers. Resource management optimization in the CDC has been the focus of researchers. The relevant service deployment strategies and scheduling methods are summarized as follows.

First, in a cloud environment, researchers tend to apply meta-heuristic strategies to obtain optimal solutions (i.e., Pareto solutions) for multi-objective optimization problems. The main reason is that the meta-heuristic strategy is a random algorithm, which is characterized by uncertainty and is not constrained by the mathematical properties of the optimization problem itself. In addition, individuals in the population can better adapt to and interact with the environment through cooperation, and have more opportunities to obtain the global optimal solution. For example, Na *et al.* [18] constructed an evolutionary game method to optimize service composition and deployment in the IoT cloud, the purpose is to reduce device energy consumption and increase device lifetime. Zhou *et al.* [19] proposed a multi-swarm parallel adaptive differential artificial bee colony (MPsaDABC) algorithm based on a bee foraging algorithm and an evolutionary

strategy for cloud manufacturing service composition and deployment problems. Guerrero *et al*. [20] constructed an improved non-dominated sorting genetic algorithm (GA-NSGA-II) for the cloud. This strategy mainly solves the deployment problems of microservices containers and considers the producer-consumer relationship between services in the deployment process. The optimization objectives are the threshold distance between microservices and containers, load balancing of computing resources, reliability of microservices and network distance between execution containers for microservices. Lin *et al*. [21] aimed at the problem of microservices deployment and resource management in the cloud. According to the dependency between services, an ant colony optimization (ACO) strategy for microservices deployment based on containers was designed. The goals are the network transmission overhead between services, resource utilization between nodes, and failure rate of microservices requests. Bouzary *et al*. [22] aimed to optimize the composition and deployment of cloud manufacturing services, and a hybrid grey wolf optimization algorithm was constructed based on the grey wolf optimization algorithm and evolutionary operators (crossover and mutation). Ma *et al*. [23] applied NSGA-III to design a knowledge-driven evolutionary algorithm for the deployment and start-up of microservice instances in the CDC. The resource utilization rate of the CDC is improved by optimizing five objectives: the average idle rate of computing and storage resources, balance rate of computing and storage resource loads, and actual idle rate of microservices. Mousa *et al*. [24] proposed an offloading strategy based on unmanned aerial vehicles (UAVs) in order to offload the tasks of IoT devices to the edge network server of UAV for execution, and ant colony optimization algorithm (ACO) is used to optimize the shortest path between IoT device clusters through which the UAV traverses, so as to reduce task delay and UAV energy consumption.

Second, container-based microservices deployment and scheduling strategies are widely used in IoT cloud environment. For example, Filip *et al*. [25] built an efficient real-time and dynamic microservice deployment strategy in a heterogeneous cloud edge environment. The purpose is to improve device utilization and reduce costs by optimizing the microservice allocation to meet different user requests. Liu *et al*. [26] established a multi-objective container scheduling strategy based on optimization problems such as Docker container resource utilization and task response time in the cloud. The strategy considers multiple objectives, such as CPU and memory utilization of physical nodes, container image transmission time, correlation between containers and physical nodes, and container aggregation. Tian *et al*. [27] modeled the problem of deploying microservices for mobile edge computing (MEC) environments in an IoT cloud as a Markov decision process (MDP). A distributed collaborative microservice deployment scheme (MIDA) was built based on deep reinforcement learning (DRL), and the main goal is to reduce the response delay of the service to improve the quality of service (QoS). Alwis *et al*. [28] used microservices to solve the problem of integrating enterprise business systems with the Industrial IoT cloud (IIoT), and adopted the k-Means clustering algorithm to optimize container deployment.

Third, owing to the effectiveness of particle swarm optimization (PSO) algorithm in multi-objective optimization problems, researchers have applied PSO as an optimization strategy in cloud environments. For example, Ramezani *et al*. [29] proposed a fuzzy particle swarm optimization strategy for the impact of VM migration in the CDC on application services. The main goal is to reduce the transmission time and power consumption and improve resource utilization. Kumar *et al*. [30] modeled an energy-saving scheduling strategy based on particle swarm optimization (PSO) technology to reduce the energy consumption of the CDC, task execution time, and cost in a cloud environment. Adhikari *et al*. [31] constructed a high-efficiency and energy-saving task scheduling strategy based on accelerated particle swarm optimization (APSO) for the deployment of execution containers in an IoT cloud environment. The optimization objectives of the strategy were task computing time, energy consumption, and CDC resource utilization. Mainak *et al*. [32] used accelerated particle swarm optimization (APSO) to establish an application service upload optimization strategy, which uploads resource-intensive tasks to a cloud data center (CDC) for processing in a fog cloud environment, aiming to overcome the resource constraints of fog devices. Huang *et al*. [33] built a task scheduling strategy based on particle swarm optimization (PSO) algorithm in a cloud environment. The purpose was to reduce the task completion time and resource cost and improve resource utilization. Wang *et al*. [34] formulated an improved competitive particle swarm optimization algorithm to improve the scheduling efficiency of cloud computing resources in the cloud. The optimization goals are the completion time of the tasks, power consumption, and load balance. Mousa *et al*. [35] constructed a graphics processing unit (GPU)-based [36] particle swarm optimization (PSO) parallel strategy. The optimization goal is to minimize the flight distance of UAV between IoT device clusters, so as to reduce the flight time and energy consumption of UAV.

Unlike the previous work, our optimization objectives are the service transmission overhead, execution container aggregation and resource load balancing of the CDC, and we use accelerated particle swarm optimization (APSO) to solve the deployment problem of the execution container for service. The deployment strategy proposed aggregates the execution containers of services in the service function chain (SFC) through the invocation relationship between services, aiming to reduce the service transmission overhead and improve the resource utilization of CDC.

III. BAND-AREA RESOURCE MANAGEMENT PLATFORM

To integrate enterprise applications, IoT applications and users in the IoT cloud, we constructed a virtual resource

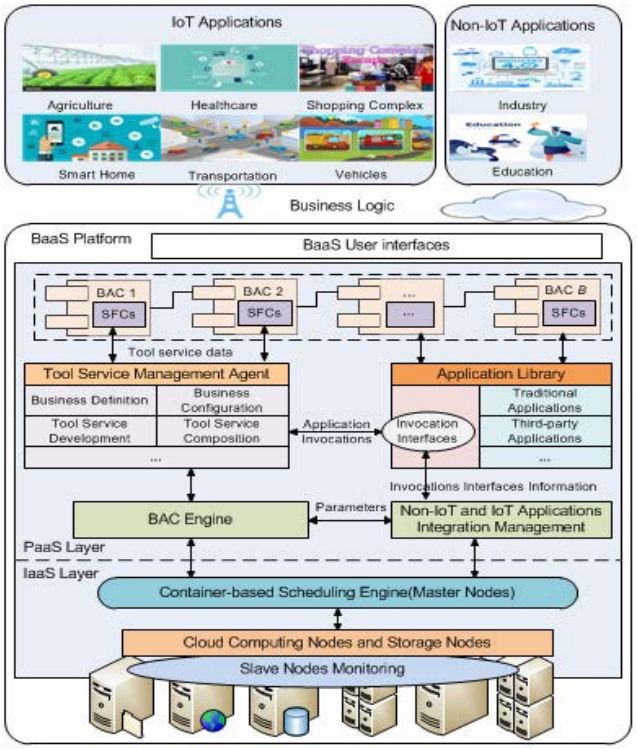


FIGURE 1. The structure of the band-area resource management platform.

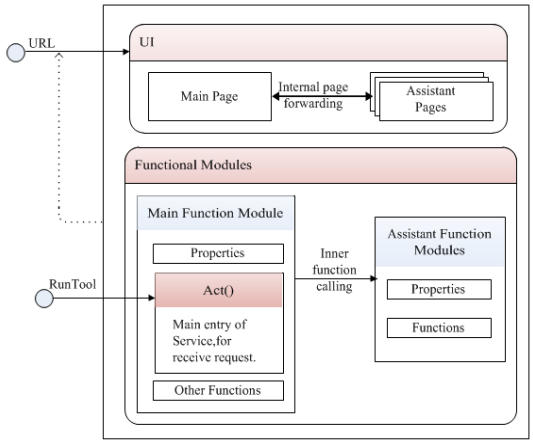


FIGURE 2. The structure of tool service model.

management platform based on Band-area application container (BAC) as a Service (BaaS) called ''wetoband'' [37]. BAC is a virtual operation area that can be customized by end users and is an aggregate composed of users, documents, tool services, messages, and relevant operation rules. BAC can be used to express a variety of things in reality, such as organizations, individuals, and projects [13]. The wetoband platform uses BAC as the core module and has two remarkable characteristics.

(1) For application system integration, users can deploy enterprise applications and IoT applications to BAC through various operations provided by BAC (such as add, delete, share, and co-share), and provide a running support environment for these application systems.

(2) For business collaboration, users can create BACs suitable for various business types according to their business requirements. On the one hand, in BAC, services are combined into a service function chain (SFC) through the invocation relationship between services to build an autonomous business sub-application system. On the other hand, because BACs can express organizations or individuals, the business relationship between organizations or individuals can be mapped to the collaborative relationship between BACs. Subsequently, various business sub-application systems are integrated into a complex and complete application system through the link and cooperation between BACs.

A. BaaS PLATFORM STRUCTURE

The structure of the Band-area resource management platform is illustrated in Figure 1. The platform can realize enterprise business logic based on tool services under the microservice architecture, and then integrate enterprise applications and IoT applications into a complete application system through the SFC of tool services in BAC. In addition, the BAC in the platform is abstracted into an independent ''virtual space'' and provides an invocation interface for service developers to facilitate business cooperation between BACs through the interaction between BACs. All computing and storage requests for tool services from SFCs in BAC are forwarded to the container-based scheduling engine of the IaaS layer through a BAC engine, and the scheduling engine performs resource scheduling.

End users can build, deploy, and migrate applications based on the operations provided by the BAC of the BaaS platform. The BaaS platform responds to user requests by managing the SFCs and allocating computing resources using a container-based scheduling engine. Without changing the existing enterprise application system, users can use the ''virtual space'' provided by BAC to add non-IoT and IoT applications in a self-service manner according to their own business requirements in the BaaS platform, which provides favorable application services and an operating environment for end users.

1) TOOL SERVICE MODEL

Microservices architecture is not invented but is derived from good practice [5]. However currently, there is no unified and standard definition model for microservices. To adapt to the BAC interface characteristics of the BaaS platform in the IoT cloud environment, we designed a fine-grained service model that can be deployed in BAC, called ''tool service.'' The tool service model is shown in Figure 2.

The main advantages of this design are as follows. First, the tool service provides two types of ports: the URL link port and RunTool function port. The URL link port is used for end users to start the execution tool service on the BaaS platform, and the RunTool functional port is the SDK function provided by the tool service development platform, which is convenient

for developers and other application services to call; Second, force the UI (user interface) in the template so that the end user can directly operate to use the functional services provided by the functional module; Third, in the IoT cloud, the code of a tool service may be executed in different places, which is beneficial to the deployment and integration of IoT applications. For example, a user page that controls a video switch may be executed on a remote web server, whereas the control code is executed on the embedded video system.

Category theory [38] and process algebra [39] can rigorously characterize system specification, design, correctness proofs, etc. Based on this, we used basic knowledge to formally describe tool services.

Definition 1: Tool Service (TS) -A TS can be formally defined as $TS =< TInfo, TIN, TOUT, FD, BEH, \psi, FunM >$, where:

(1) $TInfo = \{TID, TName, URL, BID\}$ describes the basic information of the tool service. *TID* is the unique identification of the service, *TName* is the service name, *URL* represents the URL of the service package, *BID* is the BAC ID where the service is located;

(2) *TIN* denotes the input interface set for the tool service;

(3) *TOUT* is the output interface set for the tool service;

(4) $FD = \{< Obj, Act, Pre, Aff >\}$ is a set of functional feature description information for the service. *Obj* represents the object processed by the service, *Act* indicates the action to be performed to realize the function, *Pre* is the precondition, *Aff* indicates the effect after the action is executed;

(5) *BEH* represents the external behavior of the service, which is formally described by process algebra;

(6) $\psi : \{\alpha | \alpha \in BEH\} \rightarrow \{Op \in TIN \cup TOUT\}$ is the mapping relationship between the behavior and the interface. That is, the corresponding relationship between actions in behavior description and operations in the interface description;

(7) $FunM = \{UI, Mdu\}$ denotes the code or function body of the tool service, *UI* represents the user interface, and *Mdu* is the function module.

2) TOOL SERVICES RELATIONSHIP MODEL

In the practice of microservices, with extensive cooperation between services, a complex interoperability relationship appears, which is essentially a partnership, mainly reflected in the invocation between services, and through the invocation relationship of services to build applications. Generally, the more services and businesses involved in the application, the more complex the relationship. Considering the invocation cooperation mechanism between services, the following invocation relationship definitions exist.

Definition 2: According to the complexity of the interaction relationship between services, the invocation relationship between tool services can be divided into three categories.

(1) The pipeline relationship (PR) refers to the service TS_i calling service TS_j and waiting for the execution result value of service TS_j as the input value of TS_i, which can be described as $PR(TS_i, TS_j)$. In an application, a service can call multiple services. For example, a goods-query service can call a goods-price service and a goods-stock service simultaneously to obtain the price and stock quantity of goods.

(2) Production-consumer relationship (PCR) means that service TS_i frequently calls service TS_j, in order to avoid service TS_j consuming the computing resources of the physical node and the network resources in CDC due to repeated calculation and transmission, that the TS_j saves the execution result data to the cache and waits for service TS_i to find the result when needed. Here, TS_j is the producer of data and TS_i is the consumer of data, which is formalized as $PCR(TS_i, TS_j)$. For example, a video-display service calls a video-capture service to obtain video data.

(3) The random-handshake relationship (RHR) indicates the simultaneous operation (i.e., concurrent execution) of services TS_i and TS_j, after the operation of TS_j is completed, notify TS_i through a coordination mechanism (e.g., message), and TS_i receives the notification and obtains the execution result of TS_j, which is expressed as $RHR(TS_i, TS_j)$. For example, in the process of confirming the receiving address in the random purchase of goods, send a confirmation message on the message board of BAC; after receiving the information, the customer clicks confirm.

Here, the service that makes a service calls (i.e., request) to the outside is ''master service,'' and the called service is ''slave service.'' The invocation relationship set between the services is described as $SR = \{PR, PCR, RHR\}$. Developers can convert their business flow into service function chain (SFC) on the BaaS platform based on the three service invocation relationships. We adopt the definition method in [40] to define SFC.

Definition 3: Service Function Chain (SFC) -An SFC is an application chain composed of n ($n \geq 1$) services according to the invocation relationship between services. By definition, service function chains are allowed to be combined recursively. The SFC can be represented as $SFC = <TSs, Edgs>$, where:

(1) $TSs = \{TS_j \mid TS_j \in setof(TSs, SFC), 1 \leq j \leq K \text{ or } j = s \text{ or } j = e\}$ represents the tool service set (i.e., ''node set'');

(2) $Edgs = \{(TS_i, TS_j) \mid TS_i, TS_j \in setof(TSs, SFC), 1 \leq i, j \leq K \text{ or } i = s \text{ or } j = e, i \neq j\}$, $Edgs \in Typeof(SR)$ is the edge set that describes the master-slave relationship between two services.

The benefits of an SFC defined based on the invocation relationship between services are as follows: (1) Reusability: the invocation relationship between services in the SFC can be stored in an XML file to generate an SFC template, namely the tool service suite, which can be reused in multiple organizations or personal BACs; (2) For deployment, services in the same SFC mean that they have close interaction. Deploying the execution containers of services in an SFC to the same physical node or CDC as much as possible can reduce the communication time. Note that we apply SFC to execution container deployment in the next section; (3) For composition: the SFC is deployed in BAC to form a sub-application system, and then multiple SFCs are combined into larger

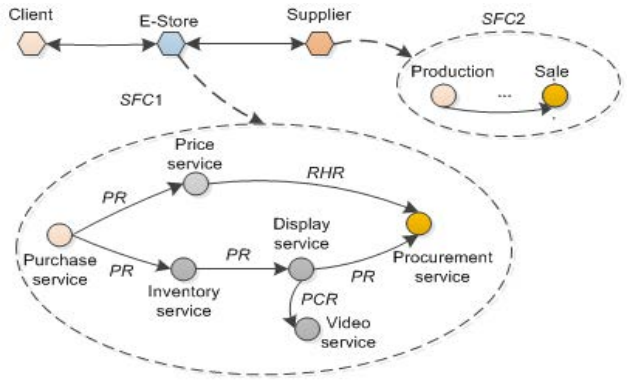


FIGURE 3. The supply-chain system based on SFC-chain and BAC-chain.

and complex application systems through the cooperation between BACs.

According to the above definition, for example, the supply-chain management system includes 3 organizations, and the corresponding BAC of each organization is the supplier BAC, E-store BAC, and client BAC. The dashed arrow connects the SFCs, in the E-store BAC, there is an SFC formed by shopping, price, inventory, video-display, video-collection, and purchase services. The supply-chain system integrated by SFC-chain and BAC-chain is shown in Figure 3.

Theorem 1: Tool Service Category (*TSC*)- The tool service category *TSC* is formed by considering the tool service in the SFC as the object and the arrow between the services as the morphism.

Proof: Suppose TS_i is a tool service of the SFC-chain, $ObjTSC = \{TS_i | i{>}0,\ i \in Z\}$ be all objects set in the category *TSC*. $TS_i \rightarrow TS_j$ is the arrow between the tool services in the SFC-chain, and $MorTSC = \{TS_i \rightarrow TS_j | i, j{>}0;\ i, j \in Z\}$ represents all morphisms set in the category *TSC*. Let the domain function be $dom : MorTSC \rightarrow ObjTSC$, co-domain function be $cod : MorTSC \rightarrow ObjTSC$, and composition function be $\circ : MorBC \times ObjBC \rightarrow MorBC$. The system $TSC = (ObjTSC, MorTSC, dom, cod, \circ)$ is a category that is proven as follows:

(1) Suppose $\forall TS_i, TS_j, TS_k \in ObjTSC, \exists u : TS_i \rightarrow TS_j, v : TS_j \rightarrow TS_k \in MorTSC$, then $v \circ u : TS_i \rightarrow TS_k \in MorTSC$, hence $dom(v \circ u) = TS_i = dom(u)$, $cod(v \circ u) = TS_k = cod(v)$, so that matching properties is satisfied.

(2) Let $w : TS_k \rightarrow TS_m \in MorTSC$, then $w \circ v : TS_j \rightarrow TS_m \in MorTSC$, and $(w \circ v) \circ u : TS_i \rightarrow TS_m \in MorTSC$, thus $(w \circ v) \circ u = w \circ (v \circ u)$, so that composition properties is satisfied.

(3) For $\forall TS \in ObjTSC$, there is a unique identity 1_{TS}, such that $dom(1_{TS}) = cod(1_{TS}) = TS$. For $\forall u \in MorTSC$, if $dom(u) = TS$ then $u \circ 1_{TS} = u$; If $cod(u) = TS$, then $1_{TS} \circ u = u$, so that identity properties is satisfied.

From the definition of the category [38], it can be seen that the service in the SFC-chain as the object, and the arrow between services as the morphism, which forms the tool service category, that is, the system *TSC* is a category. □

The tool service object is the node in the *TSC* category diagram, and the tool service morphism is a directed arc on the category diagram that reflects the composition and execution sequence of services. Here, service morphism is used to describe the invocation relationship between services, $\forall u \in typeof(SR)$. Services and service morphisms are linked to form an SFC. In the microservice architecture, applications are templates generated through the composition of microservices. According to the workflow patterns, service composition methods include sequential, parallel, selection, and circulation [41]. Selection and circulation are based on sequential and parallel methods. Only sequential and parallel compositions are described below.

Definition 4: Sequential composition co-limits for tool service -Given two tool services $TS_1 =< TInfo_1, TIN_1, TOUT_1, FD_1, BEH_1, \psi_1, FunM_1 >$ and $TS_2 =< TInfo_2, TIN_2, TOUT_2, FD_2, BEH_2, \psi_2, FunM_2 >$. The sequence composition co-limits of TS_1 and TS_2 are given by the identification $TS_c =< TInfo, TIN, TOUT, FD, BEH, \psi, FunM >$ and two synthetic morphisms $f : TS_1 \rightarrow TS_c$ and $g : TS_2 \rightarrow TS_c$, where:

(1) $Tinfo = \{TID, TName, URL, BID\}$ denotes the basic information of the composite service. *TID* is the unique identifier of the composite service, *TName* is the name of the composite service, *URL* represents the URL of the composite service, and *BID* indicates the BAC identifier;

(2) $TIN = TIN_1$;

(3) $TOUT = TOUT_1 \cup TOUT_2 - Inner(TOUT_1, TOUT_2)$ is the external output interface of the composite service that no longer includes internal ports;

(4) $FD = FD_1 \cup FD_2$;

(5) $BEH = BEH_1 \parallel BEH_2 \setminus \{\alpha | \alpha \in Inner(BEH_1, BEH_2)\}$, where $Inner(BEH_1, BEH_2)$ represents the internal behavior event of the composite service;

(6) $\psi = \psi_1 \cup \psi_2$;

(7) $FunM = FunM_1 \cup FunM_2$.

Definition 5: Parallel composition co-limits for tool service -Given two tool services $TS_1 =< TInfo_1, TIN_1, TOUT_1, FD_1, BEH_1, \psi_1, FunM_1 >$ and $TS_2 =< TInfo_2, TIN_2, TOUT_2, FD_2, BEH_2, \psi_2, FunM_2 >$. The parallel composition co-limits of TS_1 and TS_2 are given by the identification $TS_c =< TInfo, TIN, TOUT, FD, BEH, \psi, FunM >$ and two synthetic morphisms $f : TS_1 \rightarrow TS_c$ and $g : TS_2 \rightarrow TS_c$, where:

(1) $Tinfo = \{TID, TName, URL, BID\}$;

(2) $TIN = TIN_1 \cup TIN_2$;

(3) $TOUT = TOUT_1 \cup TOUT_2$;

(4) $FD = FD_1 \cup FD_2$;

(5) $BEH = BEH_1 \parallel BEH_2 \setminus \{\alpha | \alpha \in Inner\,(BEH_1, BEH_2)\}$;

(6) $\psi = \psi_1 \cup \psi_2$;

(7) $FunM = FunM_1 \cup FunM_2$.

The difference between Definitions 4 and 5 lies in items (2) and (3). The input interface of the sequential composition service TS_c is the interface of the service TS_1, and the output interface is the output interface of the service TS_2 except the interface that interacts with the service TS_1; The input/output

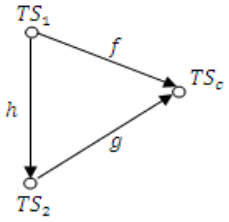


FIGURE 4. The co-limits.

interface of the parallel composition service is the union of the input/output interfaces of the two services. The co-limits of the sequential and parallel compositions of tool services are shown in Figure 4.

In addition to the composition of services, the BaaS platform provides a series of services for building non-IoT and IoT application systems including service design, service development, service deployment, service registration, service discovery, system monitoring, IoT device management, and infrastructure design. Furthermore, the BaaS platform supports the interface connection between the tool service and BAC and monitors the business system through project management (such as service development progress control, service defect testing, QoS detection, and service publishing). The BaaS platform considers the heterogeneity of other application systems and provides interface support for collaboration and integration with other application systems.

B. TOOL SERVICES VIRTUALIZATION

In the BaaS platform architecture, tool services must be virtualized to realize the transformation from business logic to an application system. The tool service virtualization system shown in Figure 5 includes the following.

(1) The infrastructure layer uses IT infrastructure (e.g., CPU, storage, network) to provide computing services to users in the form of VM or container (such as docker), and non-IoT and IoT applications are distributed across physical nodes in the CDC in the form of tool service execution packages.

(2) The tool service scheduling layer is responsible for monitoring and maintaining the operation of the physical node, and deploying the execution container instance of service to the appropriate computing node through the scheduling engine and load balancing strategy according to the application call request. Note that we propose an execution container deployment strategy for the tool services in Section V.

(3) The tool service conversion layer is primarily responsible for converting the non-IoT and IoT business applications of users into tool service templates.

(4) Tool service composition layer: Based on non-IoT and IoT business requirements, the tool services are combined into an SFC according to the invocation relationship between services to build sub-application systems and provide them to users.

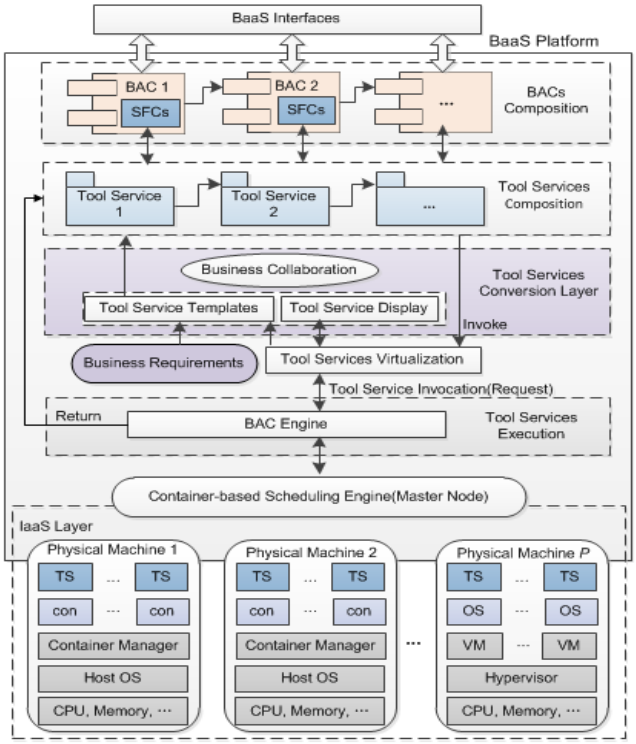


FIGURE 5. The tool service virtualization system.

(5) BAC composition layer: According to the business collaboration between organizations or individuals, it is mapped to the links between BACs, and a complete application system is constructed through the links between BACs.

The BaaS platform provides developers with programming interfaces, tool service template specifications and standards, so that developers can develop new SFCs according to business requirements and integrate them into existing business application systems; and the tool service is deployed to the computing node for execution through the container-based scheduling engine.

IV. PROBLEM STATEMENT

The system model and objective function are established in this section, and a multi-objective optimization model is proposed to solve the problem model. The main parameters and their descriptions are listed in Table 1.

A. SYSTEM MODEL

On the BaaS platform, users deploy their required non-IoT and IoT applications to a Band-area application container (BAC) through the operations provided by the BAC of the platform. Applications in BAC send resource requests to the scheduling engine through the BAC engine, and the container-based scheduling engine finds a suitable server to deploy the execution container of the service according to the number of user requests received by the tool service and the number of required resources. To reduce the response time of the service and improve the resource utilization of the CDC to

TABLE 1. Model parameters and description.

Element	Symbol	Description
Band-area Container	BA	BAC set
	$ba_l \in BA$	BAC with identifier l
User	$User$	a set of user
	$User_u \in User$	user with identifier u
	$Ureq_u$	number of user requests
Application	App	application set
	$App_a \in App$	application with identifier a
	SFC_a	SFC of application App_a
Tool Service	TS	a set of tool service
	TS_R	master-slave relationship set between tool services
	$TS_i \in TS$	tool service with identifier i
	Req_CPU_i	amount of CPU requested by service i
	Req_Mem_i	amount of memory requested by service i
	$RThr_i$	invoked (requested) threshold
	$Tsreq_i$	invoked (requested) quantity
	$Cons_i$	number of execution container instances
	$(TS_i, TS_j) \in TS\_R$	master-slave relationship between tool service TS_i and TS_j
	$Tsreq_{ij}$	number of services TS_j invoked by service TS_i
	TT_{ij}	amount of data transferred between services TS_i and TS_j
	$Consize_i$	size of execution container image
	IT_i	transmission overhead of execution container image
Cloud Data Center (CDC)	PM	physical nodes set in CDC
	$pm_k \in PM$	physical node with identifier k
	$sizeof(PM)$	number of physical nodes
	pm_CPU_k	total amount of CPU of physical node pm_k
	pm_Mem_k	total amount of memory of physical node pm_k
	$Dist_{kq}$	network distance between physical nodes pm_k and pm_q

balance the interests between users and cloud providers, our strategy is to aggregate the execution containers corresponding to tool services in the same SFC to the same physical node (computing server) or CDC as much as possible.

Suppose there are U users $User = \{User_1, User_2, \ldots, User_U\}$. The features of each user are formalized as $User_u =< Ureq_u, UPos_u >$ and $u \in \{U\}$, where $Ureq_u$ represents the number of user requests for applications and $UPos_u =< U_x, U_y >$ indicates the location of the user. The user created B BACs $BA = \{BA_1, BA_2, \ldots, BA_B\}$, and deployed A applications $App = \{App_1, App_2, \ldots, App_A\}$ to the BACs, the features of each application are described as $App_a =< Ureq_a, SFC_a >$ and $a \in \{A\}$. Here, $SFC_a =< TSs_a, Edgs_a >$ is the service function chain of application App_a. $TSs_a = \{TS_1, TS_2, \ldots, TS_N\}$, the features for each tool service are expressed as $TS_i =< Tsreq_i, Req\_CPU_i, Req\_Mem_i, RThr_i >, i \in \{N\}$. $Tsreq_i$ is the number of calls to the tool service TS_i; Req_CPU_i and Req_Mem_i are the amount of CPU and memory resources required for one call to the tool service, respectively; $RThr_i$ is the request threshold for tool service, that is, the maximum number of service requests.

At the IaaS layer, multiple cloud data centers (CDCs) connected by an internal high-speed network are distributed at different locations in two-dimensional space. Each CDC contains multiple physical nodes, which are represented by a set of $PM = \{pm_1, pm_2, \ldots, pm_P\}$. For each physical node, is formalized as $pm_k =< pm\_CPU_k, pm\_Mem_k, pm\_Pos_k >$, $k \in \{P\}$. pm_CPU_k and pm_Mem_k are the CPU and memory resources, respectively, pm_Pos_k is the physical node location.

Tool service TS_i is encapsulated and executed in the execution container con_l, which is expressed as $alloc(TS_i) \equiv con_l$ ($l \geq 1$). To effectively utilize resources and balance the load, when the actual number of service requests $RTR_i = Ureq_a \times Tsreq_i$ exceeds the threshold $RThr_i$, additional execution container instances must be started. The scaling number of execution container instance is $Cons_i = \lceil RTR_i / RThr_i \rceil = \lceil (Ureq_a \times Tsreq_i)/RThr_i \rceil$. However, starting each execution container instance consumes a certain amount of resource. Therefore, multiple execution containers must be deployed in each physical node, which is expressed as $alloc(con_l) \equiv pm_k$.

Definition 6: Deployment strategy for the execution container instance of tool service (TSDS) -For P heterogeneous physical nodes PM in the CDC, N execution container instances of tool services are deployed, which are defined as:

$$Stra(TS, PM) = [x_{ik}]_{N \times P} \tag{1}$$

here, $x_{ik} = [0, 1]$ indicates the state of the execution container instance of the tool service TS_i deployed on physical node pm_k, $1 \leq i \leq N$ and $1 \leq k \leq P$. $x_{ik} = 1$ means allocated; otherwise, it is not allocated.

B. OBJECTIVE FUNCTION

Here, we establish the following three objective functions: service transmission overhead, resource load balancing of CDC, and execution container aggregation.

1) SERVICE TRANSMISSION OVERHEAD

In CDC, because the server nodes use high-speed network interconnections, the noise interference of the channel can be ignored. Therefore, the data transmission overhead between services depends on: the number of calls (requests) between services, amount of data transmission between services, size of the execution container image, and the network distance between the physical nodes hosting the execution containers.

In an SFC, the interaction between the master and slave services through calls generates data traffic, that is, the amount of data transmission TT_{ij}, $1 \leq i, j \leq N$. In a two-dimensional space (X, Y), the distance $Dist$ between the execution containers of interacting services is the network distance between the physical nodes hosting the execution containers. The formula is given by Equation (2):

$$Dist_{kq} = \begin{cases} \sqrt{(X_k - X_q)^2 + (Y_k - Y_q)^2}, & k \neq q; \\ 0, & otherwise; \end{cases} \quad k, q \in \{P\}. \tag{2}$$

Considering that the execution containers of master-slave services can be deployed in multiple physical nodes, we use

the average transmission overhead of multiple container pairs between master and slave services to calculate the transmission overhead between the two services, which is defined as:

$$f_1(X) = \sum_{k=1}^{P}\sum_{i=1}^{N}\left\{\frac{x_{ik}}{Cons_i}\sum_{(TS_i,TS_j)\in SFC_a}\sum_{l=1\wedge alloc(con_l)\equiv pm_q}^{Cons_j}\frac{x_{jq}}{Cons_j}(Tsreq_{ij}\times Dist_{kq}\times TT_{ij}) + Dist^i_{kp}\times IT_i\right\} \tag{3}$$

where $Dist^i_{kp}$ $(k \neq p)$ is the distance from server pm_p where the execution container image of tool service TS_i is stored to deployment node pm_k, IT_i is the size of the execution container image for tool service TS_i.

2) RESOURCE LOAD BALANCING OF CDC

In the CDC, the consumption of CPU and memory resources of all physical nodes should be in a balanced state to avoid high-load and low-load problems. The high load makes the physical node unable to respond to the service request in time, and a low load will causes a waste of resources. To balance CDC load, the load must be evenly distributed among the physical nodes.

Here, the CPU and memory resource utilization of the physical node pm_k is the ratio of used resources to total resources, which is calculated using Equation (4):

$$\begin{aligned} RU_k^{CPU} &= \frac{\sum_{i=1}^{N} x_{ik}\frac{Ureq_a\times Tsreq_i}{Cons_i}\times Req\_CPU_i}{pm\_CPU_k} \\ RU_k^{Mem} &= \frac{\sum_{i=1}^{N} x_{ik}\frac{Ureq_a\times Tsreq_i}{Cons_i}\times Req\_Mem_i}{pm\_Mem_k} \end{aligned} \tag{4}$$

The load-balancing rate of the CPU and memory resources between the physical nodes is defined as follows:

$$\begin{aligned} Bal\_cpu &= \sum_{k=1}^{P}\left(RU_k^{CPU} - \overline{RU^{CPU}}\right)^2 \\ Bal\_mem &= \sum_{k=1}^{P}\left(RU_k^{Mem} - \overline{RU^{Mem}}\right)^2 \end{aligned} \tag{5}$$

where $\overline{RU^{CPU}}$ and $\overline{RU^{Mem}}$ are the average values of the CPU and memory resource utilization for the CDC, respectively:

$$\begin{aligned} \overline{RU^{CPU}} &= \frac{\sum_{i=1}^{N} x_{ik}\frac{Ureq_a\times Tsreq_i}{Cons_i}\times Req\_CPU_i}{\sum_{k=1}^{P} pm\_CPU_k} \\ \overline{RU^{Mem}} &= \frac{\sum_{i=1}^{N} x_{ik}\frac{Ureq_a\times Tsreq_i}{Cons_i}\times Req\_Mem_i}{\sum_{k=1}^{P} pm\_Mem_k} \end{aligned} \tag{6}$$

To balance the resource load rate of the CDC and improve resource utilization, we select the physical node with the greatest resource load pressure as the load balancing measure. This is formulated in Equation (7):

$$f_2(X) = \frac{1}{Bal\_cpu + Bal\_mem}\max_{1\le k\le P} \times \max(RU_k^{CPU} - \overline{RU^{CPU}}, RU_k^{Mem} - \overline{RU^{Mem}}) \tag{7}$$

3) EXECUTION CONTAINER AGGREGATION

In the process of application running, on the one hand, there is data interaction between the services of the service function chain (SFC), which may consume considerable a lot of network resources. To reduce the data transmission overhead between tool services and improve the response speed of the application, the execution container of the service with the invocation relationship should be deployed to the same physical node or same CDC as much as possible. On the other hand, in the case of sufficient CDC resources, the execution containers where computing tasks are located are concentrated to some nodes to improve resource utilization, so that some idle nodes can be shut down to save energy consumption. Thus, an aggregation of execution containers was generated. Here, we propose using the edge distance between services as a metric to measure the similarity between execution containers [26].

At each physical node, the execution containers of one or more services in the SFC may be deployed. The aggregation degree of containers is measured by calculating the edge distance between services in the SFC. The larger the edge distance, the more aggregated the containers, and the shorter the data interaction time between containers. In SFC, the edge weights between adjacent services $(TS_i, TS_j) \in Edges$ are used as edge distances, i.e., $Edist_{ij} = |w_{ij}|$, $i, j \in \{N\}$. However, the edge weights between services are related to the invocation relationship between services, that is, different invocation relationships can set different edge weights.

For example, suppose there is an SFC in the physical node: $TS_1 \rightarrow TS_2 \rightarrow TS_3$, let $SR(TS_1, TS_2) = PR$, the edge weight $w_{12} = 2$; $SR(TS_2, TS_3) = PCR$, $w_{23} = 3$; then $Edist_{SFC_a} = 5$. Therefore, in CDC, the edge distance is calculated using Equation (8):

$$Edist_{CDC} = \sum_{k=1}^{P}\sum_{x_{ik}=1\wedge x_{jk}=1\wedge i\neq j} SR(x_{ik}, x_{jk})\times w_{ij} \tag{8}$$

where, w_{ij} is the weight of the edge, $SR(x_{ik}, x_{jk})$ is defined as:

$$SR(x_{ik}, x_{jk}) = \begin{cases} x_{ik}x_{jk}, & if\,(TS_i, TS_j) \in Edgs; \\ 0, & otherwise. \end{cases} \tag{9}$$

Here, we use the reciprocal of the edge distance as the similarity between the containers, as shown in Equation (10):

$$f_3(X) = \frac{1}{Edist_{CDC}} \tag{10}$$

C. PROBLEM MODEL AND CONSTRAINTS

Aiming at the three objective functions proposed above and in order for these objectives to be optimal simultaneously, the execution container deployment model based on the service function chain is described as a multi-objective optimization problem, and the model is defined as follows:

$$minimize\, f(X) = (f_1(X), f_2(X), f_3(X)) \tag{11}$$

$$s.t. \sum_{i=1}^{N} x_{ik} Req\_CPU_i \leq pm\_CPU_k, \quad \forall pm_k \tag{12}$$

$$\sum_{i=1}^{N} x_{ik} Req\_Mem_i \leq pm\_Mem_k, \quad \forall pm_k \tag{13}$$

$$\sum_{i=1}^{N} x_{ik} RTR_i \leq RThr_i \tag{14}$$

$$\sum_{i=1}^{N} x_{ik} = 1 \tag{15}$$

$$\sum_{i=1}^{P} x_{ik} \geq 1, \quad i \in \{N\},\ k \in \{P\} \tag{16}$$

where $X = [x_{ik}]_{N \times P}$ ($X \in \Omega$) is the decision variable of the deployment model.

Equation (11) is the optimization objective, including the service transmission overhead, resource load balancing for the CDC, and execution container aggregation.

Equations (12) - (16) represent the 5 constraints of the problem model. Equations (12) and (13) are the CPU and memory resource constraints of the physical nodes, respectively. Owing to the limited CPU and memory resources of the physical nodes, the amount of CPU and memory resources consumed by executing container instances should be less than the total amount of corresponding resources. Equation (14) is the request constraint of the service, that is, the actual number of requests to the service should be less than the service request threshold. Equation (15) is the mutually exclusive constraint of services, which is mainly used to prevent multiple container instances of the same service from competing for resources, and only one execution container instance of the same service can be hosted in the same physical node. Equation (16) is the integrity constraint of the service, which ensures that at least one execution container instance of each tool service is hosted in the CDC.

The above optimization problem is aimed at reducing the service transmission overhead, balancing the resource load of the CDC, and aggregating the execution containers of services in the SFC, and considering the total amount of resources, service request, service deployment as constraints. As can be seen from the problem model, this is an NP-complete problem. It is difficult to find an optimal solution using a polynomial strategy, and the meta-heuristic method is a good method. Particle swarm optimization (PSO) algorithm is a meta-heuristic approach. In particular, accelerated particle swarm optimization (APSO) improves the calculation method of particle position and velocity, and realizes the rapid convergence of the global optimal solution by reducing randomness [31]. Therefore, this study proposes an improved accelerated particle swarm optimization (APSO) strategy, which uses the crowding distance [42] and fitness function value to evaluate the quality of the solution, and finds the Pareto front through multiple iterations to effectively obtain the optimal solution.

V. PROPOSED ACCELERATED PARTICLE SWARM OPTIMIZATION ALGORITHM

In this section, we introduce the proposed APSO-TSDS strategy, fitness function, and the algorithm design and implementation.

A. OVERVIEW OF APSO

The accelerated particle swarm optimization (APSO) algorithm is an improved version of the particle swarm optimization (PSO) algorithm. The PSO was originally proposed by Kennedy and Eberhart [43]. It is a swarm intelligence optimization algorithm developed by simulating the foraging behavior of birds. It has been applied to the field of resource optimization and scheduling in cloud computing environment.

During bird swarm foraging, individuals find food targets in the search space by constantly adjusting their flight trajectory. The factors that affect the flight trajectory of individuals are their own experiences (i.e., personal best) and swarm experiences (i.e., global best). The personal best (Pb) position is the best position that an individual can find, and the global best (Gb) position is the best position that all individuals in the swarm can find. The entire bird swarm tends to move toward the global best position. Through the continuous movement of individual positions, that is, continuous iterations, the swarm moves toward the target.

However, to improve the convergence speed of particle swarm optimization algorithm in global optimization, researchers have constructed a variant of the accelerated PSO, namely the accelerated particle swarm optimization algorithm (APSO) [30], which uses only the global optima for velocity updates. In an n-dimensional objective space, there are M particles are formalized as $Pop = \{pop_1, pop_2, \ldots, pop_M\}$. Let $X_m = \{x_m^1, x_m^2, \ldots, x_m^n\}$ and $V_m = \{v_m^1, v_m^2, \ldots, v_m^n\}$ be the position and velocity vectors of particle $pop_m (m \in \{M\})$, respectively. At any iteration/time t, the velocity vector of each particle is updated using Equation (17):

$$V_m(t+1) = V_m(t) + \alpha\varepsilon + \beta(Gb - X_m(t)) \tag{17}$$

where $\varepsilon \in [0, 1]$ denotes a random vector uniformly distributed between 0 and 1, $\alpha \in [0.1, 0.5]L$, L is the variable scale; $\beta \in [0.1, 0.7]$. The position vector is then updated using Equation (18):

$$X_m(t+1) = (1-\beta)X_m(t) + V_m(t+1) \tag{18}$$

B. PARTICLE REPRESENTION

In the APSO, every particle in the swarm is a solution of the problem. Based on the problem model proposed in Section IV, we used a string to express the particles. Each particle is described as an array, which is the tool service set, and the index is the identifier of the service. The content of each

TABLE 2. The structure of particle pop_i.

Particle pop_i	TS_1	TS_2	...	TS_K
	{5, 3, 6}	{4, 2}	...	{1, 3, 5, 7}

tool service is the allocation list of execution containers, that is, the list of physical nodes that deploy execution containers according to the number of containers. To prevent multiple containers of the same service from preempting resources on the same physical node, in this agreement, the same physical node ID can only appear once in the node list. For example, there are K tool services, and the structure of particle $pop_i(i \in M)$ is shown in Table 2.

For the identifier of the physical node $pm_j(j \in \{P\})$ where the execution container instance of any service $TS_k(k \in \{K\})$ in particle pop_i is located, we adopt a discrete binary vector (i.e., 0-1 coding) to represent [41]. Assuming that there are $P = 120$ physical nodes in the CDC, the encoding length of the physical nodes is $nVar = \lceil \log_2 P \rceil$, and the conversion to a binary vector is [1,1,1,1,0,0,0]. For example, the tool service TS_1 was deployed in three physical nodes. The deployment list was {5, 3, 6}, which was converted into a binary array as shown in Equation (19):

$$X_{TS_1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{19}$$

Then, the position of particle pop_i is expressed as $X_i = (X_{TS_1}, X_{TS_2}, \ldots, X_{TS_K})$, X_i is the candidate solution of the problem model.

Because the state value of the particle position is limited to 0 and 1 in the state space, we mapped a transition velocity for each state value 0/1. The transition velocity between position states 0 and 1 was calculated using a sigmoid function [44]:

$$sigmoid(v) = \frac{1}{1 + exp(v)} \tag{20}$$

$$x = \begin{cases} 1, & rand < sigmoid(v); \\ 0, & otherwise. \end{cases} \tag{21}$$

where v is the conversion velocity between position states 0 and 1.

In APSO, changing the moving velocity and direction of the particle pop_i and approaching the optimal solution in the target space are key problems. Based on the previous analysis of particle position representations, we refined the particle position and velocity formulations.

At any iteration/time t, the updated formula of the improved velocity for the particle is defined by Equation (22):

$$v_m(t+1) = v_m(t) + \alpha(1, nVar) + \beta(x_m^*(t) - x_m(t)) \tag{22}$$

where $m \in \{nVar\}$, x_m^* is the corresponding state position of the optimal individual. The velocity update formula is helpful for guiding particles to search for the global optimal particle position to further improve convergence. The updated formula for the improved position of the particle is given by Equation (23):

$$x_m(t+1) = \begin{cases} 1, & rand < sigmoid(v_m(t+1)); \\ 0, & otherwise. \end{cases} \tag{23}$$

C. THE FITNESS FUNCTION AND OPTIMAL SOLUTION

Here, we adopt APSO technology to solve the deployment problem of the execution container for tool service, and deploy multiple execution containers for each service to appropriate physical nodes according to the objective optimization model proposed in Section IV.C.

First, in the process of deploying the execution container for the service, a normalization idea is adopted for each objective function; that is, the maximum and minimum values of each objective function of the particle pop_i are normalized. The advantage of normalization is that it can eliminate the influence of different amplitudes on the corresponding objective. The normalization function is as follows:

$$NF_n(X_i) = \frac{f_n(X_i) - f_n^{min}}{f_n^{max} - f_n^{min}} \tag{24}$$

where, $1 \le n \le 3$, f_n^{min} and f_n^{max} represent the maximum and minimum values of the nth objective of the solution, respectively.

Second, the weighted sum method proposed by Zadeh [45] was used to aggregate each objective function into a fitness function, which is formulated as follows:

$$fit(X_i) = \sum_{n=1}^{3} \alpha_n NF_n(X_i) \tag{25}$$

where, $\alpha_n \in [0, 1]$ and $\sum_{n=1}^{3} \alpha_n = 1$. α_n is the weight of each objective function, which can be set according to the different business objectives.

Here are several concepts related to objective optimization: (1) The dominant relationship between individuals, let p and q be any two different individuals in the particle swarm Pop, if the each objective function value of individual p is better than the value of individual q ($f_n(p) \le f_n(q)$), then individual p dominates individual q ($p \prec q$), that the individual p is called non-dominated or non-inferior. Similarly, if the subset of the objective function values of individual p' is better than the corresponding objective function values of individual q', and the values of other objective subsets are worse than the values of individual q', then individual p' does not dominate individual q'. (2) The Pareto optimal solution (front) is an individual that is not dominated by other individuals at any iteration/time t. In the Pareto optimal solution set, each individual (solution) optimizes one or more objectives in the objective space. (3) The crowding distance is the distance between the non-dominated solution and Pareto optimal solution in the objective space [42].

We select the archive set through the dominant relationship between individuals, that is, the Pareto set, and then the global best individual with the smallest crowding distance

and fitness function value is selected from the archive set. The established fitness function helps evaluate the quality of the solution (particles) during each iteration to find the Pareto optimal solution as much as possible.

D. DEPLOYMENT STRATEGY FOR EXECUTION CONTAINER

The APSO algorithm realizes optimization by simulating the working principle of bird swarm foraging. During optimization, the particles move toward the optimal objective by continuously adjusting their position and velocity. An overview of the deployment strategy of the execution container for tool services based on the APSO is as follows:

1) PARTICLE SWARM INITIALIZATION

maximum and minimum flight velocity $[-vmax, vmax]$, velocity control factors α and β, coding length $nVar$, archive set size $nArch$ and iteration scale N_max. Then, the particle swarm $Pop = \{pop_i | 1 \leq i \leq M\}$ is generated, where M is the size of the swarm.

2) ITERATIVE PROCESS

(a) Calculate the objective function value $f^i = [f_1, f_2, f_3]$ and fitness value fix^i of each particle pop_i; Then, the personal best position is updated, and the archive set Arch is selected based on the dominance relationship;

(b) Select the particle with the smallest crowding distance and fitness value from the archive set $Arch$ as the global optimal individual $Leader$;

(c) Update the velocity V_i of each particle;

(d) According to the constraints of the problem model, determine whether the physical node where each tool service TS_k in the particle pop_i is located satisfies the allocation of the execution container. If satisfied, then complete the allocation; otherwise, find the next physical node. Because the execution container of the service TS_k selects different nodes each time and allocates $Cons_k$ times; Therefore, only when all execution containers of service TS_k are allocated to different physical nodes, the node allocation of service TS_k is completed, and the allocation of the execution containers of all services is completed according to the allocation principle;

(e) Update the position X_i of each particle and repeat Step (a).

3) ALGORITHM TERMINATION

When all particles complete the node matching of all execution containers for all tool services, it is called an iterative process, update the position and velocity of the particles and enter the next iteration until the iteration scale N_max, terminate the algorithm.

E. IMPLEMENTATION OF DEPLOYMENT STRATEGY

The pseudo code of APSO for the Deployment Strategy of the Tool Service Execution Container (APSO-TSDS) proposed in this study is presented in Algorithm 1.

APSO for Deployment Strategy of Tool Service Execution Container.

APSO-TSDS Algorithm 1: Deployment strategy for execution container.

Input:
$TS = \{TS_1, TS_2, \ldots, TS_N\}$;
$PM = \{pm_1, pm_2, \ldots, pm_P\}$;
$TS\_R = \{(TS_i, TS_j, ER) | TS_i, TS_j \in TS \wedge ER \in SR\}$;
$N\_max, nVar, M, vmax, nArch, \alpha, \beta$;
Output:
$TS\_PM = \{(TS_i, PM_i) | PM_i \subseteq PM\}$;

$t \leftarrow 1$;
$[Pop, f] \leftarrow$ iniPopulation(M);
$fit \leftarrow$ Calfitness(Pop);
$Arch \leftarrow$ DetermineDomination(Pop, f);
while ($t \leq N\_max$) **do**
  $leader \leftarrow$ SelectLeader($Arch, fit$);
  for each pop_i **do**
    for each TS_k **do**
      while ($c \leq Cons_k$) **do**
        while (true) **do**
          for $m = 1$ to $nVar$ **do**
            Update the velocity v_m^c by Equation (22);
            if ($v_m^c < -vmax$)
              $v_m^c \leftarrow -vmax$;
            elseif ($v_m^c > vmax$)
              $v_m^c \leftarrow vmax$;
            end if
            Update the position v_m^c by Equation (23);
          end for
          $j \leftarrow$ transform2to10(x^c);
          if ($Req\_CPU_k \leq Re\_CPU_j \wedge$
            $Req\_Mem_k \leq Re\_Mem_j$)
            Update (k, j) to TS_PM;
            break;
          end if
        end while
      end while
    end for
    if Dominates($pop_i.f, pop_i.Best.f$)
      $pop_i.Best.X \leftarrow pop_i.X$;
    else
      $pop_i.X \leftarrow pop_i.Best.X$;
    end if
  end for
  $[f, fit] \leftarrow$ Calfitness(Pop);
  $Arch \leftarrow$ DetermineDomination(Pop, f);
end while
Return TS_PM;

F. ANALYSIS OF ALGORITHM COMPLEXITY

The complexity of the APSO-TSDS algorithm is primarily reflected in the selection of servers for the execution container

TABLE 3. The tool service stack of SFC-based application.

$Name$	$ID.$	SFC	Req_CPU_i	Req_Mem_i	$Tsreq_i$	$RThr_i$	$Cons_i$	$Consize_i$	IT_i
Shopping	TS_1	SFC_a	3.8	1.5	30	10	3	64.1	10.3
Estore-shelves	TS_2	SFC_a, SFC_c	1.5	13.6	55	11	5	48.3	9.1
Supplier	TS_3	SFC_a, SFC_b	4.2	5.7	45	9	5	180.5	19.2
Plan	TS_4	SFC_b	3.1	6.5	28	7	3	41.2	8.5
Production	TS_5	SFC_b	5.0	7.0	60	10	6	80.6	30.1
Packing	TS_6	SFC_b	3.2	5.1	30	6	5	50.0	10.0
Onlie-payment	TS_7	SFC_a, SFC_b	5.6	1.6	100	10	10	22.8	8.7
Video-monitoring	TS_8	SFC_b	50	55	10	2.5	4	55.4	10.2
Sale	TS_9	SFC_b	4.5	4.3	45	9	5	88.3	9.5
Goods-collection	TS_{10}	SFC_c	3.2	2.5	55	5	11	65.0	9.2
Express-Shipping	TS_{11}	SFC_a, SFC_b	3.1	4.3	60	10	6	37.6	10.5
Purchase	TS_{12}	SFC_c	6.3	4.5	70	7	10	63.2	19.8
Price	TS_{13}	SFC_c	7.0	6.8	35	6	6	56.1	9.3
Inventory	TS_{14}	SFC_c	6.5	7.5	35	5	7	32.7	10.4
Video-display	TS_{15}	SFC_c	18	50	20	2.5	8	52.9	18.1
Video-capturing	TS_{16}	SFC_c	20	50	20	6	4	65.8	9.25
Ordering	TS_{17}	SFC_c	6.3	3.5	70	8	9	52.5	8.6
Account	TS_{18}	SFC_c	6.2	2.1	20	7	3	68.3	10.6

of each service. Let N be the number of tool services and P be the number of servers. In steps 7 to 33 of the deployment strategy for the execution container, the outer loop needs to perform $O(M \times N)$ operations. In steps 10 to 25 of the internal loop, $Cons_k$ container instances need to be allocated for each service. When traversing the binary position, $O(nVar)$ operations need to be performed, and when judging whether the remaining resources of the execution server meet the service resource request, the worst case needs to perform P operations, and the best case only needs to perform 1 operation. Thus, the container instance selection server needs to perform operations $O(\sum_{c=1}^{Cons_k} nVar)$ (the best case) and $O(\sum_{c=1}^{Cons_k} P \times nVar)$ (the worst case). The total number of operations to be performed by all the particles is $O(M \times N \times \sum_{c=1}^{Cons_k} nVar)$ (the best case) and $O(M \times N \times (\sum_{c=1}^{Cons_k} P \times nVar))$ (the worst case). Considering the number of iterations N_max, the total time complexity of the APSO-TSDS algorithm is $O(N\_max \times (M \times N \times (\sum_{c=1}^{Cons_k} P \times nVar)))$ (the worst case) and $O(N\_max \times (M \times N \times (\sum_{c=1}^{Cons_k} nVar)))$ (the best case).

VI. EXPERIMENTAL EVALUATION

In this section, the performance of the APSO-TSDS algorithm proposed in this study is investigated by experimentally evaluating various QoS parameters. The experimental dataset was obtained from the Alibaba Tracker V2018 [46]. Based on the analysis of the real dataset of tracking V2018, 18 tool services were adopted and the service function chain (SFC) is constructed according to the invocation relationship between services to form a sub-application system. The effectiveness of the APSO-TSDS algorithm is examined by implementing different user requests for sub-applications in CDCs of different scales.

A. EXPERIMENTAL ENVIRONMENT

1) EXPERIMENTAL DATA SETUP

Tables 3 and 4 summarize the experimental data for the APSO-TSDS deployment strategy. Here, we consider an application that includes three SFCs: SFC_a, SFC_b, and SFC_c.

Table 3 lists the 18 tool service stacks included in the application. Among these, Video-monitoring, Video-display and Video-capturing are tool services developed for sensor video devices. Each line describes the basic information of the service, including the SFC in which the service is located, required resources *Req_CPU* and *Req_Mem*, invoked (requested) quantity of service *Tsreq*, invoked (requested) threshold *RThr*, number of execution container instances *Cons*, size of container image *Consize*, and image transmission overhead *IT*. Where $Cons = \lceil Tsreq/RThr \rceil$.

Table 4 shows the invocation relationship information between tool services in SFC when the application receives a unit ($\times 1.0$ times) of user service requests, including edge set *Edges*, edge relationship *ER* (i.e., invocation relationship), edge weight *Weight*, number of calls (requests) between services $Tsreq_{ij}$ and the amount of data transmission between services TT_{ij}. Where (TS_i, TS_j) represents the invocation relationship between the tool services of the SFC, (0,TS_i) is the edge where the client starts the tool service TS_i in BAC. The edge relationship and edge weight mapping between the tool services are (*RHR*, 1), (*PR*, 2), and (*PCR*, 3). Here, the edge weight is set according to the amount of data transmission between services. Because the producer-consumer relationship (*PCR*) has a large amount of data transmission, a larger weight is set; the pipeline relationship (*PR*) is second, and the random-handshake relationship (*RHR*) has the smallest weight.

2) EXPERIMENTAL PARAMETERS

Here, we designed a heterogeneous cloud data center (CDC) and set the parameters for the APSO-TSDS algorithm. To evaluate the performance and effectiveness of the APSO-TSDS algorithm, two clusters with different numbers of physical nodes were set up in the CDC, and their topology was constructed based on the CloudSim platform.

(1) Number of physical nodes of two types for CDC: $sizeof(CDC) = [120, 240]$;

(2) Physical nodes have three types of CPU resource capacities: pm_CPU_k =[200.0, 400.0, 800.0];

TABLE 4. The invocation relationship information between services in the application.

Edges	*ER*	*Weight*	$Tsreq_{ij}$	TT_{ij}	*Edges*	*ER*	*Weight*	$Tsreq_{ij}$	TT_{ij}
(0, TS_1)	RHR	1	45	0	(TS_5, TS_6)	PR	2	30	6.0
(TS_1, TS_2)	PR	2	50	4.1	(TS_6, TS_9)	PR	2	25	5.1
(TS_1, TS_7)	PR	2	20	1.2	(TS_2, TS_{10})	PR	2	15	4.2
(TS_2, TS_3)	RHR	1	10	5.0	(0, TS_{10})	RHR	1	25	0
(TS_2, TS_{11})	RHR	1	6	4.2	(TS_{10}, TS_{12})	PR	2	55	4.6
(TS_2, TS_7)	RHR	1	13	2.3	(TS_{12}, TS_{13})	PR	2	35	2.3
(0, TS_3)	RHR	1	30	0	(TS_{12}, TS_{14})	PR	2	20	4.3
(TS_3, TS_4)	PR	2	80	6.7	(TS_{14}, TS_{15})	PR	2	30	2.4
(TS_3, TS_{11})	RHR	1	12	6.0	(TS_{15}, TS_{16})	PCR	3	40	82
(TS_4, TS_5)	PR	2	90	8.0	(TS_{13}, TS_{17})	PR	2	35	2.5
(TS_5, TS_8)	PCR	3	50	80	(TS_{17}, TS_{18})	PR	2	45	5.3

TABLE 5. Parameter settings of APSO-TSDS algorithm.

Element Desc.	Parameter	Value
Swarm Size	M	60
Velocity Control Parameter	α	1
Velocity Control Parameter	β	0.55
Maximum Flight Velocity	$vmax$	1.2
Number of Archive Set	$nArch$	60
Maximum Number of Iterations	N_max	100

(3) Physical nodes have three types of memory resource capacity: pm_Mem_k =[200.0, 400.0, 800.0];

(4) The network distance between physical nodes $Dist(pm_k, pm_q)$=[1.0, 4.0, 8.0]. Where $Dist_{kq} = 1.0$ represents the same physical node, $Dist_{kq} = 4.0$ indicates the same CDC, and $Dist_{kq} = 8.0$ denotes different CDCs.

The relevant experimental parameters for the APSO-TSDS algorithm are listed in Table 5.

Experimental environment: Processor: Intel (R) Celeron (R) 4-core CPU 1000M @1.80GHz 1.80GH 1.80GHz 1.80GH, Memory: 8G RAM.

To obtain valid data from the experiments, we randomly generated code for testing using the random function of the MathLab tool. Six user requests were implemented for the two CDCs, $Ureq = \{\times 1.0, \times 2.0, \times 3.0, \times 4.0, \times 5.0, \times 6.0\}$. 10 experiments were performed for each user request, each experiment was iterated 100 times, and the experimental data were counted and averaged. Finally, the experimental data were statistically analyzed and compared.

B. BASELINE DEPLOYMENT STRATEGIES

This study discusses the deployment strategy of the execution container for tool services in an IoT cloud environment. To verify the performance of the APSO-TSDS strategy, the GA-NSGA-II [20], ACO-MCMS [21], and MSG-NSGA-III [23] strategies were selected as baseline deployment strategies in the experiments. These baseline deployment strategies were introduced in related works and compared with the proposed APSO-TSDS strategy.

-The GA-NSGA-II strategy uses genetic algorithm for non-dominated sorting (NSGA-II) to realize microservice deployment and resource allocation. The algorithm considers four optimization objectives: the threshold distance between microservices and execution containers, load balancing of computing resources in the CDC, reliability of microservices, and network distance between execution containers of microservices.

-The ACO-MCMS strategy is based on an ant colony optimization algorithm to handle the container scheduling of microservices. The optimization objectives include the data transmission cost between microservices, resource utilization balance between servers, and the average failure rate of microservices.

-The MSG-NSGA-III strategy adopts reference point-based multi-objective NSGA-II algorithm (NSGA-III) to solve the deployment and startup of microservices, which optimizes five objectives: idle rate of computing and storage resources, microservices idle rate, and load balancing rate of computing and storage resources.

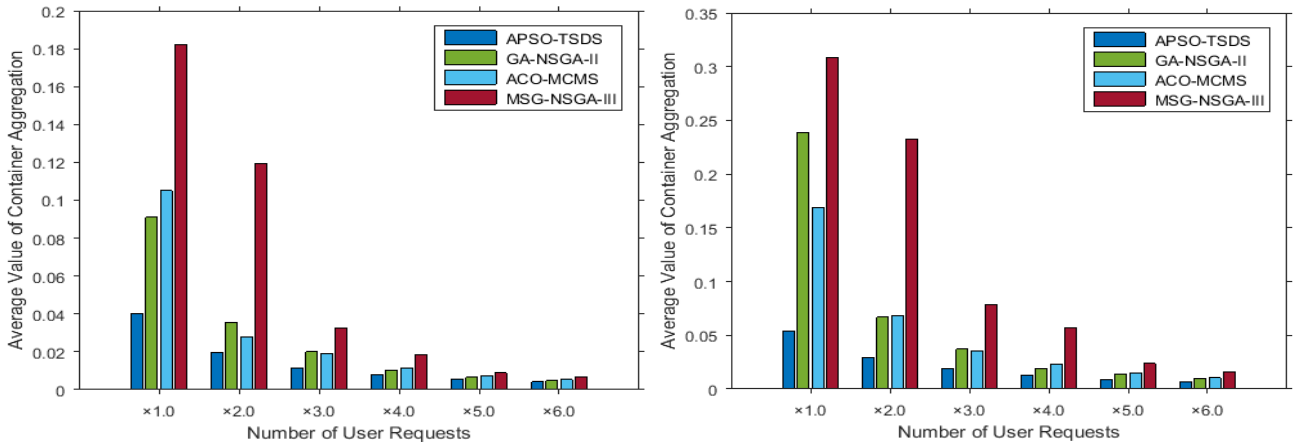
C. ANALYSIS AND COMPARISON

In this experiment, we evaluated the proposed APSO-TSDS strategy using six user requests and two CDCs of different sizes. The parameters evaluated include the container aggregation value, service transmission overhead, and CPU and memory resources utilization in the CDC.

1) CONTAINER AGGREGATION VALUE

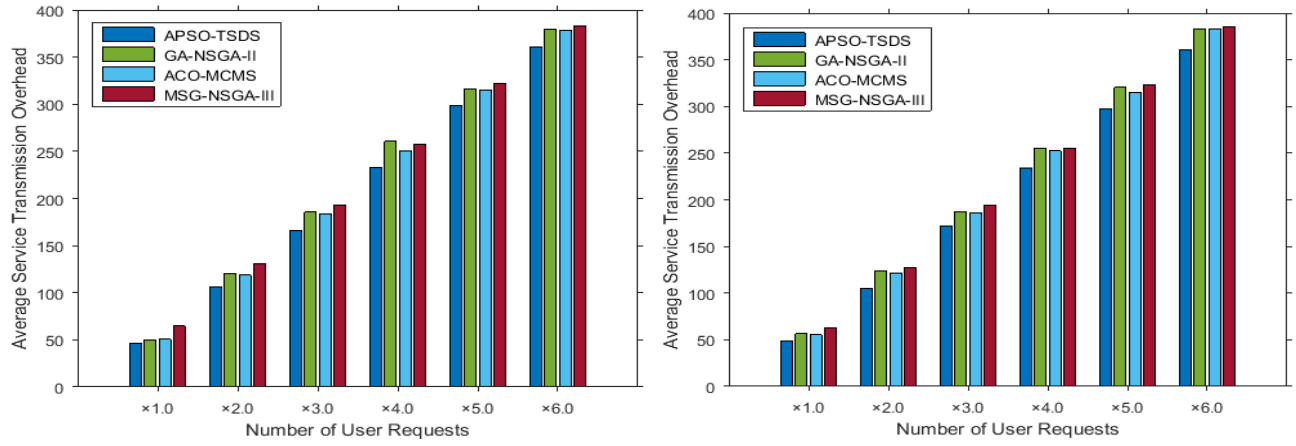
The container aggregation value is the reciprocal of the sum of edge weights between services in the SFC, which describes the degree of aggregation of execution containers between tool services. The smaller the container clustering value, the more clustered is the container. In this study, the execution containers of services with invocation relationships are deployed to the same physical node or CDC as much as possible, with the purpose of reducing the data transmission overhead by reducing the data transmission network distance between services, so as to improve the quality of service (QoS) of users. Furthermore, if resources allow clustering similar containers into a certain part of the physical nodes in a cloud data center, especially when resources are sufficient, it may improve the resource utilization of physical nodes.

Figure 6 shows a comparison of the execution container aggregation values for each strategy. We find that the MSG-NSGA-III strategy performs the worst because it only focuses on resource utilization and balance rate and does not



(a) $sizeof(CDC) = 120$ (b) $sizeof(CDC) = 240$

FIGURE 6. Comparison results of aggregation value for execution container.



(a) $sizeof(CDC) = 120$ (b) $sizeof(CDC) = 240$

FIGURE 7. Comparison results of service transmission overhead.

TABLE 6. Range of container aggregation values of four strategies.

Strategies	aggregation value of execution container	
	120 nodes	240 nodes
APSO-TSDS	$0.00411 \sim 0.0401$	$0.00699 \sim 0.0537$
GA-NSGA-II	$0.00484 \sim 0.0910$	$0.00968 \sim 0.2390$
ACO-MCMS	$0.00541 \sim 0.1051$	$0.01046 \sim 0.1688$
MSG-NSGA-III	$0.00696 \sim 0.1819$	$0.01550 \sim 0.3083$

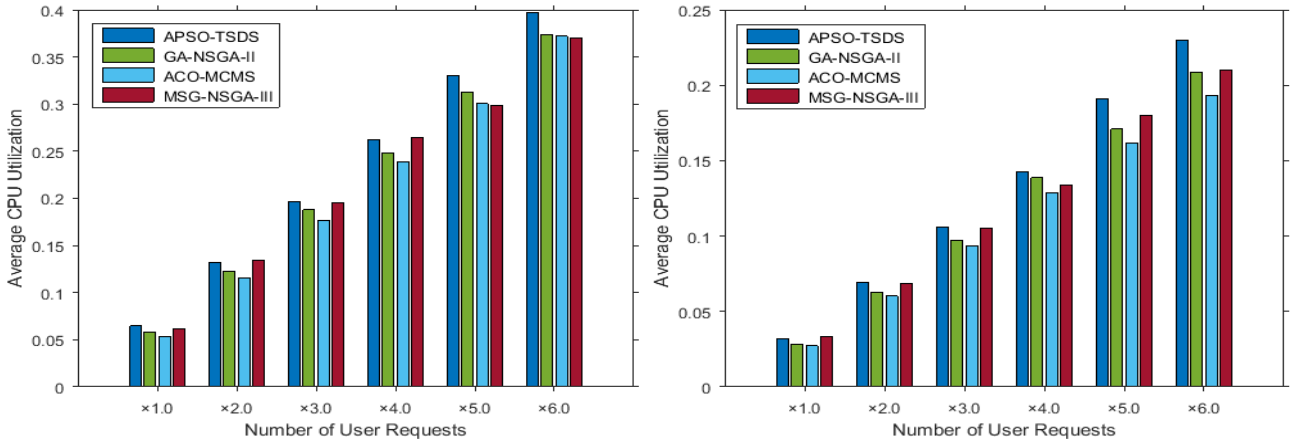
consider the invocation relationship between services. The performance of the GA-NSGA-II and ACO-MCMS strategies is second, because the GA-NSGA-II strategy considers the network distance between services, whereas the ACO-MCMS strategy considers the network distance between services and the data transmission overhead. To a certain extent, the execution container of services with an invocation relationship may be aggregated to the same physical node or CDC. The APSO-TSDS strategy performs the best because the proposed strategy considers the invocation relationship between services.

Table 6 lists the aggregate value ranges of execution container for the four strategies. It can be seen from the table that the APSO-TSDS strategy proposed has good aggregation value ranges.

2) SERVICE TRANSMISSION OVERHEAD

The service transmission overhead includes the data transmission overhead between services and the transmission overhead of the execution container image, which depends on the amount of data transmission and the network distance between physical nodes. The comparison results of the service transmission overheads of the four strategies are shown in Figure 7.

In the process of deployment for service execution containers, the GA-NSGA-II strategy only considers the network distance between services and ignores the amount of data transmission between services, whereas the ACO-MCMS strategy considers the data transmission overhead between services and ignores the transmission overhead of the container image, which may affect the optimization of the entire transmission overhead of the service. However, the



(a) $sizeof(CDC) = 120$ (b) $sizeof(CDC) = 240$

FIGURE 8. Comparison results of CPU resource utilization in CDC.

MSG-NSGA-III strategy does not consider service transmission overhead.

It can be observed from Figure 7 (a) and (b) that in CDCs with different numbers of physical nodes, compared with other deployment strategies, the proposed APSO-TSDS service deployment strategy significantly reduces the service transmission overhead. This is because we simultaneously consider the data transmission overhead between services, transmission overhead of executing container images, and impact of performing container aggregation on service transmission overhead optimization. The MSG-NSGA-III strategy performed the worst in the optimization of service transmission overhead, followed by the GA-NSGA-II strategy and ACO-MCMS, and our proposed APSO-TSDS deployment strategy performed the best. In the CDC with 120 physical nodes, the APSO-TSDS strategy was better than the GA-NSGA-II, ACO-MCMS, and MSG-NSGA-III strategies 9.36, 8.32, and 17.11, respectively; whereas in the CDC with 240 physical nodes, there were 11.26, 9.93, and 14.91, respectively. The experimental results under two different CDC configurations show that the APSO-TSDS strategy outperforms the other deployment strategies for different numbers of user requests.

Table 7 lists the statistical analysis of four strategies in the IoT cloud. This may prove the effectiveness of the APSO-TSDS strategy in the optimization of service transmission overhead. In the table, we give the maximum, average and minimum value of service transmission overhead for each strategy in different CDCs, respectively. The comparison shows that the APSO-TSDS strategy is better than other strategies.

3) CPU AND MEMORY RESOURCE UTILIZATION OF CDC

The resource utilization of physical nodes in the CDC is the ratio of resources used to total resources. Resource utilization is typically improved by deploying multiple parallel execution containers in one physical node. When CDC resources

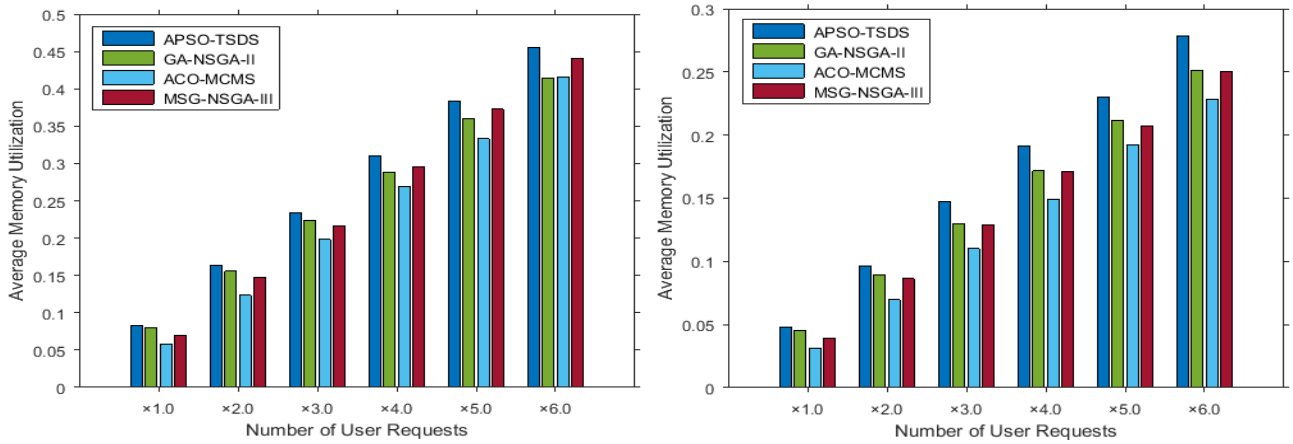
TABLE 7. Statistical analysis of service transmission overhead.

Strategies	Service transmission overhead					
	120 nodes			240 nodes		
	Min	Mean	Max	Min	Mean	Max
APSO-TSDS	195.67	201.90	213.29	193.90	202.72	211.06
GA-NSGA-II	211.21	218.87	226.13	214.37	221.15	227.92
ACO-MCMS	209.42	216.33	224.05	212.95	219.09	225.53
MSG-NSGA-III	219.98	225.05	230.39	219.98	224.72	230.39

are sufficient, energy consumption can be reduced by improving resource utilization and shutting down idle servers to increase the profit of cloud providers. Figure 8 and 9 show the comparison results of CPU and memory resource utilization of the four strategies in CDCs of different sizes.

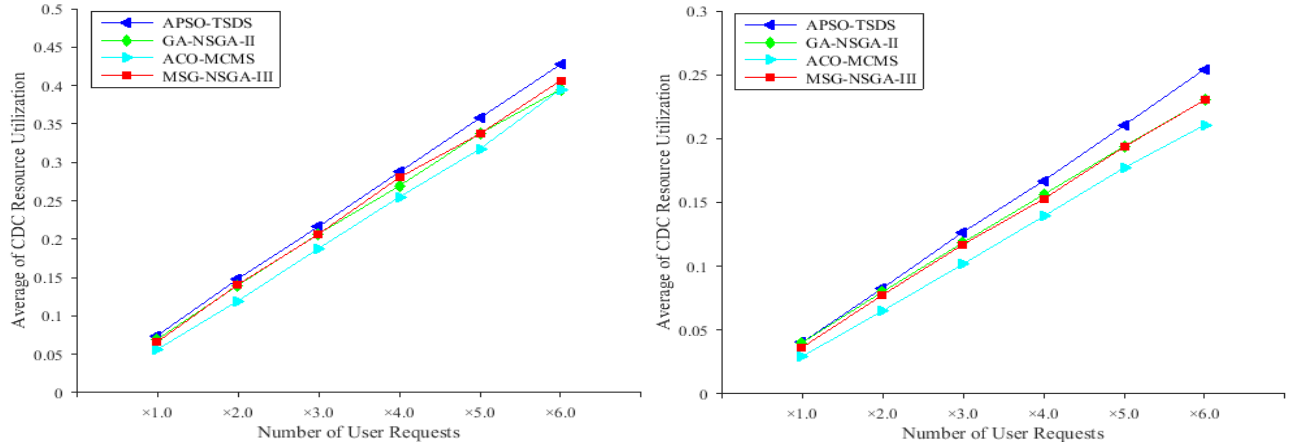
As show in Figures 8 and 9, as the number of user requests increases, the ACO-MCMS strategy maintains low resource utilization because it mainly focuses on balancing the utilization of CPU and memory resources among physical nodes, that is, keeping all physical nodes in a balanced state, which may reduce the resource utilization of the entire CDC and cause resource waste. However, the GA-NSGA-II and MSG-NSGA-III strategies consider improving resource utilization but ignore the factor of aggregation for execution containers, which affects the resource utilization performance of the CDC. APSO-TSDS strategy has a rapid growth rate in the utilization of CPU and memory resources. The main reason is that, under the constraint of load balancing, by aggregating the execution containers of services in SFC, the number of execution containers in the computing server may increase, thus improving the utilization of the server.

To facilitate discussion and understanding, Table 8 lists the CPU and memory resource utilization of the four strategies for CDCs of different sizes. In the case of 120 physical nodes, compared with the GA-NSGA-II, ACO-MCMS, and MSG-NSGA-III strategies, the average CPU utilization of the APSO-TSDS strategy increased by 6.74%, 12.03%, and 3.43%, respectively. Memory utilization increased by 6.26%,



(a) $sizeof(CDC) = 120$ (b) $sizeof(CDC) = 240$

FIGURE 9. Comparison results of memory resource utilization in CDC.



(a) $sizeof(CDC) = 120$ (b) $sizeof(CDC) = 240$

FIGURE 10. Comparison results of resource utilization in CDC.

TABLE 8. CPU and memory utilization of four algorithms.

Strategies	CPU and memory utilization			
	120 nodes		240 nodes	
	CPU	Memory	CPU	Memory
APSO-TSDS	23.06%	27.16%	12.85%	16.54%
GA-NSGA-II	21.17%	25.37%	11.77%	14.99%
ACO-MCMS	20.95%	23.31%	11.08%	13.01%
MSG-NSGA-III	22.07%	25.69%	12.18%	14.72%

22.18%, and 8.3%, respectively. For 240 physical nodes, CPU utilization increased by 9.69%, 15.63%, and 3.15%, respectively. Memory utilization increased by 9.94%, 32.82%, and 14.06%, respectively. The results from the two experiments show that our proposed APSO-TSDS strategy has better utilization values than the other deployment strategies.

As shows in Figure 10 and Table 8, the APSO-TSDS strategy performs the best in terms of overall resource utilization with different user request numbers and CDC.

VII. CONCLUSION

This study develops a resource management platform based on the Band-area Application Container (BAC) as a Service (BaaS), which is used to build and deploy IoT applications and existing enterprise business systems, and integrates these applications into a complete application system through cooperation between BACs. In the microservice framework, because there is no unified template and specification for microservices, to adapt to the interface characteristics of the BAC, we propose a fine-grained application service model called tool service. Three types of invocation relationships between tool services were constructed: pipeline relationships, producer-consumer relationships and random-handshake relationships. In BAC, developers can combine services into service function chains (SFCs) to build new sub-application systems through the invocation relationship between services. Furthermore, the BaaS platform structure and tool service virtualization are described. To solve the deployment problem of the execution container for tool

service in the BaaS platform, three optimization objective functions are built: service transmission overhead, resource load balancing in the CDC, and execution container aggregation. Based on the optimization objective function, an accelerated particle swarm optimization (APSO) for the deployment strategy of the tool service execution container (APSO-TSDS) was established. The APSO-TSDS strategy searches for Pareto solutions based on crowding distance and fitness values. We conducted simulation experiments under different CDC size and numbers of user requests. A comparison with the experimental results of three existing deployment strategies shows that the APSO-TSDS strategy has significant improvements in the execution container aggregation value, service transmission overhead, and resource utilization of the CDC, outperforming existing deployment strategies. The proposed strategy reduces service transmission overhead and improves the resource utilization of the CDC through the aggregation of execution containers for tool services. In the IoT cloud, resource allocation is a key technology. The deployment environment (optimization objective) of application services may change, such as the change of resource type. Developers can configure the objective parameters of the APSO-TSDS strategy according to actual business needs, and provide technical support for multi-objective optimization problems.

In future research, the APSO-TSDS strategy will be applied to an actual IoT cloud environment. In addition, it is important to evaluate the performance of IoT cloud data centers, which is also related to the interests of clients and cloud providers. The follow-up work will use queuing theory to optimize the performance parameters of heterogeneous cloud data centers in the IoT cloud, such as average waiting time, cloud throughput and average response time.

REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. Grid Comput. Environ. Workshop*, Austin, TX, USA, Nov. 2008, pp. 1–10.

[2] T. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges," in *Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Perth, WA, Australia, Apr. 2010, pp. 27–33.

[3] A. Razzaq, "A systematic review on software architectures for IoT systems and future direction to the adoption of microservices architecture," *Social Netw. Comput. Sci.*, vol. 1, no. 6, pp. 1–30, Oct. 2020.

[4] P. P. Ray, "A survey of IoT cloud platforms," *Future Comput. Inform. J.*, vol. 1, nos. 1–2, pp. 35–46, Dec. 2016.

[5] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the Internet of Things," in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Berlin, Germany, Sep. 2016, pp. 1–6.

[6] S. K. Datta and C. Bonnet, "Next-generation, data centric and end-to-end IoT architecture based on microservices," in *Proc. IEEE Int. Conf. Consum. Electron.-Asia (ICCE-Asia)*, JeJu, South Korea, Jun. 2018, pp. 206–212.

[7] P. Krivic, P. Skocir, M. Kusek, and G. Jezic, "Microservices as agents in IoT systems," in *Proc. KES Int. Symp. Agent Multi-Agent Syst., Technol. Appl.*, May 2017, pp. 22–31.

[8] F. Bob, *Microservices, IoT, and Azure*, vol. 1, 11th ed. New York, NY, USA: Spring, 2015, pp. 1–183.

[9] J. Gundaniya and C. H. Lung, "Investigation of containerized-IoT implementation based on microservices," in *Proc. Int. Conf. Simulation Tools Techn.* Guiyang, China: Springer, Aug. 2020, pp. 104–115.

[10] X. Wan, X. Guan, T. Wang, G. Bai, and B.-Y. Choi, "Application deployment using microservice and Docker containers: Framework and optimization," *J. Netw. Comput. Appl.*, vol. 119, pp. 97–109, Oct. 2018.

[11] D. Bhamare, M. Samaka, A. Erbad, R. Jain, and L. Gupta, "Exploring micro-services for enhancing internet QoS," *J. Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 11, pp. 1–13, Jun. 2018.

[12] X. S. Yang, S. Deb, and S. Fong, "Accelerated particle swarm optimization and support vector machine for business optimization and applications," in *Proc. Int. Conf. Netw. Digit. Technol.* Berlin, Germany: Springer, Mar. 2011, pp. 53–66.

[13] M. Ouyang, J. Xi, W. Bai, and K. Li, "Band-area application container and artificial fish swarm algorithm for multi-objective optimization in Internet-of-Things cloud," *IEEE Access*, vol. 10, pp. 16408–16423, 2022.

[14] A. Bhawiyuga, D. P. Kartikasari, K. Amron, O. B. Pratama, and M. W. Habibi, "Architectural design of IoT-cloud computing integration platform," *J. Telkomnika*, vol. 17, no. 3, pp. 1399–1408, Jun., 2019.

[15] A. Samuel, A. Brush, and R. Mahajan, "Lab of things," *J. ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 18, no. 4, pp. 37–40, Jan. 2015.

[16] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "Contemporary Internet of Things platforms," *J. Comput. Sci.*, vol. 89, pp. 5–16, Jan. 2015.

[17] R. Payal and A. P. Singh, "A study of various hardware and cloud based Internet of Things platforms," *R. EasyChair*, no. 4335, Oct. 2020.

[18] J. Na, K.-J. Lin, Z. Huang, and S. Zhou, "An evolutionary game approach on IoT service selection for balancing device energy consumption," in *Proc. IEEE 12th Int. Conf. e-Bus. Eng.*, Beijing, China, Oct. 2015, pp. 331–338.

[19] J. Zhou and X. Yao, "Multi-population parallel self-adaptive differential artificial bee colony algorithm with application in large-scale service composition for cloud manufacturing," *Appl. Soft Comput.*, vol. 56, pp. 379–397, Jul. 2017.

[20] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *J. Grid Comput.*, vol. 16, no. 1, pp. 113–135, Mar. 2018.

[21] M. Lin, J. Xi, W. Bai, and J. Wu, "Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud," *IEEE Access*, vol. 7, pp. 83088–83100, 2019.

[22] H. Bouzary and F. F. Chen, "A hybrid grey wolf optimizer algorithm with evolutionary operators for optimal QoS-aware service composition and optimal selection in cloud manufacturing," *Int. J. Adv. Manuf. Technol.*, vol. 101, nos. 9–12, pp. 2771–2784, Apr. 2019.

[23] W. Ma, R. Wang, Y. Gu, Q. Meng, H. Huang, S. Deng, and Y. Wu, "Multi-objective microservice deployment optimization via a knowledge-driven evolutionary algorithm," *Complex Intell. Syst.*, vol. 7, no. 3, pp. 1153–1171, Jun. 2021.

[24] M. H. Mousa and M. K. Hussein, "Efficient UAV-based mobile edge computing using differential evolution and ant colony optimization," *J. PeerJ Comput. Sci.*, vol. 8, no. e870, pp. 1–24, Feb. 2022.

[25] I.-D. Filip, F. Pop, C. Serbanescu, and C. Choi, "Microservices scheduling model over heterogeneous cloud-edge environments as support for IoT applications," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2672–2681, Aug. 2018.

[26] B. Liu, P. F. Li, W. W. Lin, N. Shu, Y. Li, and V. Chang, "A new container scheduling algorithm based on multi-objective optimization," *Soft Comput.*, vol. 22, no. 23, pp. 7741–7752, Jul. 2018.

[27] H. Tian, X. Xu, T. Lin, Y. Cheng, C. Qian, L. Ren, and M. Bilal, "DIMA: Distributed cooperative microservice caching for Internet of Things in edge computing by deep reinforcement learning," *World Wide Web*, vol. 10, no. 2, pp. 1–24, Aug. 2021.

[28] A. A. C. D. Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, "Microservice remodularisation of monolithic enterprise systems for embedding in industrial IoT networks," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.* Cham, Switzerland: Springer, Jun., 2021, pp. 432–448.

[29] F. Ramezani, M. Naderpour, and J. Lu, "A multi-objective optimization model for virtual machine mapping in cloud data centres," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Vancouver, BC, Canada, Jul. 2016, pp. 1259–1265.

[30] M. Kumar and S. C. Sharma, "PSO-COGENT: Cost and energy efficient scheduling in cloud environment with deadline constraint," *Sustain. Comput., Informat. Syst.*, vol. 19, pp. 147–164, Sep. 2018.

[31] M. Adhikari and S. N. Srirama, "Multi-objective accelerated particle swarm optimization with a container-based scheduling for Internet-of-Things in cloud environment," *J. Netw. Comput. Appl.*, vol. 137, pp. 35–61, Jul. 2019.

[32] M. Adhikari, S. N. Srirama, and T. Amgoth, "Application offloading strategy for hierarchical fog environment through swarm optimization," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4317–4328, May 2019.
[33] X. Huang, C. Li, H. Chen, and D. An, "Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies," *Cluster Comput.*, vol. 23, no. 2, pp. 1137–1147, Jun. 2020.
[34] Z. Wang, Y. Zhang, and X. Shi, "Cloud computing resource scheduling strategy based on competitive particle swarm algorithm," *J. Hunan Univ.*, vol. 48, no. 6, pp. 80–87, Mar. 2021.
[35] M. H. Mousa and M. K. Hussein, "Efficient UAV-based MEC using GPU-based PSO and Voronoi diagrams," *J. Comput. Model. Eng. Sci.*, vol. 2022, no. 2039, pp. 1–21, Mar. 2022.
[36] M. H. Mousa and M. K. Hussein, "Toward high-performance computation of surface approximation using a GPU," *J. Comput. Electr. Eng.*, vol. 99, no. 107761, pp. 1–12, Apr., 2022.
[37] *Wetoband*. [Online]. Available: https://www.wetoband.com/
[38] M. Barr and C. Wells, *Category Theory for Computing Science*, vol. 1, 13th ed. New York, NY, USA: Prentice-Hall, 1990, pp. 16–281.
[39] R. Milner, *Communicating and Mobile Systems: The π-Calculus*, vol. 1, 11th ed. London, U.K.: Cambridge Univ. Press, 1999, pp. 3–151.
[40] W. H. Bai, J. Q. Xi, S. W. Huang, and J. X. Zhu, "Multi-granular application management platform and multi-core-aware parallel scheduling model," *J. Metall. Mining Ind.*, vol. 9, pp. 829–842, Jan. 2015.
[41] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.
[42] J. H. Zheng and J. Zou, *Multi-objective Evolutionary Optimization*. Beijing, China: Science Press, 2017, pp. 1–291.
[43] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, Perth, WA, Australia, Nov. 1995, pp. 1942–1948.
[44] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. Comput. Cybern. Simulation*, Orlando, FL, USA, Oct. 1997, pp. 4104–4108.
[45] L. Zadeh, "Optimality and non-scalar-valued performance criteria," *IEEE Trans. Autom. Control*, vol. AC-8, no. 1, pp. 59–60, Jan. 1963.
[46] Alibaba Corp. *Alibaba Cluster Trace V2018*. Accessed: Sep. 26, 2021. [Online]. Available: https://github.com/alibaba/clusterdata

MINGXUE OUYANG received the B.S. degree from the National University of Defense Technology, in 2004, and the M.S. degree from the School of Software Engineering, South China University of Technology, in 2010, where he is currently pursuing the Ph.D. degree. His research interests include cloud computing, parallel processing, high-performance computing, formal theory of software systems, and formal semantics.

JIANQING XI received the M.S. degree from the National University of Defense. Technology, in 1988, and the Ph.D. degree, in 1992. He is currently a Full Professor with the South China University of Technology and the Head of the Infrastructure Software and Application Construction Technology Laboratory of Guangdong Province. His research interests include cloud computing platform, parallel scheduling, and software architecture, formal theory of software systems, and formal semantics.

WEIHUA BAI received the M.E. degree from the School of Computer Science, South China Normal University, in 2006, and the Ph.D. degree from the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China, in 2017. He is currently an Associate Professor with the School of Computer Science, Zhaoqing University. His research interests include cloud computing, parallel scheduling, and software architecture. He is a member of the China Computer Federation.

KEQIN LI (Fellow, IEEE) is currently a SUNY Distinguished Professor of computer science with the State University of New York. He is also a National Distinguished Professor with Hunan University, China. He has authored or coauthored over 850 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds over 70 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top five most influential scientists in parallel and distributed computing in terms of both single-year impact and career-long impact based on a composite indicator of Scopus citation database. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, and intelligent and soft computing. He is an AAIA Fellow. He is also a member of the Academia Europaea (Academician of the Academy of Europe). He has chaired many international conferences. He is also an Associate Editor of the *ACM Computing Surveys* and the *CCF Transactions on High Performance Computing*. He has served on the Editorial Boards for the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, the IEEE TRANSACTIONS ON SERVICES COMPUTING, and the IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING.