Received December 21, 2021, accepted February 1, 2022, date of publication February 9, 2022, date of current version February 15, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3150326

Band-Area Application Container and Artificial Fish Swarm Algorithm for Multi-Objective Optimization 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; and in part by the Zhaoqing University Science and Technology Projects under Grant zlgc201933.

ABSTRACT Container virtualization methods based on application deployment levels have been widely adopted in cloud-computing environments to implement application construction, deployment, and migration. However, most application containers focus on the interface between the applications and hosts and lack collaboration between application containers. This study proposes a new application container model that contains users, application services, documents, and messages, called Band-area Application Container. A salient feature of the Band-area is that it can express a variety of things in reality, such as organizations or individuals. End users can build a complex and changeable application system through cooperation between the Band-areas. However, the resource allocation of non Internet-of-Thing and Internet-of-Thing tasks from the application container is an open issue. The resource allocation method of tasks should not only improve the quality of the user experience, but also reduce energy consumption by improving the resource utilization of the server. To solve this problem, an artificial fish swarm algorithm is proposed to optimize container-based task scheduling. The algorithm considers not only the reliability, processing time overhead, and energy consumption of the task, but also the resource utilization of the servers. Experimental evaluation shows that, compared with the existing three algorithms, the algorithm obtains a better improvement rate in task processing time overhead, energy consumption, reliability, and cluster load balancing.

INDEX TERMS Application container, artificial fish swarm algorithm, Internet-of-Things, multi-objective optimization, task scheduling.

I. INTRODUCTION

Cloud computing is a service-oriented computing paradigm that provides users with three categories of services: Infrastructure-as-a-Service (IaaS), Software-as-a-Service (SaaS), and Platform-as-a-Service (PaaS) [1]. IaaS is the virtualization hardware layer that consists of servers, storage, and network devices in the cloud data center (CDC). The various computing resources (e.g., CPU, memory, network, etc.) of each server are packaged into VM or container instances of different sizes and provide abstract computing unit services for users and PaaS through APIs, such as Amazon EC2. SaaS provides single or combined services (applications) to the end users. Each application is deployed in a specific PaaS container without managing the infrastructure and platform for running the application, such as Google Mail&Docs. PaaS provides users with a platform for developing, testing, and deploying applications including programming languages, databases, web servers, and operating systems (OSs). It uses IaaS by requesting a VM or container, such as the Google App Engine.

Although PaaS applications and microservices can be used to create enterprise applications in SOA architecture, there are drawbacks when building business systems [2], [3]. The PaaS

The associate editor coordinating the review of this manuscript and approving it for publication was Michael Lyu.

platform only provides coarse-grained modules, such as Human Resources (HR), Office Automation (OA), Customer Relationship Management (CRM) etc. While microservices supports only fine-grained business goals. Neither of these methods integrates diverse types of businesses into a comprehensive enterprise management system. The container virtualization method based on the application deployment level (i.e., the application container) is an efficient way to solve these drawbacks [4]. For example, the Google App Engine (GAE) can deploy diverse types of applications into the execution container, provide a running environment for applications, and apply the API interface between the container and the host system to use the computing resources of the IaaS layer [5].

However, most application containers focus on the interface between the application and the host. In the Internet of Things (IoT) cloud environment, with the rapid increase in the number of people and IoT devices, and the close integration of people and things, end users are also included in the construction of business systems [6]. Based on this, virtual resource management includes not only hardware and software resources but also user resources and interaction messages and between users and between users and things. For this reason, an application container for coordinating and managing these virtual resources is needed to improve the collaboration between users and between users and things. This study proposes an application container that considers software, hardware, user resources, and interactive messages, called Band-area Application Container (BAC). Users can operate applications and microservices through various operations (e.g., deploying, executing and deleting) provided by BAC according to business requirements, and applications and microservices use the computing resources provided by the VM or container in the IaaS layer through the task scheduler between the application container and the host system.

Due to the VM instances consume too many resources and require nearly a minute of deployment time, which reduces the profits of cloud providers and quality of experience (QoE) for users [7]. To compensate for the drawbacks of virtual machine instances, a lightweight OS-based container technology is proposed, such as Linux container and docker. They require less deployment time and computing resources. However, the optimization of container resource management is an NP-hard problem. However, related solutions and implementations exist. For example, Kubernetes proposed a two-phase strategy involving: Predicates and Priorities. The Docker swarm proposes three scheduling strategies: Spread, Random and Binpack. These strategies focus more on the allocation of computing resources, and open issues remain for the optimization of container resource management [8]. Strategies to optimize the processing time overhead, energy consumption, and execution reliability for IoT and non-IoT tasks have not yet been implemented. The container allocation strategy should meet the performance requirements of both tasks and the CDC. To improve the processing time overhead, energy consumption, execution reliability of IoT and non-IoT tasks, and CDC performance, further exploration is required.

At present, there have been many scheduling strategies are proposed which on the resources allocation of container, and various meta-heuristic strategies have been adopted to solve multi-objective optimization problems (MOPs). The artificial fish swarm algorithm (AFSA) has been widely used as a meta-heuristic strategy in MOPs [9]. AFSA is a global optimization algorithm that does not depend on the mathematical properties of the optimization problem itself and has characteristics such as uncertainty and probability [10]. In this study, we propose an artificial fish swarm algorithm to solve the optimization of three objectives: reducing the processing time overhead and energy consumption, and improving the execution reliability of IoT and non-IoT tasks. In addition, we consider the optimization for the load balancing of physical nodes.

The contributions of this work are summarized as follows.

-Firstly, a new resource management model is established, called Band-area Application Container (BAC), which includes users, documents, fine-grained application services (e.g., microservices), messages, and a set of related operation rules. A BAC is a virtual resource unit that can express a variety of things in reality (e.g., organizations or individuals). End users can create BACs and objects in BACs, and build complex and changeable application systems through links between BACs. Furthermore, we describe the framework and components of the BAC system.

-Secondly, we propose a container-based task scheduling problem model that includes the processing time overhead, energy consumption, and reliability models for the tasks. The model of resource utilization for the execution container and physical node was also considered.

-Thirdly, we established a multi-objective optimization problem model for task deployment. The mode optimizes the processing time overhead, energy consumption, and failure rate for task execution and for each task to select an appropriate container. According to the resource capacity of the physical nodes as constraints, select an optimal physical node to load the selected container for further execution.

-Finally, to solve the container-based task scheduling, we propose an artificial fish swarm algorithm. The algorithm adds a mutation operator to further search for a local optimal solution.

The remainder of this paper is organized as follows. Related work is introduced in Section II. Section III introduces the band-area application container model and system framework. The problem model is described in Section IV. In Section V, an artificial fish swarm algorithm is described in detail. Section VI presents the results and comparison for the experiment. Finally, we give the conclusions in Section VII.

II. RELATED WORK

A. CONTAINER-BASED VIRTUALIZATION

In recent years, the Container-as-a-Service (CaaS) method has been widely concerned and applied [11]. According to different application purposes, containers as a lightweight virtualization technology can be divided into two types:

1) OS-BASED CONTAINER TECHNOLOGY

The Linux-based implementations include the Linux Container (LXC), Linux Vserver, FreeVPS, OpenVZ, and Docker. Non-Linux implementations include BSD jails, Solaris Zones, etc. The OS-based container virtualization method allows multiple independent container instances to share the host OS. Through the computing resource isolation mechanism with the container as the granularity unit, multitenancy can share the computing resources of the same host OS. For example, the Linux container [12] does not simulate an underlying hardware layer. Instead, it provides users with a virtual environment with CPU, memory, network space, and block I/O using native Linux functions such as cgoup, namespace, and chroot. LXC can be regarded as dividing the Linux kernel system into smaller units, and each unit has its own computing resources and a network stack for host applications. Docker [13] used a kernel and an application-level API to extend LXC. Docker further implements image layering so that supporting libraries can be shared. This feature makes the Docker containers lightweight, fast, and scalable. In addition, Docker used the layered file system AuFS to facilitate container migration.

2) APPLICATION-BASED CONTAINER TECHNOLOGY

The container technologies include Azure [14], EAC [15], CloudSNAP [16], VAPP [17], Google App Engine [5], which are deployment modes for implementing applications in a cloud computing infrastructure, mainly focusing on rapid and flexible deployment and rapid migration of applications. In other words, the application container deploys applications into the container and provides a running environment. Applications in containers are independent of the host OS and use the interfaces between the container and host to use computing resources. However, application containers are independent individuals, and there is a lack of communication mechanisms between application containers.

The two container-based technologies are mentioned above based on the more refined and flexible use of computing resources in the cloud computing infrastructure. At the application container layer, application services can be deployed quickly and flexibly, which is a type of computing-resource management model. The model is not aimed at business processing and cannot express things in reality, such as the expressions of virtual organizations or individuals. Therefore, it cannot meet the requirements of business collaborative computing between organizations or individuals. Based on this, we propose a Band-area Application Container (BAC) model that can express a variety of things in reality and complete business collaborative computing through the links of BACs.

B. RELATED SCHEDULING STATEGIES

In the IoT cloud environment, resource management and scheduling methods are key issues. This is because it affects the consumption of computing resources and QoE of users. This subject is a research hotspot. The related scheduling strategies are summarized in the following three aspects.

First, various multi-objective optimization problems (MOPs) exist in multiple fields, that require multi-objectives to be optimized at the same time. However, these optimization objectives often conflict with and influence each other. Therefore, researchers have adopted various multi-objective optimization strategies to obtain the optimal solution (i.e., Pareto solution) in a cloud environment. For example, Garg *et al.* [18] aimed to improve resource utilization and profit of VMs in the CDC, based on the prediction model of the artificial neural network (ANN) in which a multi-objective scheduling algorithm is built. The algorithm satisfies the specified QoS requirements for users in the SLA. Guerrero *et al.* [19] used the non-dominated sorting genetic algorithm II (NSGA-II) for the resource management optimization of containers in the cloud architecture; and proposed a container allocation strategy that aims to balance container workload, improve application reliability and reduce network communication overhead. Adhikari *et al.* [20] established a resource allocation and task scheduling strategy using the bat algorithm (BA) and K-means for the VM of the IaaS layer, aiming to minimize the completion time and execution cost of tasks. Kaur *et al.* [21] aiming at the impact of VM migration on the applications performance, a multi-objective optimization model is designed based on Fuzzy Particle Swarm Optimization (FPSO), which aims to minimize transmission time and power consumption, maximize resource utilization of VM. Lin *et al.* [22] presented a container-based scheduling strategy for microservices. A multi-objective optimization model is established based on ant colony optimization (ACO), which aims to reduce the application failure rate and network transmission overhead, and balance the workload of the nodes.

Second, because containers have the advantage of less deployment time and resources to perform tasks, various container-based scheduling strategies have been studied in the IoT cloud. For example, Kaur *et al.* [23] modeled a new architecture for the container selection and scheduling of tasks at the network edge. The architecture adopts cooperative game theory to maximize the task completion time and reduce the NDC energy consumption. Yin *et al.* [24] aimed at the service requirements of limited resources and low latency in fog computing and proposed a container based scheduling strategy. The main goal is to deploy most tasks to the fog node to improve the number of concurrent tasks and reduce delay of task. Adhikari *et al.* [25] formulated a container-based energy-efficient scheduling strategy using accelerated particle swarm optimization (APSO) technology in an IoT cloud environment, aiming to minimize the computational time and energy consumption of the task and improve the resource utilization of the CDC. In addition,

CO_2 emissions and temperature emissions were also considered.

Third, as a heuristic global optimization evolutionary method, the artificial fish swarm algorithm (AFSA) has the characteristics of distributed computing, independent of the optimization problem itself and heuristic domain search, and has been applied to cloud environments. For example, Ying *et al.* [26] presented an AFSA that adds a new behavior strategy and applies the algorithm to virtual machine allocation in the CDC, aiming to balance the load of clusters by reducing the number of virtual machine migrations to achieve energy conservation and environmental protection. The same authors [27] established a new AFSA to solve container-based scheduling problems in clouds, aiming to improve the resource utilization and performance of clusters by adjusting the mapping between the physical nodes and containers. Qin *et al.* [28] proposed an improved AFSA combined with tabu search for task scheduling in Hadoop, which can improve system performance and reduce task running time. Zhang *et al.* [29] designed an improved adaptive AFSA to solve task scheduling with the aim of reducing the total execution time of a task. Luo [30] developed an artificial fish swarm algorithm to solve the cooperation between distributed nodes in a cloud environment. The strategy can not only ensure QoS and reliability but also effectively reduce the number of VM migrations. Albert *et al.* [31] used AFSA to assign tasks to clustering VMs in the cloud, aiming to improve the resource utilization of VMs and the QoE of users.

Deffer from the abovementioned study, this paper considers the processing time overhead, energy consumption and execution reliability of the task as the optimization objectives. To solve the task scheduling problem, an artificial fish swarm algorithm was adopted. The algorithm selects a container suitable for task execution based on the fitness values of processing time overhead, energy consumption, and failure rate of the task and allocates the selected container to the server with the optimal standard deviation of resource utilization of physical nodes.

III. BAND-AREA APPLICATION CONTAINER

We have developed a new resource management model based on application container technology to provide the users with a favorable user-defined virtual operation area [32], which is equivalent to a "virtual space," called Band-area Application Container (BAC). The BAC includes users, documents, application services, messages, and related operation rules.

Unlike the application-based container introduced in Section II, BAC has two salient features:

(1) Abstraction: BAC is a kind of aggregation that can express a variety of things, and users can use BAC to express specific things according to actual application scenarios. For example, BAC can be expressed as institutions, departments, and individuals, as well as engineering projects.

(2) For end users, on the one hand, the end user can control the life cycle of BAC and its contained object set after verifying the authorization. On the other hand, users can build

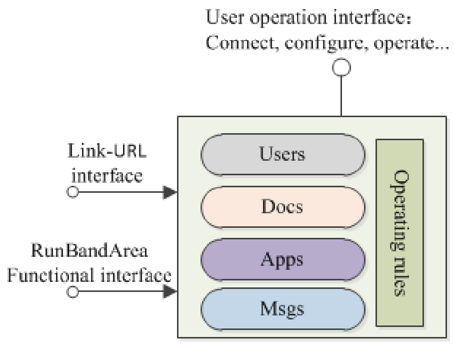

User operation interface:
Connect, configure, operate...
Link-URL interface
RunBandArea Functional interface
Users
Docs
Apps
Msgs
Operating rules

FIGURE 1. The BAC model.

different business application systems through links between BACs according to their own business requirements.

A. BAC MODEL

The BAC model is shown in Figure 1.

- Users refer to the operators of computer software, including IoT users, non-IoT users, developers, and administrators.

- Documents refer to various data files that can be processed using computer software.

- Application services are fine-grained application services (e.g. microservices) or IoT devices application, which can be deployed on any server in the entire server cluster like "tools." We call this "tool services."

- Messages refer to the data transmitted between two objects for communication purpose. Messages are conducive to communication and collaboration between users and between users and application services.

- Operating rules are user activity rules in the BAC that restrict the implementation of the business. Examples include authorization and sharing.

BAC is a functional module that provides the Link-URL interface and RunBandArea functional interface, where the interfaces can be called by applications and developers. In addition, the provided user operation interface is used to connect, configure, and operate the object elements in the BAC. There are two main advantages of this design: first, the user builds the business sub-system by forming a small-grained application flow through the combination of application services in the BAC; and second, BAC can represent organizations (e.g., institutions, departments, etc.) or individuals, that is, virtual organizations and virtual users. Through the links between BACs to achieve business collaboration between organizations and form a larger-grained workflow to build the entire business system.

Owing to the formal method can describe a system model strictly and accurately. Here, we use the basic knowledge of category theory [33] to formally define BAC.

Definition 1: Band-area Application Container (BAC) -A BAC can be represented by a nine-tuple, which is denoted as $BAC = < BID, BName, Users, TSs, Docs, Msgs, P, Bin, Bout >$, where:

(1) BID is the unique identifier of the BAC;

(2) $BName$ represents the name of the BAC;

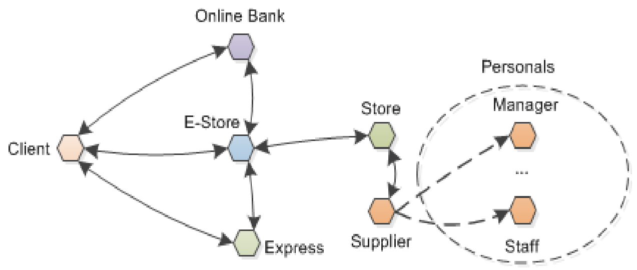


FIGURE 2. The structure of BAC-based supply chain.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "q'"] \arrow[rd, "r'"] & \\
C \arrow[r, "q''"] \arrow[rrd, "r''"'] & R \arrow[r, "r"] & \\
& & R'
\end{tikzcd}$$

FIGURE 3. The pushout.

(3) $Users = \{User_i|i > 0\}$ indicates the user set;

(4) $TSs = \{TS_j|j \geq 0\}$ donotes the application services (i.e.tool services) deployed in the BAC;

(5) $Docs = \{doc_k|k \geq 0\}$ is the document data set;

(6) $Msgs = \{msg_l|l \geq 0\}$ indicates that the message set;

(7) P is the operation rule, that is, operation permissions. The authority of the user is ultimately manifested in the operation control of the object. The end user operates each object (e.g., tool service, document) according to their own role permissions. The mapping relationship between roles and objects can be expressed as $P = \{p|p \in (2^R \rightarrow O) \cup (R \rightarrow 2^O)\}$, where R represents the set of roles, $O =< TSs, Docs, Msgs >$ is a set of objects.

(8) *Bin* denotes the input interface set of the BAC;

(9) *Bout* is output interface set of the BAC.

According to Definition 1, for example, suppose the supply chain management system involves 5 organizations or multiple individuals; these are represented by BACs, including client BAC, online store BAC, online bank BAC, store BAC, express BAC, and supplier BAC. The dotted line connects individual BACs, such as supplier managers and staff. The links between BACs are formed according to the business chain relationship, and Figure 2 shows its structure.

Theorem 1: BAC Category (*BC*)- BAC category *BC* is composed of BAC as the object and the arrow between BACs as the morphism.

Proof: Let $ObjBC = \{B_i|i > 0, i \in Z\}$ be a set of all objects of category BC, and the B_i is a BAC of the BAC-chain. $MorBC = \{B_i \rightarrow B_j|i,j > 0; i,j \in Z\}$ be a set of all morphisms in category BC, and $B_i \rightarrow B_j$ is the arrow between the BACs. Let $dom : MorBC \rightarrow ObjBC$ be a domain function, and $cod : MorBC \rightarrow ObjBC$ be a co-domain function, $\circ : MorBC \times ObjBC \rightarrow MorBC$ be a composition function. The following proves that the system $BC = (ObjBC, MorBC, dom, cod, \circ)$ is a category.

(1) Let $\forall A, B, C \in ObjBC$, $\exists f : A \rightarrow B$, $g : B \rightarrow C \in MorBC$, then $g \circ f : A \rightarrow C \in MorBC$, which satisfies matching properties, i.e., $dom(g \circ f) = A = dom(f)$, $cod(g \circ f) = C = cod(g)$.

(2) Let $h : C \rightarrow D \in MorBC$, then $h \circ g : B \rightarrow D \in MorBC$, and $(h \circ g) \circ f : A \rightarrow D \in MorBC$, $h \circ (g \circ f) : B \rightarrow D \in MorBC$, which satisfies composition properties, i.e. $(h \circ g) \circ f = h \circ (g \circ f)$.

(3) For $\forall A \in ObjBC$, $\exists! id_A \in MorBC$, such that $dom(id_A) = cod(id_A) = A$, the description $\exists!$ indicates only existence, id_A is the identity of object A. For $\forall f \in MorBC$, if $dom(f) = A$, then $f \circ id_A = f$. If $cod(f) = A$, then $id_A \circ f = f$, which satisfies the identity properties.

According to the definition of category [33], the system BC is a category, i.e., the category BC is composed of BAC as the object and the arrow between BACs as the morphism. □

BC morphism reflects a directed arc on the *BC* category chart, depicting the interaction and combination between BACs (i.e., organizations). BAC as the object, that is a node in the category graph. In this way, BACs and BAC morphisms are intertwined to form a large BAC (organizational structure) network.

Definition 2: Pushout -In the BC category, the pushout of a pair of morphisms $f : A \rightarrow B$ and $g : A \rightarrow C$ with the same source is an object R and a pair of morphisms $q' : B \rightarrow R$ and q'': $C \rightarrow R$, so that the square (commutative chart) shown in Figure 3 is exchangeable: $q' \circ f = q'' \circ g$, and satisfies the following conditions: for any objects R' and any of morphisms $r' : B \rightarrow R'$ and r'': $B \rightarrow R$ for which $r' \circ f = r'' \circ g$ there is a unique morphism $r : R \rightarrow R'$, such that $r \circ q' = r'$ and $r \circ q'' = r''$. The pushout process is illustrated in Figure 3.

For example, new cooperation can be generated through the pushout of *BC*. Each time customer *A* purchases goods in online store *B*, *A* has to transport the goods by express *C*. *R* as the first successful collaboration, if there is a problem with the goods, *A* can be returned to store *B* by express *C*, and the return business collaboration *R'* can also be carried out.

B. BAC SYSTEM FRAMWORK

In the BAC-based system framework, according to the type of the function node, including the BAC server, application scheduling actuator, computing node, and data storage. Figure 4 shows the logical structure of the BAC system framework.

The interface service as the entrance for administrators and end users to enter the BAC system and provides user account login and BAC operation interface (i.e., user-visible interfaces). After the user passes authentication, the interface provides an application system map and operation services. The BAC system supports multitenancy. End users use BAC under virtual isolation from each other, and use operation rules to manage object instances such as users, documents, message boards (i.e., carriers of messages), application services (tools), as if everyone has a separate "virtual space,"

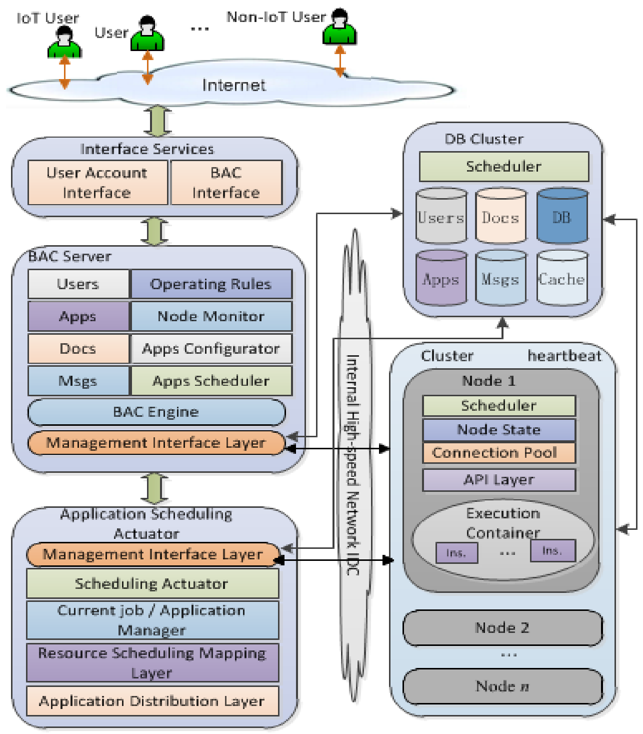


FIGURE 4. Logical structure of the BAC system framework.

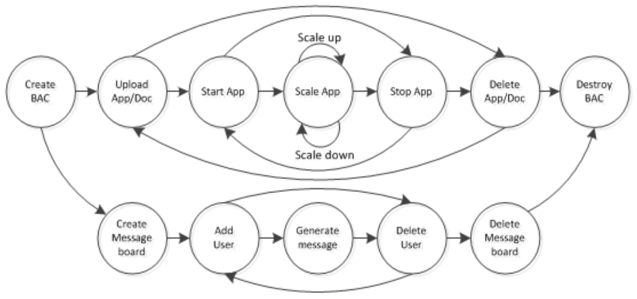


FIGURE 5. The life cycle of BAC.

but the data in the space is secure and is not affected by other user activities.

1) BAC SERVER (BAS)

The BAC is the basic computing resource unit of a BAC system. The BAC engine (BAE) is a key component of the BAC server, and its main functionalities of the BAC engine are as follows:

- Control lifecycle: The BAC engine supports the end user to operate the BAC and the objects of the BAC and controls their life cycle. After the user is authenticated and authorized, the user can create a BAC. Then the user can deploy application services, upload documents, create a message board, and establish a user relationship group in BAC. When the BAS receives the request to create a BAC, it notifies the BAC engine to obtain the server resources required by the BAC, and then instantiates a new BAC to control its life cycle. As shown in Figure 5.

-Support end users in manipulating objects in the BAC: The users can use various operating rules (e.g., sharing) provided by BAC to manipulate objects in BAC. For example, user A shares application services or documents with user B through a sharing operation.

-Support end users to establish relationships between BACs: Users can map business contacts of the business system to BAC relationships and integrate different types of businesses into a comprehensive enterprise management system through BAC relationships. For example, the hierarchical relationship between enterprise and departments, the supply-demand relationship in the supply chain, and the timing relationship between project tasks.

-Monitoring computing nodes: After the application service instances are deployed to the BAC, BAE monitors computing capacity and collects the status information of the computing node based on the heartbeat mechanism, such as the resource usage of the CPU, memory, and network traffic. According to the configuration information of the calling relationship between the application services, it provides decision support for the scheduling of the application task set.

2) APPLICATION SCHEDULING ACTUATOR

The executor is mainly responsible for receiving application call requests, performing task scheduling, and forwarding application execution results. When the end user executes the application service request in BAC, the scheduling executor receives the application scheduling request and then deploys the application service instance to the selected computing node for execution according to the current job/application scheduling request information and load balancing strategy, to distribute large-scale concurrent tasks to the computing node. It monitors and maintains the execution of application services in the computing node and is responsible for caching or forwarding the application execution results, or allowing the computing node to directly return the execution results to the user. Note that we provide a container-based task scheduling algorithm in the next section.

3) COMPUTING NODES(CLUSTER)

The scheduler of the computing node receives the application or task request and deploys the application or task to the execution container (e.g., the Docker container) for execution. Simultaneously, the scheduler collects the resource usage status of the computing node and feeds it back to the BAC server using the heartbeat mechanism. In addition, it maintains the database connection pool, current status information of the task set, and cache of application execution results based on the requirements for the application-scheduling actuator.

4) DATA STORAGE

The main responsibility of data storage is to provide database storage services for application services, documents, and messages in BACs, as well as to cache the execution results of application services and store execution container images.

TABLE 1. Summary of parameters and their description.

Element	Parameter	Description
Band-area Application Container (BAC)	BA	a set of BAC
	$ba_l \in BA$	BAC with identifier l
	$User$	user set
	$User_u \in User$	user with identifier u
	TS	Tool Service set
	$TS_k \in TS$	Tool Service with identifier k
Physical node	PM	a set of physical nodes in a CDC
	$pm_i \in PM$	physical node with identifier i
	$Load_i$	the workload of the physical node pm_i
	AR_i	available resources utilization of physical node pm_i
Execution container	CON	a set of executing container
	$con_l \in CON$	executing container with identier l
	$Load_l$	the workload of the container con_l
	AR_l	available resources utilization of executing container con_l
Task	T	a set of task from Tool Services
	T_k	task with identifier k
	ST_{up}	serialization time of uploading data of task T_k from user u to CDC j
	ST_{dn}	serialization time of downloading data of task T_k from CDC j
	PT	data propagation time on the network
	$Tran_{kj}$	total transmission time of the data of task T_k between user u and CDC j
	$Itrans_{lj}$	transmission time for the execution container con_l image
	$Exec_{kl}$	execution time of task T_k in container con_l
	FT_{kl}	processing time of task T_k
	PW_t	transmission power
	TPC_k	total transmission energy of task T_k
	CPC_l	energy consumped by container con_l
	PC_{kj}	total energy consumed by task T_k in container con_l of CDC j
	TF_{kj}	task T_k failure rate during execution

IV. PROBLEM DESCRIPTION

This section presents the problem statement, including the system and optimization objective models. Table 1 lists the main parameters and their descriptions.

A. SYSTEM MODEL

End users (IoT and non-IoT users) start tool services through Band-area Application Containers (BAC). The tool services include memory-intensive or CPU-intensive tasks and request-based or event-driven tasks. Task request includes two cases: First, for tasks from devices (e.g., PCs, laptops, mobile devices, etc.) of the non-IoT, they are uploaded to the CDC for processing. Second, for tasks from devices (e.g., Smart Phone, Smart Things, Arduino, etc.) of the IoT, because most IoT devices have certain computing capabilities (such as microcontrollers, microprocessors, SOCs, FPGAs, etc.) and storage, they can usually execute fine-grained applications or small computing tasks, which are called "local computing;" otherwise, long tasks or coarse-grained applications in IoT devices need to be uploaded to the CDC for further processing, called "remote computing" [34]. Remote computing can save the energy of IoT devices, so as to improve the service life of device batteries. To simplify the research on this problem, only remote computing is considered for IoT devices.

The admission controller of the BAE determines whether to receive a task by judging the execution authority of the user and the resource availability of the computing node and assigns the control right of the received task to the application scheduling actuator for further processing.

Let B Band-area Application Containers exist, which are formalized as $BA = \{BA_1, BA_2, \ldots, _B\}$. BA contains U users and N tool services, represented as $User = \{User_1, User_2, \ldots, User_U\}$ and $TS = \{TS_1, TS_2, \ldots, TS_N\}$ respectively. There are M IoT devices and V non-IoT devices. Each tool service contains multiple tasks, so all tool services can be regarded as a set of tasks $T = \{T_1, T_2, \ldots, T_K\}$, and the instruction size of the task is measured by million instructions (MI).

For the CDC, we consider P heterogeneous physical nodes (i.e., computing servers) connected by the same Intranet High-speed Network, which is denoted as $PM = \{pm_1, pm_2, \ldots, pm_P\}$. Each physical node can be formalized as $pm_i =< CPU_i^{pm}, Mem_i^{pm}, Store_i^{pm}, BW_i^{pm}, fail_i^{pm} >, i \in \{P\}$, namely the CPU, memory, disk, bandwidth, and failure rate. Capacity of the memory and CPU are in MB and Million Instructions Per Second (MIPS) respectively. According to the resource availability, physical nodes can host L heterogeneous container instances, which are formalized as $CON = \{con_1, con_2, \ldots, con_L\}$, and each container instance can be represented as $con_l =< CPU_l^c, Mem_l^c, Store_l^c, BW_l^c, fail_l^c >, l \in \{L\}$.

B. OPTIMIZATION OBJECTIVE MODEL

The objective models include the processing time overhead, energy consumption model, and failure rate model of the task. Resource utilization was also considered.

1) PROCESSING TIME OVERHEAD MODEL

In the case of no noise interference in the channel, the data transmission overhead between the device of the end user and CDC depends on the network distance, amount of data transmission, and bandwidth. The transmission time includes the data serialization and propagation time.

First, the required serialization time (ST_{up}) for uploading task T_k data of user u to CDC j is defined as follows:

$$ST_{up} = BW_{uj} \times \frac{TS_k}{TR_{uj}} \tag{1}$$

where $TS_k = sizeof(T_k)$ represents the data size for task T_k, and the BW_{uj} denotes the bandwidth between the user device and CDC j, and TR_{uj} indicates the transmission rate.

Second, the required serialization time (ST_{dn}) for downloading the task calculation result from CDC j to user u is formulated as

$$ST_{dn} = BW_{ju} \times \frac{TE_k}{TR_{ju}} \tag{2}$$

here, TE_k represents the calculation result and communication data of task T_k.

Third, the propagation time (PT) of the data on the network is defined as the ratio between the network distance ($Dist$) and network bandwidth (*BW*), including the propagation time of the uplink and the propagation time of the downlink:

$$PT = PT_{up} + PT_{dn} = \frac{Dist_{uj}}{BW_{uj}} + \frac{Dist_{uj}}{BW_{ju}} \tag{3}$$

where, in two-dimensional space (x, y), the network distance is $Dist_{uj} = \sqrt{(X_u - X_j)^2 + (Y_u - Y_j)^2}$.

Therefore, by combining Equations (1), (2), and (3), the total transmission time of task T_k between user u and CDC j can be formulated as Equation (4):

$$\begin{aligned} Tran_{kj} &= ST_{dn} + PT_{up} + PT_{dn} \\ &= \left(BW_{uj} \times \frac{TS_k}{TR_{uj}} + BW_{ju} \times \frac{TE_k}{TR_{ju}}\right) \\ &\quad + \left(\frac{Dist_{uj}}{BW_{uj}} + \frac{Dist_{uj}}{BW_{ju}}\right) \end{aligned} \tag{4}$$

If the physical node pm_j in the CDC does not have the required image for the execution container con_l, then the image is pulled from the data storage to the local repository of the physical node. The image transmission time of the execution container is defined as

$$Itrans_{lj} = BW_{lj} \times \frac{Image_l}{TR_{lj}} \tag{5}$$

where $Image_l$ denotes the size of the image file for the execution container con_l.

The processing time overhead of a task depends on the date transmission time of the task, image transmission time of the container, and execution time of the task. This is expresszed by Equation (6):

$$FT_{kl} = Tran_{kj} + Itrans_{lj} + Exec_{kl} \tag{6}$$

where $Exec_{kl}$ is the execution time of task T_k in container con_l, i.e., $Exec_{kl} = \frac{T_{kMI}}{CPU_{lc}}$. Here, T_{kMI} represents the size of the task instructions measured by the MIs.

2) RESOURCE UTILIZATION

Resource utilization is a key decision parameter that determines task execution time and balance load. If the resource utilization of an execution container is overloaded, the container cannot respond to the request in time, and a low load can waste resources. Here, we only consider CPU and memory resources.

The workload of execution container con_l at each time interval(t' to t'') is formulated as follows:

$$\begin{aligned} Load_l^{cpu} &= \sum_{t=t'}^{t''} CPU_l^c(t) \\ Load_l^{mem} &= \sum_{t=t'}^{t''} Mem_l^c(t) \end{aligned} \tag{7}$$

Here, the available resource utilization of the execution container con_l is the ratio of resources used to total resources. This is formalized in Equation (8):

$$\begin{aligned} AR_l^{cpu} &= \frac{Load_l^{cpu}}{total\_Load_l^{cpu}} \\ AR_l^{mem} &= \frac{Load_l^{mem}}{total\_Load_l^{mem}} \end{aligned} \tag{8}$$

where $total\_Load_l$ denotes the total resource capacity of container. The container with the lowest *AR* value was selected when selecting the execution container for the task.

The remaining capacity of the container was formalized as $Re\_Load_l = (Re\_Load_l - Load_l)$. If Re_Load_l meets the resource requirements of task T_k, then the resource is allocated to the task; otherwise, the task is in the queue waiting for further execution.

Similar to the definitions of Equations (7) and (8), *Load*, *AR* of physical nodes can be defined in Equations (9) and (10):

$$\begin{aligned} Load_j^{cpu} &= \sum_{t=t'}^{t''} CPU_j^c(t) \\ Load_j^{mem} &= \sum_{t=t'}^{t''} Mem_j^c(t) \end{aligned} \tag{9}$$

$$\begin{aligned} AR_j^{cpu} &= \frac{Load_j^{cpu}}{total\_Load_j^{cpu}} \\ AR_j^{mem} &= \frac{Load_j^{mem}}{total\_Load_j^{mem}} \end{aligned} \tag{10}$$

Note that we propose a strategy for selecting the physical nodes for the execution container in the next section.

3) ENERGY CONSUMPTION MODEL

The energy consumed by IoT and non-IoT tasks depends on the following two factors.

First, that Network Transmission Energy Consumption: which includes the energy consumed for data transmission of the task and image transmission of the execution container. The energy consumption of network data transmission is determined by the transmission time and transmission power. Here, let PW_t indicates that fixed transmission power is consumed. The total transmission energy consumption of task T_k is defined as follows:

$$TPC_k = (Tran_{kj} + Itrans_{lj}) \times PW_t \tag{11}$$

Second, that Task Execution Energy Consumption: that is, the energy consumed by computing resources during task execution, which depends on the workload of the execution container. The energy consumed by the execution container is related to the utilization of computing resources (e.g., network, memory, CPU) and the execution time of the task. However, the contribution of the CPU to energy consumption is dominant; in other words, energy consumption and CPU utilization are linearly related, and CPU utilization is related to the workload [34]. To be compatible with the VM energy-consumption model proposed in [37], we use the energy-consumption model of the execution container as follows:

$$CPC_l = PC_l^{idle} + (PC_l^{busy} - PC_l^{idle}) \times AR_l^{cpu} \tag{12}$$

where PC_l^{busy} and PC_l^{idle} represent the energy consumption of the execution container con_l when CPU utilization is 100% and 0%, respectively. Therefore, the energy consumed by task T_k is defined as follows:

$$TCPC_k = CPC_l + Exec_{kl} \tag{13}$$

According to Equations (11) and (13), the total energy consumption of task T_k is formulated as follows:

$$\begin{aligned} PC_{kl} &= TPC_k + TCPC_k \\ &= (Tran_{kj} + Itrans_{lj}) \times PW_t \\ &\quad + CPC_l \times Exec_{kl} \end{aligned} \tag{14}$$

4) FAILURE RATE MODEL

Failure of the running environment (such as hardware or software) where the application is located is inevitable, and the failure rate of the application is reflected in the execution of the tool services. When a tool service task fails, the tool service fails. When all container copies fail, tool service or task fails. The failure rate of a task is related to the failure rate $fail_l$ of a container and the failure rate $fail_j$ of a physical node. This is defined as follows:

$$TF_{kj} = (fail_l + fail_j) \times x_{kl} \tag{15}$$

where x_{kl} is an decision variable, which is formulated as follows:

$$x = \begin{cases} 1, & if\, T_k \in alloc(con_l); \\ 0, & otherwise; \end{cases}$$
$$k \in \{U, V\}, con_l \in CON, \tag{16}$$

where $alloc(con_l)$ represents the set of task allocated in a container con_l. $x_{kl} = 1$ indicates that task T_k is allocated to execution container con_l.

C. MULTI-OBJECTIVE OPTIMIZATION MODEL

Based on the previous model, this study optimizes three objectives: processing time overhead, energy consumption, and failure rate for task. To achieve the best results, we established a model for the multi-objective optimization of task scheduling as follows:

$$minimize\ FT_{kl}(X) \tag{17}$$
$$minimize PC_{kl}(X) \tag{18}$$
$$minimize\ TF_{kj}(X) \tag{19}$$
$$s.t \quad Load_l \le \beta \tag{20}$$
$$\sum_{t=t'}^{t''} Load_l \in total\_Load_j, l \in \{L\}, j \in \{P\} \tag{21}$$
$$AU_j \le 100 \tag{22}$$

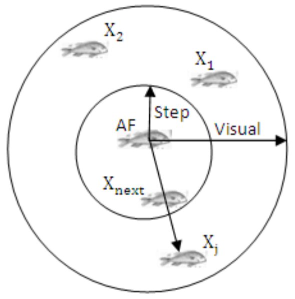


FIGURE 6. The AF model.

$$x = \begin{cases} 1, & if\, T_k \in alloc(con_l); \\ 0, & otherwise; \end{cases}$$
$$k \in \{U, V\}, con_l \in CON. \tag{23}$$

$$\sum_{j=1}^{P} x_{kj} = 1,\ j \in \{P\}, k \in \{U, V\} \tag{24}$$

Equations (17) - (19) denote the optimization of the three objevctives: minimizing processing time overhead, minimizing energy consumption, and minimizing the execution failure rate of the task, respectively.

Equations (20) - (24) indicate that the load of the execution container con_l must satisfy the threshold limit and be less than the maximum capacity of the physical node, respectively. Equation (22) limits the utilization of physical node j. Equations (23) - (24) are decision variable constraints, x_{kl} indicates whether task T_k is assigned to execution container con_l. And $\sum_{j=1}^{P} x_{kj} = 1$ means that each task can only be assigned to one physical node, and each physical node can be assigned multiple containers to execute multiple tasks simultaneously.

Owing to the effectiveness of the meta-heuristic method in multi-objective optimization problems, it is often used to solve such problems. The artificial fish swarm algorithm has fast convergence speed and accuracy in the multi-objective optimization process. Therefore, we propose an artificial fish swarm algorithm to obtain an optimal or suboptimal solution for the previous optimization objective model.

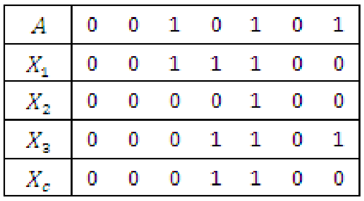
V. PROPOSED ARTIFICIAL FISH SWARM OPTIMIZATION ALGORITHM

A. MODEL OF ARTIFICIAL FISH

The artificial fish swarm algorithm (AFSA) is a swarm intelligence global optimization strategy that simulates the foraging behavior of fish [9,10]. The AF model is illustrated in Figure 6.

B. REPRESENTATION OF ARTIFICIAL FISH AND NEIGHBORS

In AFSA, a suitable candidate solution representation method is the key to solving this problem. Candidate solution X can be represented based on the representation of strings [27]. Each solution was called an artificial fish (AF). Based on the objective optimization mathematical model proposed in



A	0	0	1	0	1	0	1
X_1	0	0	1	1	1	0	0
X_2	0	0	0	0	1	0	0
X_3	0	0	0	1	1	0	1
X_c	0	0	0	1	1	0	0

FIGURE 7. Generation of neighbor center position X_c.

Section 4, we define a 0-1 string to represent each execution container. For example, if n containers are represented by 0-1 code, the container code length is $m = \lceil \log_2 n \rceil$. The integer n is converted into a binary vector $(x_1, x_2, \ldots, x_m)$, that is, the candidate solution X is represented by the vector $[X] = (x_1, x_2, \ldots, x_m)$.

The visual range (*visual*) of a fish is the farthest "distance" that the fish can see, which is a predetermined value. For artificial fish, $A = (a_1, a_2, \ldots, a_m)$ and $B = (b_1, b_2, \ldots, b_m)$, the distance between A and B is defined as follows:

$$dist(A, B) = \sum_{i=1}^{m} |a_i - b_i| \tag{25}$$

where the neighbor set within the visual distance of fish A is denoted as $Near = \{B|dist(A, B) \leq vsiual\}$, $n_f = |Near|$ denotes the number of elements for the neighbor set, that is, the number of neighboring individuals for fish A.

Each status bit x_i^c $(1 \leq i \leq m)$ of neighbor center x_c in the neighbor set is 0 or 1, which appears most frequently in the corresponding status bit of the neighboring individuals. Let $Near_A = \{X_1, X_2, X_3\}$ be the neighbor set of fish A, the generation of the neighbor center x_c is shown in Figure 7.

C. THE EVALUATION FUNCTION OF SOLUTION

When the artificial fish AF_i traverses all execution containers, the position state x_i^c $(1 \leq i \leq m)$ x_i is a candidate solution for the three optimization objectives. To obtain the optimal or suboptimal solution in the optimization process, we use the standardized values of the three objectives to evaluate the candidate solutions. Here, we adopted the weighted sum method [35] to linearly aggregate the three objective functions into one objective function as follows:

$$\begin{aligned} fit(X) = \lambda_1 \times \frac{FT_{kl}(X) - FT_{min}}{FT_{max} - FT_{min}} \\ \lambda_2 \times \frac{PC_{kl}(X) - PC_{min}}{PC_{max} - PC_{min}} \\ \lambda_3 \times \frac{TF_{kj}(X) - TF_{min}}{TF_{max} - TF_{min}} \end{aligned} \tag{26}$$

where the weights $\lambda_1, \lambda_2, \lambda_3 \in [0, 1]$ can be set according to the actual environment. Here let $\lambda_1 = \lambda_2 = \lambda_3 = \frac{1}{3}$. That the FT_{min}, PC_{min}, and TF_{min} are the minimum values for the processing time overhead, energy consumption, and failure rate, respectively, and the FT_{max}, PC_{max}, and TF_{max} denote the maximum values for the corresponding objectives. X is the candidate solution. The value of the fitness function $fit(X)$ reflects the deviation between candidate solution X and the optimal solution.

D. EXECUTION CONTAINER SELECTION POLICY

According to the evaluation value of Equation (26), an execution container suitable for task execution is selected. In the process of fish foraging, AF individuals follow four behaviors: Swarm, Follow, Prey, and Randomly move. To achieve a local search, we added a mutation operator. The AFSA algorithm is summarized as follows:

(1) Initialization: population size *Popsize*, particle set of the artificial fish $\{x_i \mid i \leq Popsize\}$, crowding factor δ, visual range of the artificial fish *visual*, maximum step length of artificial fish movement *step*, number of tries *try_number*.

(2) Swarm: x_c is the neighbor center of the artificial fish x_i, if the fitness of x_c is better $(fit(x_c) < fit(x_i))$ and there are fewer neighbors $(n_f/Popsize < \delta)$, the fish x_i can move to x_c; otherwise, the fish execute following.

(3) Follow: x_{min} is the fish with the best fitness value near x_i, if the fitness of x_{min} is better $(fit(x_{min} < fit(x_i))$ and there are fewer neighbors $(n_f/Popsize < \delta)$, the fish x_i can move to x_c; otherwise, the fish execute preying.

(4) Prey: x_j is a randomly selected fish, according to the *visual* value of artificial fish x_i. If the fitness of x_j is worse $(fit(x_j) > fit(x_i))$, then randomly select another x_j and try again; otherwise, fish x_i can move to x_j. when *try_number* exceed that of the fish execute randomly moving.

(5) Randomly movement: The artificial fish x_i can move to another status x_j with step restraints.

(6) Mutation: Randomly change the bit value of status x_i (such as 0 to 1, 1 to 0) to generate x_j, if the fitness of x_j is better $(fit(x_j < fit(x_i))$, change the status of x_i to x_j.

(7) Termination: Once all the artificial fish are matched with the task, it is regarded as an iterative process, and until the maximum number of iterations, the algorithm terminates.

E. EXECUTION SERVER SELECTION POLICY

Here, we choose an optimal physical node with available resources to deploy the selected execution container con_l $(l \in \{L\})$. The current load of each physical node depends on the available utilization of the CPU and memory resources proposed in Equations (9) and (10), respectively. We used the maximum value of the resource utilization rate as the load-balancing metric. Similar to paper [22], we calculate the standard deviation of the CPU and memory resource utilization rate for the physical nodes, and take it as the coefficient value of the corresponding resource utilization for each physical node. The maximum value of the resource utilization rate of physical node pm_j $(j \in \{P\})$ is calculated as follows:

$$AR_j = max(AR_j^{cpu} \times \sigma_1, AR_j^{mem} \times \sigma_2) \tag{27}$$

where σ_1 and σ_2 are the standard deviations for CPU utilization and memory utilization of the physical nodes in a CDC, respectively. In this stage, the AR of the physical nodes is sorted, and the physical node with the smallest AR

Algorithm 1 Artificial Fish Swarm Optimization Algorithm of Multi-Layer Container-Based Task Scheduling AF-MLCS. Execution Container Selection

```
Input:
   $PM = \{pm_j | j = 1, 2, \ldots, P\}$;
   $CON = \{con_l | l = 1, 2, \ldots, L\}$;
   $T = \{T_k | k = 1, 2, \ldots, K\}$;
   $FT = \{FT_{l,k} | l = 1, 2, \ldots, L; k = 1, 2, \ldots, K\}$;
   $PC = \{PC_{l,k} | l = 1, 2, \ldots, L; k = 1, 2, \ldots, K\}$;
   $PopSize, N_{max}, mutationProb, visual, \delta$;
   $try\_number, step$;
Output:
$Mapping\_list = \{(k, l, j) | k \in T, l \in CON, j \in PM\}$;
1: for each $T_k$ do
2:    $n \leftarrow 1$;
3:    $Pop \leftarrow$ initialize Population();
4:    Calculate the fitness values $fitness$ by Equation (26);
5:    $fronts \leftarrow$ CalculateFronts($Pop, fitness$);
6:    while ($n <= N_{max}$) do
7:       for each $Pop_i$ do
8:          $[X_t, Y_t, flag] \leftarrow$ Swarm();
9:          if ($flag = 0$)
10:            $[X_t, Y_t, flag] \leftarrow$ Follow();
11:            if ($flag = 0$)
12:               $[X_t, Y_t, flag] \leftarrow$ Prey();
13:               if ($flag = 0$)
14:                  $[X_t, Y_t] \leftarrow$ RandomMove();
15:               end if
16:            end if
17:         end if
18:         if ($Y_t < Pop_i.fitness$)
19:            $Pop_i.X \leftarrow X_t$;
20:         end if
21:         if (rand() $< mutationProb$)
22:            $[X_t, Y_t] \leftarrow$ mutation($Pop_i.X$);
23:            if ($Y_t < Pop_i.fitness$)
24:               $Pop_i.X \leftarrow X_t$;
25:            end if
26:         end if
27:      end for
28:      $P_{off} \leftarrow P_{off} \cup Pop$;
29:      Calculate the fitness values $fitness$ by Equation (26);
30:      $fronts \leftarrow$ CalcuteFronts($P_{off}, fitness$);
31:      $BestPop \leftarrow fronts[1]$;
32:   end while
33:   $l \leftarrow BestPop.X$;
34:   $pm_j \leftarrow$ algorithm2($k, l$);
35:   Add($k, l, j$) to $Mapping\_list$ and then return
      $Mapping\_list$;
36: end for
```

and sufficient available resources is selected as the optimal server, i.e., $opt\_server_j = min\{AR_1, AR_2, \ldots, AR_P\}$.

F. ALGORITHM IMPLEMENTATION

In this study, an Artificial Fish Swarm Optimization algorithm for Multi-layer Container-based Task Scheduling (AF-MLCS) is proposed. Algorithm 1 is the pseudo-code for executing container selection, and Algorithm 2 is the pseudo-code for physical node selection.

Lemma 1: The best-case and worst-case time complexities of the AF-MLCS algorithm are $O(L \times n \times n_f)$ and $O(L \times n \times (n_f + try\_number))$.

TABLE 2. Experimental parameters.

Element	Parameter	Value
Cloud Data Center	$\|CDC\|$	2
Physical machine	$\|PM\|$	12
	Store	100G
	CPU Capacity	$100 \sim 800$
	Memory Capacity	$100 \sim 800$
	Failure rate	0.025
Execution Container	CON	100
	CPU Capacity	$1 \sim 32$
	Memory Capacity	$0.38 \sim 6.25$
	Failure rate	$0.001 \sim 0.04$
BA Container	$\|BA\|$	100
IoT device	$\|U\|$	100
Non-IoT device	$\|V\|$	100

Proof: The AF-MLCS algorithm involves the selection of the execution containers and physical nodes. Let L be the total number of execution containers. For each task k, in the step 6 to 32 of the container selection phase (Algorithm 1), the outer loop requires $O(L \times n)$ operations to be performed. The algorithm requires internal operations, including Swarm behavior requires $O(n_f)$ operations, Follow behavior requires $O(n_f)$ operations, and Prey behavior requires $O(try\_number)$ operations. In the best case, the time complexity is $O(n_f)$ operations that only perform the Swarm behavior. The worst case is that all operations are performed, that requires $O(2n_f + try\_number)$ operations. Therefore, the time complexity for the executing container selection phase is $O(L \times n \times n_f)$ (for the best-case) and $O(L \times n \times (n_f + try\_number))$ (for the worst-case). Let N be the physical nodes scale in a CDC. In the physical node selection phase (Algorithm 2), the time complexity from step 1 to 17 is $O(2N)$ (for the best-case) and $O(3N)$ (for the worst-case). Therefore, the total time complexity for the AF-MLCS algorithm is $O(L \times n \times n_f + 2N) = O(L \times n \times n_f)$ (for the best-case) and $O(L \times n \times (n_f + try\_number) + 3N) = O(L \times n \times (n_f + try\_number))$ (for the worst-case), $L \gg N$. □

VI. PERFORMANCE EVALUATION

A. SIMULATION SETUP

Here, we designed experiments to evaluate the QoS parameters and verify the performance for the proposed scheduling strategy. We adopted the dataset cluster trace V2018 provided by Alibaba [36]. The dataset was analyzed to setup the parameter settings and dataset of the experiment, as follows:

1) PARAMETERS SETTING

Tables 2 and 3 list the experimental simulation parameters used. In Table 2, we consider 2 CDCs, and a topology composed of 12 heterogeneous physical nodes is built on the CloudSim platform with sufficient CPU and storage resources. The heterogeneous CDCs were as follows:

(1) Four types of CPU resource capacities for physical nodes: $CPU_j = [100.0, 200.0, 400.0, 800.0]$;

(2) Four types of memory resource capacities for physical nodes: $Mem_j = [100.0, 200.0, 400.0, 800.0]$;

(3) All physical nodes have the same failure rate: $fail_j = 0.025$;

Considering 100 execution containers, and the relevant parameters are as follows:

(1) The CPU resources of the execution container are random integer between 1-32;

(2) The memory resource of the execution container is a random number between 0.38-6.25;

(3) The failure rate $fail_l$ of the execution container was a random number between 0.001 and 0.04.

Users in the Band-area application container randomly send multi-task requests to CDCs using non-IoT and IoT devices. It is assumed that non-IoT and IoT devices are connected to CDCs through a long term evolution (LTE) wireless interface, and that CDCs use a WiFi interface to connect IoT and non-IoT devices. Let the transmission rates of the WiFi and LTE interfaces be randomly and evenly distributed in [2.01, 4.01] Mbps and [4.85, 6.85] MBPS respectively.

Table 3 lists the relevant parameters of the AF-MLCS algorithm proposed in this study.

TABLE 3. Experimental parameters setting of AF-MLCS algorithm.

Element	Parameter	Value
Maximum number of iterations	N_max	100
Population Size	$PopSize$	60
Try Number	try_number	50
Visual Range	$visual$	1
Crowdedness	δ	0.3
Moving Step	$step$	0.4
Mutation Probability	$mutationProb$	0.1

2) EXPERIMENT TEST DATA

The arrival time of user requests from non-IoT and IoT devices to the CDC follows an exponential distribution; that is, in each time slot t, the arrival process of the task flow requested by the user is a Poisson process [36]. Here, the priority between tasks and the task arrival time is considered to be negligible.

The related configurations for the experimental computer are as follows: 2×CPUs: Intel(R) Celeron(R) CPU 1000M @1.80GHz 1.80GH; memory: 8G RAM.

In addition, based on the analysis of the task dataset provided by Alibaba, and using the random function of the MathLab tool, the synthetic datasets for testing are randomly generated as follows:

(1) According to the execution container parameters set in Table 2, 100 container instance datasets used to execute tasks were randomly generated;

(2) Randomly generate 10 synthetic datasets with different task numbers; the number of execution container instances is 100, and the number of tasks is: 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000. There are two types of dataset:

-Synthetic dataset for task processing time: The processing time required for each task to be uploaded to each execution container instance for execution.

-Synthetic dataset for task energy consumption: The energy consumed by each task uploaded to each execution container instance for execution.

To compare the performance of each algorithm in this experiment, we recorded the average value of 1000 iterations.

Algorithm 2 AF-MLCS. Execution Server Selection

```
Input:
  $k \in T, l \in CON$;
Output:
  $j \in PM$;
1:  for each $pm_j$ do
2:      Apply Equations (9) and (10) to calculate the CPU
        and memory available utilization of the servers
        respectively;
3:  end for
4:  Calculate the standard deviation:
    $\sigma_1 \leftarrow std(AR^{cpu}), \sigma_2 \leftarrow std(AR^{mem})$;
5:  for each $pm_j$ do
6:      Calculate the available utilization $AR_j$ by
        Equation (27);
7:      Sort by $AR_j$ in ascending order;
8:  end for
9:  while do
10:     $j \leftarrow min(AR_j)$;
11:     if $(CPU_l^c < Re\_CPU_j \wedge Mem_l^c < Re\_Mem_j)$
12:         Deploy the execution container
            to the server $pm_j$;
13:         Return $j$;
14:     else
15:         Find the next suitable physical node;
16:     end if
17: end while
```

B. BASELINE ALGORITHMS

Based on the IoT cloud environment, we study task scheduling between multilayer containers, which is different from the task or job scheduling of VMs in the cloud environment. In this experiment, we compared the AF-MLCS algorithm with the GA-MOCS algorithm [19], APSO-EECS algorithm [25], and Binpack algorithm implemented in Docker Swarm to verify the effectiveness of the AF-MLCS algorithm.

-The Binpack algorithm deploys a new container to physical node with high CPU and memory resource utilization rates.

-The GA-MOCS strategy adopts NSGA-II to realize container allocation. In the GA-MOCS algorithm, only three optimization objectives related to this study are considered: load balancing for computing resources between physical nodes, reliability of application services, and data transmission overhead of application services.

-The APSO-EECS algorithm adopts accelerated particle swarm optimization (APSO) to solve a container-based scheduling strategy. This strategy considers two optimization objectives: task computational time and energy consumption. In addition, it considers the resource utilization rate of the computing nodes.

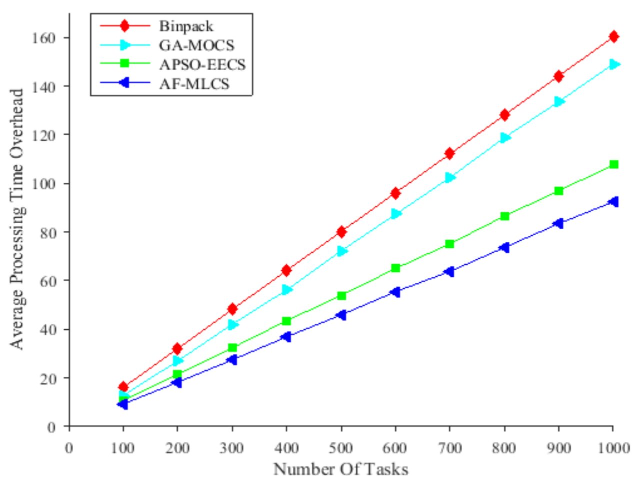


FIGURE 8. Comparison result of task processing time overhead.

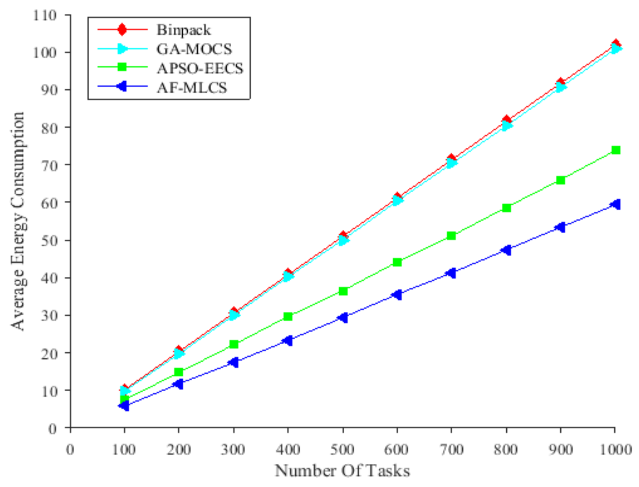


FIGURE 9. Comparison result of task energy consumption.

C. ANALYSIS FOR EXPERIMENTAL RESULT

Ten experiments were designed based on the task synthesis dataset, and we measured the performance for the four algorithms from four aspects: processing time overhead, energy consumption, reliability, and load balancing of the physical nodes in the CDCs.

1) PROCESSING TIME OVERHEAD

The comparison result of the task processing time overhead of the four algorithms with different task numbers is shown in Figure 8.

The Binpack algorithm deploys the execution container to a centralized node as far as possible; that is, it is allocated centrally in nodes with high utilization of the CPU and memory resources of the server, which easily causes node overload. Thus, the container cannot respond to requests in time, which increases the response time of the tasks. The GA-MOCS algorithm reduces the data transmission overhead of the task by assigning the relevant container to the physical node with a short network distance, thereby partially reduceing the processing time overhead of the tasks. However, the APSO-EECS algorithm ignored the image-transmission overhead of the execution container. The processing time overhead can be observed in Fig. 8. As the number of tasks increased, because the Binpack algorithm did not consider the optimization of the transmission time, its performance was the worst. However, the GA-MOCS and APSO-EECS algorithms both partially considered the transmission time, which was better than that of the Binpack algorithm. The AF-MLCS algorithm simultaneously optimized the execution time of tasks in the container, the transmission time of the task data, and the image-transmission time of the execution container. Compared with the Binpack, GA-MOCS, and APSO-EECS algorithms, the average improvement rates of the AF-MLCS algorithm in the average processing time overhead are 74.21%, 54.92%, and 17.29%, respectively. The results show that the AF-MLCS algorithm is better than other algorithms for synthetic datasets with different task numbers.

2) ENERGY CONSUMPTION

The energy consumed by a task is measured in two aspects: the energy consumed by data transmission and the energy consumed by the CPU resources of pysical node when executing a task. The comparison results of the energy consumed by the tasks are shown in Figure 9.

The experimental results show that the Binpack and GA-MOCS algorithms show an approximately linear growth and almost overlap, whereas the APSO-EECS algorithm shows a slow growth with an increase in the number of tasks. The main reason is that the Binpack algorithm and GA-MOCS algorithm did not take into account the impact of task execution time on execution energy consumption. The APSO-EECS algorithm considered the energy consumption of task execution and task data transmission, but did not consider the impact of container image transmission on energy consumption. The AF-MLCS algorithm simultaneously consideres the optimization of the energy consumption of task data transmission, task execution, and container image transmission. Compared with the Binpack, GA-MOCS and APSO-EECS algorithms, the average improvement rates of the AF-MLCS algorithm in the average execution energy consumption were 73.1%, 69.98%, and 25.28%, respectively. As shown in Figure 9, for synthetic datasets with different task numbers, the performance of our algorithm is better than that of the other algorithms.

3) EXECUTION RELIABILITY OF TASKS

The execution reliability of the tasks is measured by the failure rate of the execution containers and physical nodes. The comparison results of the execution failure rate of tasks are shown in Figure 10.

As shown in Figure 10, the Binpack and APSO-EECS algorithms almost overlap with the increase in the number of tasks. The main reason is that the Binpack and APSO-EECS algorithms did not take into account the execution failure rate of the task. Although the failure rate of the container and node are considered by the GA-MOCS algorithm, it focuses on

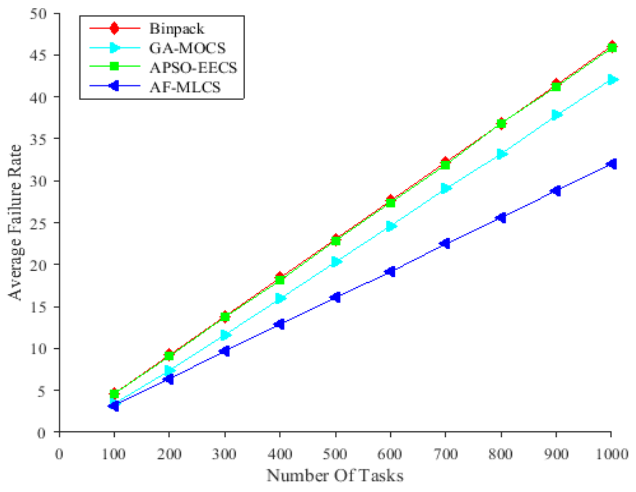


FIGURE 10. Comparison results of average failure rate of task execution.

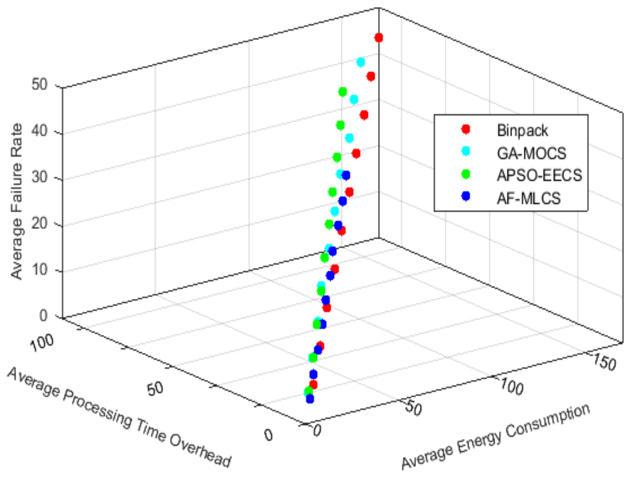


FIGURE 11. Comprehensive comparison of three objectives.

the balance between the workload of the container and the computing resources of the node. The AF-MLCS algorithm fully accounted for assigning tasks to execution containers and physical nodes with a low failure rate during task scheduling, thereby reducing the failure rate of tasks and improving the execution reliability of the task. As shown in Figure 10, compared with the Binpack, GA-MOCS, and APSO-EECS algorithms, the average improvement rates of the AF-MLCS algorithm for the average failure rate were 43.66%, 24.19%, and 42.59%, respectively. The results indicate that the AF-MLCS algorithm has a better objective value for synthetic datasets with different task numbers.

To facilitate comparison and analysis, Figure 11 shows a comparison of performance indicators, such as average processing time overhead, average execution energy consumption, and average failure rate of different algorithms. As shown in the figure, the AF-MLCS algorithm has a better scheduling performance than the Binpack, GA-MOCS, and APSO-EECS algorithms.

TABLE 4. STD for resource utilization of four algorithms.

Algorithms	value ranges of the standard deviation		
	CPU	Memory	σ_{CDC}
Binpack	0.327 ∼ 0.515	0.212 ∼ 0.325	0.2760 ∼ 0.4309
GA-MOCS	0.0097 ∼ 0.010	0.017 ∼ 0.152	0.0141 ∼ 0.1078
APSO-EECS	0.313 ∼ 0.497	0.247 ∼ 0.379	0.2824 ∼ 0.4395
AF-MLCS	0.015 ∼ 0.078	0.016 ∼ 0.091	0.0154 ∼ 0.0851

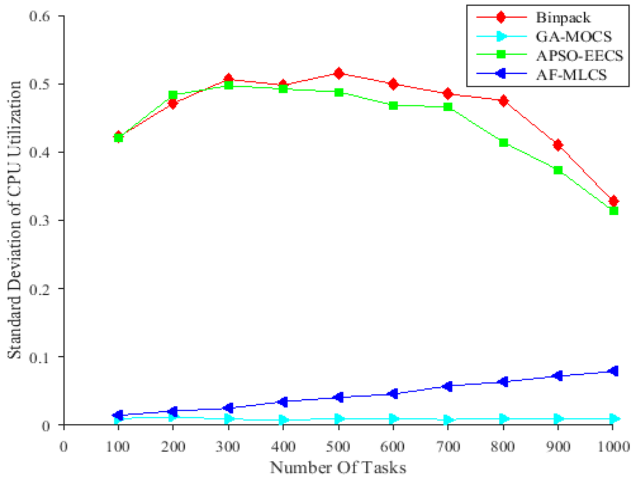


FIGURE 12. Comparison results of CPU resource utilization.

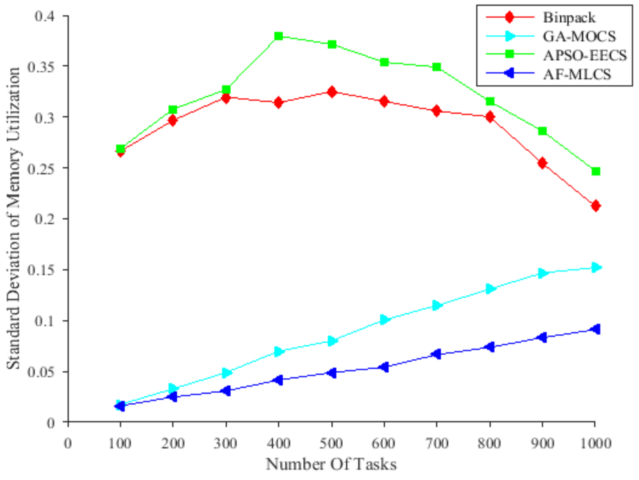


FIGURE 13. Comparison results of memory resource utilization.

4) RESOURCE LOAD

We used the standard deviation of the CPU and memory resource utilization to evaluate the load of the CDCs. The method in paper [22] is adopted.

$$\sigma_{CDC} = \sqrt{\frac{1}{2}\sigma_1^2 + \frac{1}{2}\sigma_2^2}$$

Figures 12, 13, and 14 show the experimental results. To facilitate discussion and understanding, Table 4 lists the value ranges of the standard deviation (STD) for the CPU and memory resource utilization of the four algorithms.

As shown in Table 4, the GA-MOCS algorithm is better than the other algorithms in terms of CPU resource load but worse than the AF-MLCS algorithm for memory resources.

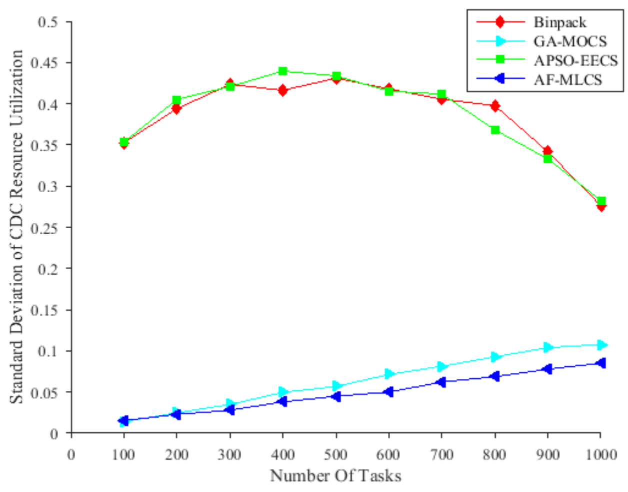


FIGURE 14. **Comparison results of two resource utilization.**

The main reason for this is that the optimization for CPU resource utilization is only considered in the GA-MOCS algorithm, and ignoring the optimization of memory resources. However, our algorithm considers the optimization of CPU and memory resources at the same time. The Binpack and APSO-EECS algorithms performed the worst in terms of CPU resources and memory resource load. The reason is that the Binpack algorithm only selects the node with the highest resource utilization to deploy the container. The APSO-EECS algorithm considers the node with the lowest weighted sum of CPU resources and memory resources to deploy containers, which could deploy containers to nodes with higher CPU or memory resource utilization; increases the load of the CPU or memory resources of the node, resulting in load unbalanced. However, the AF-MLCS algorithm uses the resource with the largest standard deviation of CPU and memory resource utilization of the node as the worst-case load, and the node with the lowest resource utilization is selected to deploy the container, which can avoid excessive resource load and balance the load of the CDCs. As shown in Figure 14, the AF-MLCS algorithm performed the best.

VII. CONCLUSION

In this study, based on application container technology, we established a new resource management model called Band-area Application Container (BAC). BAC contains users, application services, documents, messages, and related operating rules. One salient feature of BAC is that it can express a variety of things in reality, such as organizations or individuals. This study applies the basic knowledge of category theory to describe the BAC and the cooperation between BACs, and the end users can build enterprise application systems through BAC and cooperation between BACs. In addition, a BAC system framework was introduced. To solve the non-IoT and IoT tasks in BAC to select a suitable execution container and physical node, three objective models related to task execution were proposed: processing time overhead, energy consumption, and reliability. Through the weighted sum method, the three objective models were unified into an optimization model, and an artificial fish swarm algorithm was proposed to solve the container-based task scheduling problem in the IoT cloud environment. The proposed algorithm is verified through simulation experiments. Through analysis and comparison, this algorithm is obviously superior to other algorithms in terms of task completion time overhead, execution energy consumption, reliability, and balancing CDC load.

Our future work will involve deploying the proposed task scheduling algorithm in a real IoT cloud environment. Furthermore, we study the various QoS parameters of application services under various dependencies through heuristic algorithms and find a suitable container for application services through scheduling.

REFERENCES

[1] M. Litoiu, M. Woodside, J. Wong, J. Ng, and G. Iszlai, “A business driven cloud optimization architecture,” in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2010, pp. 380–385.

[2] G. Lawton, “Developing software online with platform-as-a-service technology,” *Computer*, vol. 41, no. 6, pp. 13–15, Jun. 2008.

[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] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: State-of-the-art and research challenges,” *J. Internet Services Appl.*, vol. 1, no. 1, pp. 7–18, May 2010.

[5] C. Vecchiola, S. Pandey, and R. Buyya, “High-performance cloud computing: A view of scientific applications,” in *Proc. 10th Int. Symp. Pervasive Syst., Algorithms, Netw.*, Taipei, Taiwan, 2009, pp. 4–16.

[6] J. A. Stankovic, “Research directions for the Internet of Things,” *IEEE Internet Things J.*, vol. 1, no. 1, pp. 3–9, Feb. 2014.

[7] M. Masdari, S. S. Nabavi, and V. Ahmadi, “An overview of virtual machine placement schemes in cloud computing,” *J. Netw. Comput. Appl.*, vol. 66, pp. 106–127, May 2016.

[8] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, “Open issues in scheduling microservices in the cloud,” *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 81–88, Sep./Oct. 2016.

[9] X. L. Li, F. Lu, G. H. Tian, and J. X. Qian, “Applications of artificial fish school algorithm in combinatorial optimization problems,” *J. Shandong Univ. (Eng. Sci.)*, vol. 34, no. 5, pp. 64–67, Oct. 2004.

[10] X. L. Li and J. X. Qian, “Studies on artificial fish swarm optimization algorithm based on decomposition and coordination techniques,” *J. Circuits Syst.*, vol. 2003, pp. 1–6, Feb. 2003.

[11] M. K. Hussein, M. H. Mousa, and M. A. Alqarni, “A placement architecture for a container as a service (CaaS) in a cloud environment,” *J. Cloud Comput.*, vol. 8, no. 1, pp. 1–15, May 2019.

[12] K. T. Seo, H. S. Hwang, I. Y. Moon, and O. Y. Kwon, “Performance comparison analysis of linux container and virtual machine for building cloud,” in *Proc. Netw. Commun.*, May 2014, pp. 105–111.

[13] J. Stubbs, W. Moreira, and R. Dooley, “Distributed systems of microservices using Docker and serfnode,” in *Proc. 7th Int. Workshop Sci. Gateways*, Budapest, Hungary, Jun. 2015, pp. 34–39.

[14] C. Vecchiola, X. Chu, and R. Buyya, “Aneka: A software platform for.NET-based cloud computing,” *High Speed Large Scale Sci. Comput.*, vol. 18, no. 3, pp. 267–295, 2009.

[15] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, “Elastic application container: A lightweight approach for cloud resource provisioning,” in *Proc. IEEE 26th Int. Conf. Adv. Inf. Netw. Appl.*, Fukuoka, Japan, Mar. 2012, pp. 15–22.

[16] R. Mondéjar, P. García-López, C. Pairot, and L. Pamies-Juarez, “CloudSNAP: A transparent infrastructure for decentralized web deployment using distributed interception,” *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 370–380, Jan. 2013.

[17] T. Mirzoev and R. Alvarez, “Leveraging VMware vCloud director virtual applications (vApps) for operational expense (OpEx) efficiency,” *J. World Comput. Sci. Inf. Technol. J. (WCSIT)*, vol. 3, no. 9, pp. 156–163, Apr. 2014.

[18] S. K. Garg, A. N. Toosi, S. K. Gopalaiyengar, and R. Buyya, "SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter," *J. Netw. Comput. Appl.*, vol. 45, pp. 108–120, Oct. 2014.

[19] 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.

[20] M. Adhikari, S. Nandy, and T. Amgoth, "Meta heuristic-based task deployment mechanism for load balancing in IaaS cloud," *J. Netw. Comput. Appl.*, vol. 128, pp. 64–77, Feb. 2019.

[21] M. Kaur and S. Kadam, "A novel multi-objective bacteria foraging optimization algorithm (MOBFOA) for multi-objective scheduling," *Appl. Soft Comput. J.*, vol. 66, pp. 183–195, May 2018.

[22] 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.

[23] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 48–56, Jun. 2017.

[24] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4712–4721, Oct. 2018.

[25] 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.

[26] Y. Li, "Artificial fish swarm algorithm for virtual machine placement," *J. Comput. Eng. Appl.*, vol. 51, no. 4, pp. 323–327, Nov. 2015.

[27] Y. Li, J. Zhang, W. Zhang, and Q. Liu, "Cluster resource adjustment based on an improved artificial fish swarm algorithm in mesos," in *Proc. IEEE 13th Int. Conf. Signal Process. (ICSP)*, Nov. 2016, pp. 6–10.

[28] J. Qin and Z. Zhao, "Research on composite service performance optimization based on Hadoop mapreduce," *J. Comput. Technol. Develop.*, vol. 26, no. 5, pp. 61–65, May 2016.

[29] X. L. Zhang, "Application of improved artificial fish swarm algorithm in cloud computing task schedule," *J. Electron. Design Eng.*, vol. 25, no. 6, pp. 14–18, Jun. 2018.

[30] H. Luo, "A distributed management method based on the artificial fish-swarm model in cloud computing environment," *Int. J. Wireless Inf. Netw.*, vol. 25, no. 3, pp. 289–295, Sep. 2018.

[31] P. Albert and M. Nanjappan, "An efficient kernel FCM and artificial fish swarm optimization-based optimal resource allocation in cloud," *J. Circuits, Syst. Comput.*, vol. 29, no. 16, Dec. 2020, Art. no. 2050253.

[32] J. Q. Xi, "Virtual operating area supporting customized definition and operating method and system architecture thereof," U.S. Patent 9 971 597, May 15, 2018.

[33] M. Barr and C. Wells, *Category Theory for Computing Science*, vol. 1, 13th ed. New York, NY, USA: Prentice-Hall, 1990, pp. 16–281.

[34] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[35] L. He and Q. H. Liu, *Multi-Objective Optimization Theory and Continuous Method*. Beijing, China: Science Press, 2015, pp. 96–97.

[36] Alibaba Corp. *Alibaba Cluster Trace V2018*. Accessed: Sep. 26, 2021. [Online]. Available: https://github.com/alibaba/clusterdata

[37] M. Blackburn. (2008). *Five Ways to Reduce Data Center Server Power Consumption*. Green Grid, USA. [Online]. Available: https://www.greenbiz.com/sites/default/files/document/White_Paper_7_-_Five_Ways_to_Save_Power.pdf

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, 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 810 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds over 60 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top ten most influential scientists in parallel and distributed computing 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 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.