Received: 15 December 2022 | Revised: 17 January 2023 | Accepted: 3 February 2023

DOI: 10.1002/cpe.7665

RESEARCH ARTICLE

WILEY

A container deployment strategy for server clusters with different resource types

Mingxue Ouyang[1] | Jianqing Xi[1] | Weihua Bai[2] | Keqin Li[3]

[1]School of Software Engineering, South China University of Technology, Guangzhou, China

[2]School of Computer Science, Zhaoqing University, Zhaoqing, China

[3]Department of Computer Science, State University of New York, Albany, New York, USA

Correspondence
Jianqing Xi, School of Software Engineering, South China University of Technology Technology, Guangzhou, China.
Email: jianqingxi@163.com

Weihua Bai, School of Computer Science, Zhaoqing University, Zhaoqing, China.
Email: bandwerbai@gmail.com

Funding information
Science and Technology Plan Project of Guangdong Province, China, Grant/Award Numbers: 2014B010112007, 2016B010124010

Abstract

The method of deploying microservices based on container technology is widely used in cloud environments. This method can realize the rapid deployment of microservices and improve the resource utilization of cloud datacenters. However, resource allocation and deployment of container-based microservices are key issues. With the continuous growth of computing- and storage-intensive services, it is necessary to consider the deployment of microservices of different business types. This study establishes a multi-objective optimization problem model with the similarity between containers and servers, load balance of clusters, and reliability of microservice execution as the optimization objectives. An improved artificial fish swarm algorithm is proposed for the container deployment of computing- and storage-intensive microservices. The comprehensive experimental results show that, compared with the existing deployment strategies, the matching degree between the container and server, cluster load balance value, service execution reliability, and other performance parameters are improved while shortening the running time of the algorithm. In addition, under the constraint of load balancing, the resource utilization of the computing and storage server clusters is improved.

KEYWORDS

artificial fish swarm algorithm, cloud computing, container, microservices, multi-objective optimization

1 | INTRODUCTION

Cloud computing is a commercial computing model. It provides users with on-demand computing power, storage space, and information services by distributing computing tasks to a resource pool composed of a large number of computing servers.[1] The resource pool is a self-managed and maintainable virtual computing resource, that is, a large server cluster that includes computing servers, storage servers, and broadband resources. These resources can meet the needs of users and increase application scale through the dynamic scaling of virtual machines (VMs) or containers.[2] In particular, the wide use of containers provides strong environmental support for application deployment based on the microservice architecture (MSN).[3] In addition, the resource pool uses data multi-copy fault tolerance, computing nodes that are isomorphic and interchangeable, and other measures to ensure high reliability of services.

Microservice architecture (MSN) is currently an important application development mode that divides the monolithic application into a group of independent fine-grained and componentized services.[4] A lightweight communication mode is adopted between services, and the service function chain (SFC) is formed through the call relationship to form the application.[5] Because of their high reusability, scalability, and flexibility, microservices are usually deployed in multiple computing servers or cloud data centers (CDC) in the form of VMs and containers. However, as a resource virtualization technology, virtual machine (VM) needs a separate operating system (OS) image, which will inevitably increase the cost of storage and

computing resources,[6] so that VM instances can extend the VM startup time on the one hand; On the other hand, it affects the portability of applications between CDC provided by different cloud providers. In contrast to VM, the OS level container technology is lightweight, with seconds of startup time and less storage. The application service process in a container operates directly in the host OS kernel.[7] These characteristics of the container provide convenience for the flexible deployment and scalability of microservices, and are deeply concerned with cloud service providers and researchers. Typical OS-level container technologies include Solaris Zones, Kubernets, Linux Container (LXC), and Docker containers. Docker containers are a widely used container technology.[8]

However, resource allocation and deployment of container-based microservices are important issues;[9] that is, while meeting the quality of service (OoS) for user obligations, it can also maximize the use of various resources in the IaaS layer.[10] Although there are many container-based microservice deployment strategies, there are typically three methods implemented by Swarm, the native cluster scheduling tool in Docker container: Binpack, Random, and Spread. However, these deployment strategies exhibit low performance and cannot meet the requirements of specific business-type services. In this study, we consider an application scenario, to formulate the corresponding scheduling strategies for different types of micro services in SFC (such as intensive computing and storage) to request different resource types. In other words, computing-intensive microservices are deployed to computing server nodes, storage-intensive microservices are deployed to storage server nodes, and microservices without special resource-type requests are scheduled to be executed in common server nodes. For example, in Docker's native environment, Swarm, a constraint filter is used to deploy a container of a specific type to a server node of a specific type.[11] On the one hand, according to the similarity between microservices and servers, specific types of microservices are scheduled to corresponding types of server nodes to improve the resource utilization of computing and storage servers; On the other hand, adjacent microservices of the same type in SFC should be deployed to the same node as much as possible to save transmission costs. In addition, the reliability of microservice execution was also considered.

In the cloud environment, for the above-mentioned microservice deployment problem to meet different business types, which involves the optimization of multiple objectives, it is difficult to obtain the optimal or suboptimal solution in effective time by applying an exhaustive search and greedy algorithm.[12] Deep-learning methods require training data, which are difficult to implement in random and dynamic deployment environments. However, the meta-heuristic algorithm has incomparable advantages for the multi-objective combinatorial optimization problem, which can obtain the solution of the multi-objective optimization problem model with multiple constraints under a limited number of iterations. Therefore, we used an improved artificial fish swarm optimization algorithm[13,14] to optimize the matching degree between containers and servers, load balancing of server clusters, and reliability of microservice execution.

The main contributions of this study are as follows:

1. We built a problem model for execution container deployment of service that includes the matching degree between containers and servers, load balancing of server clusters, and service execution reliability.
2. Aiming at the proposed target model, a multi-objective optimization problem model for service execution container deployment was established. The simultaneous optimization objectives of this model are matching degree between containers and servers, load balancing of server clusters, and reliability of service execution. The resource demand of the execution container is adjusted based on the number of user requests, and the service is deployed based on whether the total resources of the container and server match.
3. Based on the optimization problem model, we proposed an improved artificial fish swarm optimization algorithm to solve the service execution container deployment problem. In the algorithm, the front-edge (Pareto) solution set is obtained by calculating the crowding distance,[15] and the global optimal solution is obtained from the front-edge solution set according to the evaluation function values of the three optimization objectives.

The remainder of this paper is organized as follows. In Section 2, work related to microservice deployment and scheduling is introduced. Inspired by relevant work, the problem description, system model, and proposed optimization problem model are described in Section 3. Section 4 details the proposed deployment strategy for an artificial fish swarm, and the analysis and comparison of the experimental results and validation of the proposed deployment strategy are discussed in Section 5. Section 6 provides a summary of this study and a discussion of future work.

2 | RELATED WORK

In the cloud computing environment, resource management of server clusters in the CDC and container-based microservice scheduling and deployment are key issues, and their scheduling algorithms and deployment strategies affect the OoS of cloud users and the profits of cloud providers.[16] In this section, we discuss relevant scheduling algorithms and deployment strategies.

First, the meta-heuristic strategy is a search technique of swarm intelligence, which does not depend on the mathematical properties of multi-objective and multi-constraint optimization problems and the structural characteristics of solving problems. This type of algorithm can find the optimal or suboptimal solution from multiple feasible solutions in the target space through the continuous interaction between individuals and the environment, and has been applied by researchers to solve multi-objective optimization problems with multiple constraints in

the cloud. For example, Guerrero et al.[17] adopted NSGA-II to propose a container-based microservice deployment strategy that optimizes the threshold distance between containers, network distance between containers, execution reliability of microservices, and load balance of computing resources. Ullah et al.[18] used the artificial bee colony algorithm (ABC) to balance the workload among server nodes in the cloud and reduce energy consumption. Lin et al.[19] designed an improved ant colony optimization (ACO) algorithm that realized the container deployment of microservices by optimizing network transmission costs, load balancing of clusters, and average failure rate of microservices. Ma et al.[20] built a knowledge-driven evolutionary algorithm using the NSGA-II algorithm for reference points (i.e., NSGA-III) to solve the deployment and start-up of microservices. The algorithm considers the actual idle rate of microservices, idle rate of computing and storage resources, and load balancing of computing and storage resources. Muniswamy et al.[21] established a container dynamic scalable task scheduling (DSTS) strategy, which realizes container-based task scheduling by combining the pigeon heuristic optimization (PIO) algorithm and a neural network (NN). Its goal is to improve CPU resource utilization and reduce task execution costs. In a previous paper,[5] we established an improved accelerated particle swarm optimization (APSO) algorithm, which is based on the service function chain (SFC) to gather the execution containers of services to the same physical node as much as possible to solve the deployment problem of microservices, considering the transmission overhead between services, resource utilization rate of CDC clusters, and aggregation degree between containers as optimization objectives to improve the resource utilization of the CDC.

Second, there are many placement strategies and scheduling algorithms have been proposed for microservice execution containers in the field of cloud computing. For example, Zhang et al.[22] designed an effective adaptive scheduler using integer linear programming to reduce overhead by optimizing the energy consumption of docker container hosts, cost of container image pulling, and cost of workload network conversion between container hosts. Liu et al.[23] established a multi-objective docker container scheduling strategy that considered the utilization of CPU and memory for physical nodes, container image transmission time, correlation between containers and physical nodes, and container aggregation. The same author built a container scheduling algorithm for big data applications based on Kubernetes using particle swarm optimization (K-PSO) with the goal of improving resource utilization.[24] Xie et al.[25] applied the NSGA-II algorithm to build an improved harmonious search algorithm. The algorithm considers the mathematical expression of the docker model and the service optimization of microservice scheduling for a data platform as its objectives. Fan et al.[26] used particle swarm optimization (PSO) algorithm to build a container-based microservice deployment strategy for edge computing. The optimization goals of this strategy are the network delay between microservices, load balance of clusters, and reliability of microservice applications. Zhu et al.[27] designed an adaptive task scheduling algorithm (ADATSA) using learning automata based on a container, which improves resource utilization and QoS performance by optimizing resource imbalance, resource surplus, and task running state. Chen et al.[28] established a container-based microservice scheduling strategy (MOPPSO-CMS) using a parallel particle swarm optimization algorithm. The optimization objectives of this strategy are network transmission cost, load balancing, optimization speed, and service reliability. Liu et al.[29] proposed a multi-objective container deployment strategy (MOCP-MFEA) based on a multi-factor evolutionary algorithm (MFEA) for the container deployment of microservices. This strategy mainly aims at node loss, average network distance, resource cost, and load balancing to solve the container layout problem in a heterogeneous cluster environment

Finally, as a meta-heuristic search strategy, the artificial fish (AF) swarm algorithm is an uncertain, probabilistic, and globally optimized intelligent algorithm that has been applied to resource allocation and scheduling in cloud environments. For example, Gupta et al.[30] applied an artificial fish swarm optimization algorithm to select channels for mobile nodes in mobile ad hoc networks (MANET) to improve the network life of mobile nodes and reduce energy consumption. Feng et al.[31] proposed an improved artificial fish swarm algorithm to maximize network coverage and minimize energy consumption by optimizing the wireless sensor network coverage. Ajitha et al.[32] established a multi-objective artificial fish swarm resource optimization task scheduling strategy based on bivariate correlation opposition (BCO-MAFSROTS) to improve task scheduling efficiency with minimum completion time and resource utilization. The main optimization objectives are CPU time, bandwidth, memory, and energy. Albert et al.[33] used an artificial fish swarm algorithm to assign user tasks to clustered virtual machines in cloud computing to improve resource utilization and quality of user experience. Li et al.[34] established a computing scheduling algorithm based on an artificial fish swarm for offloading computing-intensive tasks of mobile devices to edge servers for execution, with the goal of reducing the transmission time and execution energy consumption. Uma et al.[35] proposed an improved fish swarm algorithm (FSA) to optimize the allocation of virtual resources in the cloud to meet the requirements of users to execute tasks in a specific time period. Manikandan et al.[36] built a fuzzy C-means drastic hybrid algorithm for task scheduling and resource allocation in the cloud to reduce the cost, energy consumption, and resource utilization. We designed a task scheduling strategy based on multilayer containers using an artificial fish (AF) swarm optimization algorithm. This strategy optimizes task processing time, energy consumption, task execution reliability, and other objectives.[37]

Based on the above work, to solve the problem that microservices of different business types are deployed to corresponding types of server nodes, we take the matching degree between containers and server nodes, reliability of service execution, and resource load balancing between server nodes as optimization objectives. A multi-objective optimization problem model of service execution container deployment is established, and the Pareto solution of the optimization problem model is realized through an improved artificial fish swarm optimization (AFSO) algorithm. In the service deployment process, relevant containers are clustered into one server node as much as possible according to the matching degree between containers and server nodes and the adjacent relationship between services, to improve the resource utilization of different types of server clusters.

3 | PROBLEM DESCRIPTION

In this section, we discuss the system model, optimization objectives, and methods proposed for solving the problem model. The relevant symbols for the system model are listed in Table 1.

3.1 | System model

The CDC model for container-based service deployment is illustrated in Figure 1. The BAC as a Service (BaaS) platform is a type of Internet of Things (IoT) cloud platform proposed by us, which is composed of a group of band-area application containers (BACs).[5] BAC is an aggregation of tool services, documents, users, messages, and other objects, as well as relevant operation rules. Tool services are a fine-grained application service model based on a microservice architecture.[37] BAC can express organizations or individuals, map the business relationship between organizations or individuals to the collaboration relationship between BACs, and integrate each business sub-system into a complex and complete application system through the link and collaboration between BACs. In each BAC, the service function chain (SFC) required by the user is deployed. SFC is a sub-application composed of a set of tool services through call relationships.

When a user sends a service request to an application in the BAC, the admission controller of the BAC engine sends a resource request to the dispatcher according to the user permissions and type of service resource request. The container-based scheduling engine evaluates the resource type and resource amount of the service request, and deploys the service to the appropriate type of server (physical node) for the CDC according to the evaluation value. For example, computing-intensive services are deployed to computing servers, storage-intensive services are deployed to storage servers, and microservices without specific resource-type requests are deployed to common nodes. To improve the resource utilization of various servers and the reliability of service execution, the deployment solution of this study is to allocate the service execution container to the

TABLE 1 Relevant symbols of the system model

Component	Symbol	Description
Application	$Apps$	A set of application
	$App_a \in Apps$	Application with identifier a
	SFC_a	SFC for application App_a
Tool service	TSs	Tool service set
	$Edgs$	A set of edge between services
	$TS_i \in TSs$	Tool service with identifier i
	CPU_i	CPU consumed for service TS_i
	$Store_i$	Store consumed for service TS_i
	Thr_i	Threshold value requested for service TS_i
	$Tsreq_i$	Number of requests for service TS_i
	$Cins_i$	Number of execution container for service TS_i
	$Type_i$	Type for service TS_i
User	$Users$	User set
	$User_v \in Users$	User with identifier v
	$Ureq_v$	Number of user requests
CDC	PMs	A set of server node set
	$pm_k \in PMs$	Server node with identifier k
	CPU_k^{pm}	CPU capacity for server node pm_k
	$Store_k^{pm}$	Store capacity for server node pm_k
	$Re\_CPU_k^{pm}$	Remaining CPU capacity for server node pm_k
	$Re\_Store_k^{pm}$	Remaining store capacity for server node pm_k
	$Type_k^{pm}$	Type for server node pm_k

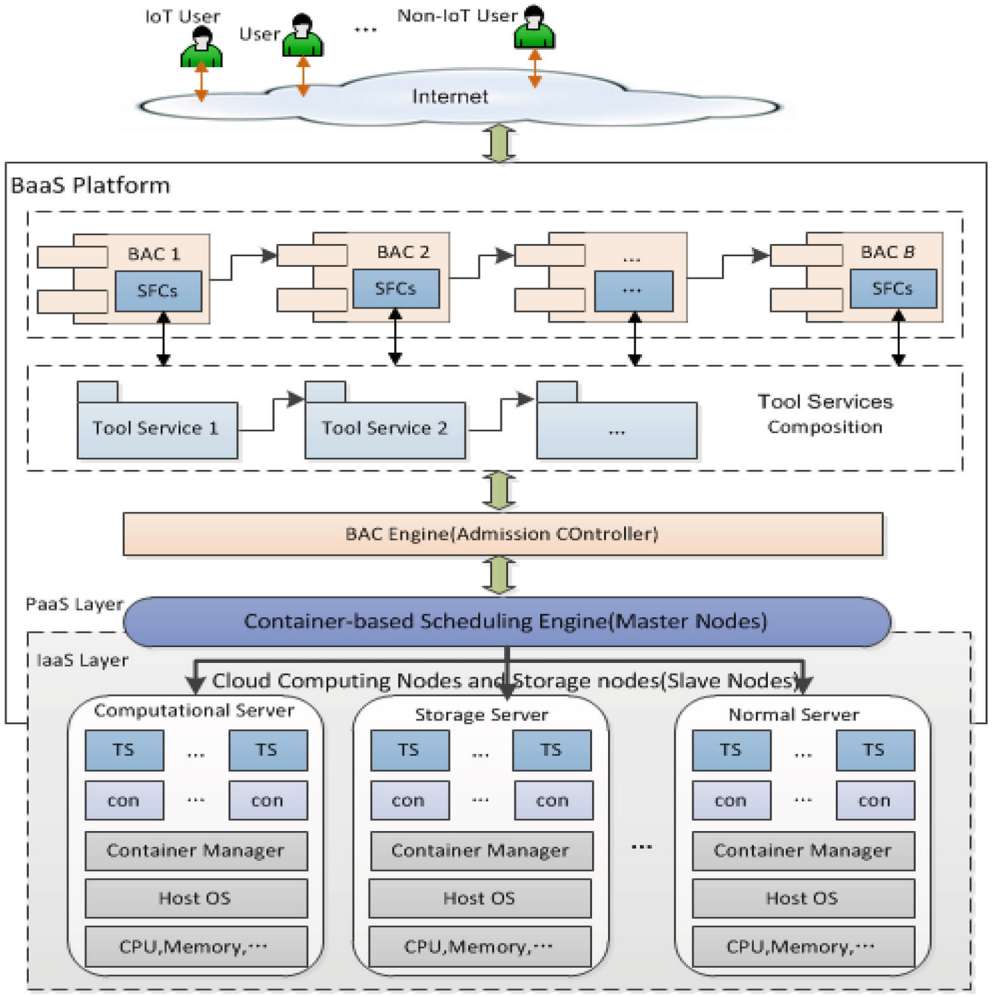

FIGURE 1 Container-based CDC

corresponding type of server with a low failure rate according to the service type of the application, and deploy adjacent services of the same type in the SFC to the same server node or the same CDC as much as possible to reduce the transmission cost between services.

On the PaaS (Platform as a Service) layer, suppose A application $Apps = \{App_a \mid 1 \le a \le A\}$ are deployed in the BAC of the BaaS platform. $SFC_a = < TSs_a, Edgs_a >$ is the service function chain of application App_a, where TSs_a is the tool service set in SFC_a and $Edgs_a$ is the edge set. SFC_a is essentially a directed graph composed of a group of services and the call relationship between the services. Here, tool services can be defined as $TS_i = < Type_i, Tsreq_i, CPU_i, Store_i, Thr_i >$, where $Type_i$ represents the service business type, including computing intensive services, storage intensive services, and common services; $Tsreq_i$ is the number of calls/requests for service TS_i; CPU_i and $Store_i$ are the amount of computing and storage resources required for the unit request service TS_i, respectively; Thr_i represents the threshold value of the requests. When the request threshold is exceeded, the execution container must be expanded. Users are consumers of tool service sequences, and the user set is formalized as $Users = \{User_v \mid 1 \le v \le V\}$, the user can describe as $User_v = < Ureq_v, Plat_v >$, $Ureq_v$ and $Plat_v$ are the number of user requests and the platform used, respectively.

In the IaaS (Infrastructure as a Service) layer, different types of server clusters are connected via an internal high-speed network with M server nodes formally defined as $PMs = \{pm_k \mid 1 \le k \le M\}$. The server node is described as $pm_k = < Type_k^{pm}, CPU_k^{pm}, Store_k^{pm} >$, where $Type_k^{pm}$ represents the server type, including computing, storage, and common type servers; CPU_k^{pm} and $Store_k^{pm}$ are computing and storage resources, respectively.

In the tool service deployment process, the tool service is packaged and encapsulated in the execution container, and the form is defined as $alloc\ (TS_i) \equiv con_c$ and $c \ge 1$. When a new container instance must be started, the container image is pulled to the server node. Considering that the number of service requests exceeds the request threshold, that is, $RRq_i = (Ureq_a \times Tsreq_i) > Thr_i$, the number of container instances to be expanded is $Cins_i = \lceil RRq_i / Thr_i \rceil$. To improve the resource utilization of the server node, each server can deploy multiple container instances that are recorded as $alloc\ (con_c) \equiv pm_k$. Then, a matrix can be used to describe the deployment relationship between the tool services and server nodes. The formal definitions are as follows:

$$Alloc(TSs, PMs) = [x_{ik}]_{N\times M} \tag{1}$$

$$x_{ik} = \begin{cases} 1, & if\ alloc(TS_i) \equiv pm_k; \\ 0, & otherwise; \end{cases}$$

$$i \in \{N\} \wedge k \in \{M\}. \tag{2}$$

where $x_{ik} = 1$ indicates that the container instance of service TS_i is deployed in the server node pm_k.

3.2 | Optimization objectives

The optimization objectives of this study are the similarity between containers and server nodes, load balancing between server nodes, and service execution reliability. Through objective optimization, deploy specific types of services in SFC to corresponding types of server nodes with low failure rates for execution to improve resource utilization and service execution reliability of computing and storage server nodes. The objective functions are as follows:

3.2.1 | Similarity between containers and server nodes

In the service deployment process, the similarity between the container and server nodes must be considered to improve the resource utilization of computing and storage servers.[23] Here, we measure the similarity by the matching degree between the container and server node, including two aspects:

First, the feature correlation between the container and server node: Only when the service type matches the node type and the resource capacity is met that the service execution container deployed to the node. Here, we use the method proposed by Liu to calculate the correlation between containers and nodes.[23] Assume that the feature set of server node pm_k is $FT_k^{pm} = \{ft_z^{pm} \mid 1 \le z \le Z\}$, where Z is the number of features for the server node. When creating a container instance con_l for TS_i, that is, $alloc\,(TS_i) \equiv con_l$, its features be $FT_l^{con} = \{ft_z^{con} \mid 1 \le c \le C\}$, where C is the number of container features. The feature intersection between the container con_l and node pm_k is given by Equation (3):

$$FT' = FT_l^{con} \cap FT_k^{pm} \tag{3}$$

For example, let con_2 be the container instance of service TS_3, its features be $FT_2^{con} = \{Type_2^{con} = '1', CPU_2^{con} = 23.1, Store_2^{con} = 2.3, Edgs_3 = (TS_3, TS_5), \ldots\}$, and the feature set of the server node pm_{12} be $FT_{12}^{pm} = \{Type_{12}^{pm} = '1', CPU_{12}^{pm} = 800, Store_{12}^{pm} = 400, \ldots\}$. At this time, the type feature values of the container con_2 and node pm_{12} are the same, that is, $Type_2^{con} = Type_{12}^{pm} = '1'$, $CPU_2^{con} \le CPU_{12}^{pm} \wedge Store_2^{con} \le Store_{12}^{pm}$, which means that the container con_2 can be deployed to the node pm_{12}.

Second, feature correlation between containers and containers deployed in server nodes: Considering the interaction cost between services, two adjacent services of the same type in the same SFC should be deployed to the same node as far as possible to reduce the transmission cost.[3] Suppose the node pm_k has deployed L containers, and its feature set is:

$$FTs_k^{con} = \bigcup_{i=1}^{L} FT_i^{con} \tag{4}$$

Then, the feature intersection between the container and deployed container in the server node can be calculated using Equation (5):

$$FT'' = FT_l^{con} \cap FTs_k^{con} \tag{5}$$

For example, con_5 is a container instance of service TS_5, and its feature collection is $FT_5^{con} = \{Type_5^{con} = '1', CPU_5^{con} = 25, Store_5^{con} = 3.2, Edgs_3 = (TS_3, TS_5), \ldots\}$, where $Edgs_3 = (TS_3, TS_5)$ indicates that the services corresponding to containers con_5 and con_2 are on the same edge, and $Type_5^{con} = Type_2^{con} = '1'$, which means that con_5 can also be deployed in node pm_{12} and has an aggregation with con_2.

Synthesizing Equations (3), (4), and (5), we use the matching degree of the container and server nodes to measure their similarity. The matching degree between the container con_l and node pm_k is given by Equation (6):

$$Md_{l,k} = \frac{|FT' \cup FT''|}{|FT_k^{pm} \cup FTs_k^{con}|} = \frac{|(FT_l^{con} \cap FT_k^{pm}) \cup (FT_l^{con} \cap FTs_k^{con})|}{|FT_k^{pm} \cup FTs_k^{con}|} \tag{6}$$

In CDC, we use the reciprocal of the matching degree value (1/Md) as the similarity between containers and nodes. This is formalized in Equation (7):

$$\begin{aligned} MD(X) &= \sum_{k=1}^{M}\sum_{i=1}^{N} x_{ik} \sum_{l=1 \wedge alloc(TS_i)\equiv con_l}^{Cins_i} \frac{1}{Md_{l,k}} \\ &= \sum_{k=1}^{M}\sum_{i=1}^{N} x_{ik} \sum_{l=1 \wedge alloc(TS_i)\equiv con_l}^{Cins_i} \frac{|(FT_l^{con} \cap FT_k^{pm}) \cup (FT_l^{con} \cap FTs_k^{con})|}{|FT_k^{pm} \cup FTs_k^{con}|} \end{aligned} \tag{7}$$

3.2.2 | Load balancing between server nodes

In CDC, it is necessary to consider load balancing while improving resource utilization. The load balancing of the server clusters should include two aspects. First, in a single-server node, the consumption of resources (such as CPU and storage) in all dimensions should be balanced as far as possible to avoid resource fragmentation. Second, the load between the server nodes should be balanced to deal with low-load and high-load problems. To balance the load of the CPU and the storage resources in the CDC, the load must be evenly distributed to each server node.

The CPU and storage resource utilization of the server node pm_k is the ratio of resources used to total resources. The calculation Equations are as follows:

$$UR_k^{CPU} = \frac{\sum_{i=1}^{N} x_{ik} \frac{RRq_i}{Cins_i} CPU_i}{CPU_k^{pm}} \tag{8}$$

$$UR_k^{Store} = \frac{\sum_{i=1}^{N} x_{ik} \frac{RRq_i}{Cins_i} Store_i}{Store_k^{pm}} \tag{9}$$

We take the standard deviation of the CPU and storage resource utilization of the server node as the coefficient of the corresponding resources and measure the load balance value of the server node with the highest resource load pressure.[37] This is defined as Equation (10):

$$CLB(X) = \frac{1}{std_{CPU} + std_{Store}} \max_{1\le k\le M} max(std_{CPU} \times UR_k^{CPU}, std_{Store} \times UR_k^{Store}) \tag{10}$$

where std_{CPU} and std_{Store} represent the deviation degree between the CPU and storage resource utilization rate, respectively; that is, the standard deviation of the resource utilization rate. The first max() function is used to balance the CPU and storage resources in a single server node, and the $\max_{1\le k\le M}()$ function is used to balance the load between the server nodes.

3.2.3 | Service execution reliability

When users request the execution of tool services through the platform, the stability of the service execution is related to the quality of experience (QoE) for users. Therefore, this study considered the failure factors related to the service execution process as a service execution reliability measurement. The failure factors considered include the failure rate $fail^{pm}$ of the server node where the service container is deployed, failure rate $fail^{con}$ of the execution container, and failure rate $fail^{pl}$ caused by the platform where the service is located and physical environment.

$$SEF(X) = \sum_{k=1}^{M}\sum_{i=1}^{N} x_{ik} \sum_{l=1 \wedge alloc(TS_i)\equiv con_l}^{Cins_i} (fail_k^{pm} + fail_l^{con} + fail_i^{pl}) \tag{11}$$

where $fail_l^{con}$ is the average failure rate of the execution tool service in the CDC, and $fail_i^{pl}$ is the average failure rate when the tool service platform starts execution. The formulas are shown in Equations (12) and (13):

$$fail_l^{con} = fail_k^{pm} \times \frac{RRq_i}{Cins_i} \tag{12}$$

$$fail_i^{pl} = fail_i^{TS} \times \frac{RRq_i}{Cins_i} \tag{13}$$

3.3 | Problem model statement

To obtain the optimal target value for the optimization objectives $MD(X)$, $CLB(X)$, and $SEF(X)$, X is the decision variable. We modeled the multi-objective optimization problem as a nonlinear optimization model, and the formulas are defined as follows:

$$minimize \quad MD(X) \tag{14}$$

$$minimize \quad CLB(X) \tag{15}$$

$$minimize \quad SEF(X) \tag{16}$$

$$s.t \quad CC_1 = \frac{RRq_i}{Cins_i} \times CPU_i \leq Re\_CPU_k^{pm}, \forall pm_k, k \in \{M\} \tag{17}$$

$$CC_2 = \frac{RRq_i}{Cins_i} \times Store_i \leq Re\_Store_k^{pm} \tag{18}$$

$$CC_3 = RRq_i \leq Thr_i \tag{19}$$

$$CC_4 = \sum_{k=1}^{M} x_{ik} \geq 1, i \in \{N\} \tag{20}$$

$$CC_5 = \sum_{i=1}^{N} x_{ik} = 1, k \in \{M\} \tag{21}$$

Equations (14)–(16) are the optimization objectives: matching degree between containers and server nodes, load balancing between nodes, and service execution reliability.

Equations (17)–(21) are the constraints of the optimization model. Among them, the constraints CC_1 and CC_2 specify that the total amount of CPU and storage remaining resources for the server node should be greater than the amount of resources requested by the execution container instance of the service. CC_3 indicates that the actual number of requests for services should be less than the service request threshold. CC_4 means that each tool service has at least one execution container instance deployed in the CDC. CC_5 stipulates that at most one execution container instance of the same service can be deployed to the same server node to avoid multiple container instances of the same service competing for resources.

For the optimization model of the above problem, it is not advisable to use the exhaustive search method and greedy algorithm to obtain the optimal solution of the problem. This is because it is highly complex and time-consuming to calculate. In addition, the optimization problem involves multiple constraints, including service resource requests, server node resources, and deployment rules. Therefore, for this problem model, the optimal or sub-optimal solution of the NP-complete optimization problem can be obtained within acceptable complexity and time, we considered a meta-heuristic strategy to handle the deployment of each type of service. The artificial fish (AF) algorithm is a meta-heuristic intelligent optimization strategy with good global convergence and robustness. For this reason, we have studied an improved artificial fish swarm algorithm, which uses fitness value and crowding distance[15] as a method to evaluate the solution to find the optimal or sub-optimal solution for the problem.

4 | THE PROPOSED AF META-HEURISTIC

This section describes the proposed AF-CSDS strategy, evaluation function, algorithm implementation, and time-complexity analysis.

4.1 | Overview of AF algorithm

The artificial fish (AF) swarm algorithm was first proposed in 2002 and has been widely used in multi-objective optimization problems,[38] such as resource allocation and scheduling. AF is a new bottom-up optimization mode that can obtain a sub-optimal or optimal solution to the problem in the solution space by simulating the feeding activities of fish swarms.

During the foraging activities of artificial fish, the amount and concentration of food in the water are usually perceived by vision or taste, and the simple behavior of individuals or groups tends toward food. The AF includes four behaviors: Prey, Follow, Swarm and random movement. Each behavioral change of artificial fish depends on its own state and environment (partner) state, and affects the activities of its companions through its own activities. In the optimization process of artificial fish, bulletin boards are used for identification to record the state of historically optimal artificial fish (global optimal). Iteration is carried out by comparing the status of individuals and bulletin boards to make the fish swarm tend toward the target.

In an n-dimensional target search space, there are F artificial fish, which are expressed as $AF = \{AF_1, AF_2, \ldots, AF_F\}$. Let $X_i = (x_i^1, x_i^2, \ldots, x_i^n)$ be the current state of the artificial fish AF_i and f_i be the objective value. Then, at any time/iteration t, the four behaviors of the artificial fish AF_i can be summarized as follows:

1. Random movement: X_i' is randomly generated position within the field of vision by applying Equation (22):

$$X_i' = X_i + Visual \times Rand() \tag{22}$$

where *Visual* represents the field of vision for the artificial fish AF_i and *Rand*() is used to generate a random number between 0 and 1. The artificial fish X_i approaches the state X_i' under the limit of the step size.

2. Prey: Randomly generated X_i' within the field of vision. If $f_i' \le f_i$, the artificial fish will move forward to position X_i'; otherwise, X_i' will be regenerated until the retry number *try_num* is exceeded, and the artificial fish will perform random behavior. The forward formula is given by Equation (23):

$$X_i(t+1) = X_i(t) + \frac{X_{i'} - X_i(t)}{\|X_{i'} - X_i(t)\|} \times Step \times Rand() \tag{23}$$

where *Step* is the maximum moving step and $d_{i',i} = \|X_{i'} - X_i(t)\|$ is the Euclidean distance between $X_{i'}$ and X_i.

3. Follow: Let X_{min} be the partner with the best target position within the visual field of X_i, that is, $d_{i,min} \le Visual$. If $f_{min} < f_i$ and, the nearby area is not too crowded, that is, $f_{min}/n_f < \delta f_i$. Here, n_f represents the number of neighbor partners within the field of vision and δ is the crowding degree factor. Then, the fish moves to position X_{min} based on Equation (23); otherwise, the fish will perform Prey behavior.
4. Swarm: Let X_c be the partner of the regional center near the artificial fish X_i. If $f_c < f_i$ and the nearby area is not too crowded, that is, $f_c/n_f < \delta f_i$, then the fish moves to position X_c based on Equation (23); otherwise, the fish will perform Follow behavior.

4.2 | AF coding

In the artificial fish swarm algorithm, each fish in the swarm is the solution of the target space. To adapt to the solution of the optimization model proposed in Section 3.3, the coding problem of artificial fish must be solved. An array was used to describe each artificial fish. The array corresponds to the tool service set to be deployed, and the service identifier is used as the index. Because each service is deployed to different server nodes according to the number of extensions of the container instance, the server-node ID list serves as the specific content of the service. Table 2 shows the array structure of the artificial fish AF_i ($1 \le i \le F$).

In Table 2, the artificial fish contains N services, and the execution container of each service TS_j is deployed in $Cins_j$ server nodes. For example, the TS_2 is deployed in four nodes and the location status code is X_{TS_2}={3, 12, 13, 21}. Then, as the candidate solution of the optimization model, the position state of AF_i is $X_i = (X_{TS_1}, X_{TS_2}, \ldots, X_{TS_N})$ and the position length is $K = \sum_{j=1}^{N} Cins_j$.

Based on the analysis of the above artificial fish structure, within the field of vision, the distance between two artificial fish was defined as the sum of the number of different elements at the corresponding position. Let $X_a = [x_1^a, x_2^a, \ldots, x_K^a]$ and $X_b = [x_1^b, x_2^b, \ldots, x_K^b]$ be the position states of artificial fish AF_a and AF_b, respectively. The distance between AF_a and AF_b can be expressed as:

$$d_{ab} = \sum_{k=1}^{K} sign(|x_k^a - x_k^b|) \tag{24}$$

where *sign* is a symbolic function:

$$sign(x) = \begin{cases} 0, & x = 0; \\ 1, & x > 0. \end{cases} \tag{25}$$

TABLE 2 Structure of artificial fish AF_i

AF_i	Deployment node list
TS_1	{5, 8, 31}
TS_2	{3, 12, 13, 21}
…	…
TS_N	{5, 21, 7, 3, 29}

In the field of vision, the set of neighboring partners of the artificial fish AF_a is $Nbs = \{X_b | d_{ab} \leq Visual\}$, where $n_f = |Nbs|$ is the number of neighboring partners. Let $X_b = [x_1^b, x_2^b, \ldots, x_K^b]$ $(1 \leq b \leq n_f)$ be the position of the neighbor partner for artificial fish AF_a; then, the status bit x_i^c $(1 \leq i \leq K)$ of the neighbor center position X_c takes the server node ID where the status bit x_i^b of all neighbor partners appears most.

4.3 | Fitness evaluation function

Here, we used the AF optimization strategy to deploy each type of service to the corresponding type of server node. In the service deployment process, the state X_i of each artificial fish AF_i is a solution to the problem model. The fitness of state X_i must be evaluated to determine the optimal or sub-optimal solution. The evaluation method aggregates the three optimization objective functions in Section 3 into one evaluation function[39] and normalizes each objective function using the normalization method. The evaluation function was as follows:

$$Eav(X) = \beta_1 \frac{MD(X) - MD_{min}}{MD_{max} - MD_{min}} + \beta_2 \frac{CLB(X) - CLB_{min}}{CLB_{max} - CLB_{min}} + \beta_3 \frac{SEF(X) - SEF_{min}}{SEF_{max} - SEF_{min}} \tag{26}$$

where $\forall \beta_i \in [0, 1]$, $\sum_{i=1}^{3} \beta_i = 1$, and the weight value β_i can be adjusted according to actual business needs. The minimum and maximum values of each objective were used to eliminate the amplitude. The minimum values are MD_{min}, CLB_{min}, SEF_{min}; The maximum values are MD_{max}, CLB_{max}, SEF_{max}. X is the candidate solution.

In each iteration of AF optimization, first, non-dominated or non-inferior individuals are selected through the dominance relationship among individuals, and the non-dominated individuals are gathered into the archive or Pareto sets. Second, the crowding distance is used to calculate the distance between non-dominated individuals and the optimal solution in the search space. Finally, the individuals in the Pareto set and with the minimum crowding distance and evaluation value are selected as bulletin boards, and the updating of bulletin boards can promote the artificial fish swarm to approach the optimal solution continuously.

4.4 | Service deployment process

To host the execution container of each type for service to the appropriate type of server node, in each iteration of the AF optimization strategy, the individual must constantly change the position state and approach the optimal solution. The service deployment process using the AF optimization strategy is as follows:

1. Artificial fish swarm initialization: Set the relevant parameters F, δ, *step*, *Visual*, *try_num*, and *max_iter*. Based on the constraints of the problem model, the artificial fish swarm $AFs = \{AF_1, AF_2, \ldots, AF_F\}$ and initial position $\{X_i | i \leq F\}$ are generated;
2. Iteration steps:
   a Calculate the function value $Eva(X_i)$ of each artificial fish AF_i using Equation (26). The archive set *Arc* was selected based on the dominance relationship, and the artificial fish with the optimal position state was used as the bulletin board;
   b Evaluate each artificial fish and select the behaviors to be performed, including Prey, Follow, Swarm and Random movement behavior;
   c Perform behavior: In the process of executing each behavior, judge whether the resource type requested and resource quantity required of the execution container for each service in the artificial fish AF_i match the selected server node based on the constraint rules. If it matches, then deploy the container to this node; otherwise, look for the next server node. Here, the deployment criterion is that each service must be deployed $Cins_j$ times and deployed to different nodes each time to guide all execution containers of each service to complete the deployment;
   d Update the position status X_i of each artificial fish AF_i and repeat step a;
3. Algorithm completion: When the number of iterations is greater than the maximum number of iterations *max_iter*, the algorithm operation is terminated and the deployment of the service execution container is completed. The algorithm outputs the position state *Bulletin.X* of the bulletin board, that is, the mapping set between the service and the server node.

4.5 | Algorithm implementation

The pseudocode of the container-based service deployment strategy for the artificial fish (AF) swarm optimization algorithm (AF-CSDS) proposed in this paper is shown in Algorithms 1 and 2.

In Algorithm 1 of the AF CSDS strategy, when performing behaviors such as Prey, Follow, Swarm, and Random movement behavior, Algorithm 2 is called according to each behavior rule in order to find the appropriate type of server node for each service according to the constraints. Its pseudocode is shown in Algorithm 2.

Algorithm 1. Find an appropriate server node for the execution container

Input:
$TSs=\{TS_i|1 \le i \le N\}$; $PMs=\{pm_k|1 \le k \le M\}$;
$Edgs=\{(TS_i, TS_j)|TS_i, TS_j \in TSs\}$;
$F, Step, Visual, \delta, try\_num, max\_iter$;
Output: $TS\_PMs=\{(TS_i, pm_k)|TS_i \in TSs \wedge pm_k \in PMs\}$;
$t \leftarrow 1$;
$AFs \leftarrow$ InitializationAF(F);
$Evas \leftarrow$ Evaluate(AFs);
$Arc \leftarrow$ Cal_CrowdDistance(AFs);
while ($t \le max\_iter$) **do**
  $Bulletin \leftarrow$ UpdateBulletin($Arc, Evas$);
  for each AF_l **do**
    $[X_O, Eva_O, isMat]$ = Execute_Swarm($AF_l, Visual, \delta$);
    if (not $isMat$) **then**
      $[X_O, Eva_O, isMat]$ = Execute_Follow($AF_l, Visual, \delta$);
      if (not $isMat$) **then**
        $[X_O, Eva_O, isMat]$ = Execute_Prey($AF_l, Visual, \delta, try\_num$);
        if (not $isMat$) **then**
          $[X_O, Eva_O, isMat]$ = Execute_RandomMove($AF_l, Visual, \delta$);
        end if
      end if
    end if
    if ($AF_l.Eva \le Eva_O$) **then**
      $AF_l.X \leftarrow X_O$;
    end if
  end for
  $Evas \leftarrow$ Evaluate(AFs);
  $Arc \leftarrow$ Cal_CrowdDistance(AFs);
end while
$TS\_PMs \leftarrow Bulletin.X$;
Return TS_PMs;

Algorithm 2. Server node selection for service

Input: $TMSs, PMs$;
Output: TS_PMs;
for each $TS_i \in TSs$ **do**
  while ($l \le Cins_i$) **do**
    Randomly generate server node ID k;
    if ($TS_i.Type = PM_k.Type$) **then**
      if ($CPU_i < Re\_CPU_k \wedge Store_i < Re\_Store_k$) **then**
        $TS\_PMs \leftarrow (i, k)$;
        Break;
      end if
    end if
  end while
end for

4.6 | Algorithm complexity analysis

The time complexity of the AF-CSDS deployment strategy comprises Algorithms 1 and 2. Here, let F be the scale of the artificial fish, N be the number of tool services, and M server nodes exist in the CDC. In Algorithm 1, from Steps 7 to 21, the number of operations performed by the external loop is $O(F)$. When traversing the artificial fish swarm, the number of operations performed from Steps 8 to 20 are: the number of Prey operations is $O(try\_num)$, the number of Follow operations is $O(n_f)$, and the number of Swarm operations is $O(n_f)$. In Algorithm 2, because the execution container of each service needs to deploy $Cins$ times, it must perform $O(N \times Cins)$ times from Steps 1 to 11. Because Algorithm 2 needs to be called by each behavior, the number of Prey operations is $O(try\_num \times N \times Cins)$, the number of Follow operations is $O(n_f \times N \times Cins)$, and the number of Swarm operations is $O(n_f \times N \times Cins)$. It can be seen from Algorithm 1 that the execution behavior is selective. Ideally, only Swarm behavior needs to be performed, and the number of Swarm operations is $O(F \times n_f \times N \times Cins)$. The worst-case is to execute all the behaviors, and the number of operations is $O(F \times (2n_f + try\_num) \times N \times Cins)$. Considering the number of iterations max_iter, the total time complexity of the AF-CSDS algorithm is $O(max\_iter \times F \times n_f \times N \times Cins)$ (the best) and $O(max\_iter \times F \times (2n_f + try\_num) \times N \times Cins)$ (the worst).

5 | EXPERIMENT AND RESULT EVALUATION

This study used the tracking V2018 real dataset[40] provided by Alibaba Cloud to evaluate the performance of the proposed AF-CSDS strategy. Based on the analysis of tracking the V2018 dataset, in an application, the service function chain (SFC) is formed by the call relationship between 18 services, and the experiment is carried out considering the different number of user requests for the application. In the experiment, based on the proposed QoS parameters, the proposed algorithm is compared with the existing deployment algorithm to verify the effectiveness of the proposed algorithm.

5.1 | Experimental data and parameter settings

Based on the analysis of tracking V2018 dataset, Table 3 shows the tool service stack of an application, with 18 tool services distributed across three SFCs. Each row described the basic characteristics for a service. *SFC* represents the service function chain of the service. *Type* represents the service type: 1-computing intensive services; 2-storage intensive services; and 3-common services. *CPU* and *Store* represent the amount of computing and storage resources required by the service in the unit request (×1.0), respectively. Where *Tsreq* is the number of calls to a service; *Thr* is the threshold for calls/requests; and $Cins = \lceil \frac{Tsreq}{Thr} \rceil$ is the number of execution containers to be expanded when a service is requested per unit (×1.0). *Adj_TS* is the successor of the service.

In CDC, we used the CloudSim platform to build two server cluster environments of different sizes, including 120/240 server nodes. Three types of heterogeneous server clusters are considered:1-computing servers, 2-storage servers, and 3-common servers, which are divided by 1:1:1. The relevant CDC parameters are listed in Table 4.

The simulation parameter configurations of the AF-CSDS algorithm are presented in Table 5.

The hardware environment of this experiment is as follows: processor: AMD A6-6310 APU with Radeon R4 Graphics 1.80GHz, 8G RAM..

According to the above experimental data and parameter configuration, the random function based on the MathLab tool generated the code for testing the AF-CSDS algorithm. Six different user requests were implemented in two server cluster environments with different numbers for testing to obtain more valid data to verify the performance of the AF-CSDS algorithm, as shown in Table 6. Repeat 10 runs for each user request, and each run iteration is 100 iterations. The experimental data are averaged based on each proposed QoS parameter after statistics, and the experimental data are recorded in detail for statistical analysis and comparison. In addition, to better compare the efficiency of existing deployment strategies, that is, the running time, each user request of different cluster sizes is run 40 times separately, which is based on the fact that the meta-heuristic strategy generally stops searching after approximately 40 iterations.

5.2 | Comparison of deployment strategies

Here, we evaluate the proposed AF-CSDS deployment strategy based on the above experimental data and the simulation running environment. In the experiment, it was compared with existing deployment strategies such as APSO-TSDS,[5] MSG-NSGA-III,[20] ACO-MCMS,[19] GA-NSGA-II,[17] and the Spread algorithm implemented in Docker Swarm.[8]

1. The APSO-TSDS strategy is a service deployment strategy based on an accelerated particle swarm optimization algorithm. This strategy optimizes the transmission cost between services, aggregation correlation between containers, and resource load balancing of the CDC to realize the deployment of service execution containers.

TABLE 3 Service stack in an application

TSID	*SFC*	$Type_i$	CPU_i	$Store_i$	$Treq_i$	Thr_i	$Cins_i$	*Adj_MS*
TS_1	SFC_a	1	36	3.1	6	3	2	$\{TS_2, TS_7\}$
TS_2	SFC_a, SFC_c	2	2.4	27.3	8	5	2	$\{TS_3, TS_7, TS_{10}, TS_{11}\}$
TS_3	SFC_a, SFC_b	3	10.4	8.2	20	4	5	$\{TS_4, TS_{11}\}$
TS_4	SFC_b	1	20.6	5.1	4	2	4	$\{TS_5\}$
TS_5	SFC_b	1	26.2	7.2	8	4	2	$\{TS_6, TS_8\}$
TS_6	SFC_b	2	4.1	30.6	6	2	3	$\{TS_9\}$
TS_7	SFC_a, SFC_b	2	7.1	32	5	1.3	4	{}
TS_8	SFC_b	1	30	12.5	6	2	3	{}
TS_9	SFC_b	2	6.8	26	4	2	2	{}
TS_{10}	SFC_c	3	6.2	5.2	25	15	2	$\{TS_{12}\}$
TS_{11}	SFC_a, SFC_b	1	22	6.5	4	2	2	$\{TS_3\}$
TS_{12}	SFC_c	3	14.6	11.3	18	6	3	$\{TS_{13}, TS_{14}\}$
TS_{13}	SFC_c	1	34.0	6.9	6	3	2	$\{TS_{17}\}$
TS_{14}	SFC_c	2	7.6	40.5	4	2	2	$\{TS_{15}\}$
TS_{15}	SFC_c	2	12.4	33.6	4	2.4	2	$\{TS_{16}\}$
TS_{16}	SFC_c	2	4.2	46	3	1.1	3	{}
TS_{17}	SFC_c	1	30.3	6.4	5	1.2	5	$\{TS_{18}\}$
TS_{18}	SFC_c	1	2.4	4.6	6	4	3	{}

TABLE 4 CDC parameter setting

Parameter	Description	Value
$\|CDC\|$	CDC Scale	[120, 240]
CPU_k	Server CPU Capacity	200 ~ 800
$Store_k$	Server Store Capacity	200 ~ 800
$Type_k$	Server Type	1, 2, 3
$fail_k$	Server Failure Rate	0.001 ~ 0.03

TABLE 5 Parameter configuration for AF-CSDS algorithm

Parameter	Description	Value
F	Artificial fish scale	80
max_iter	Maximum number of iterations	100
Step	Maximum step	0.5
δ	Crowdedness factor	0.3
Visual	Visual range	$\leq K$
try_num	Try number	10

TABLE 6 Two experimental clusters

Test type	M	User requests	Running time
Lab. A	120	{×1.0, ×2.0, ×3.0, ×4.0, ×5.0, ×6.0}	10
Lab. B	240	{×1.0, ×2.0, ×3.0, ×4.0, ×5.0, ×6.0}	10

2. The MSG-NSGA-III strategy is an NSGA-III algorithm based on reference points (i.e., NSGA-III), which is used for starting and deploying microservices. The algorithm considers the actual idle rate of microservices, idle rate of computing and storage resources, and load balancing of computing and storage resources.
3. The ACO-MCMS strategy uses the Ant Colony Optimization (ACO) algorithm to solve the deployment of microservices. It realizes the container deployment of microservices by optimizing the network transmission cost, maximum negative resource utilization of the cluster, and average failure rate of microservices.
4. The GA-NSGA-II strategy is a microservice deployment strategy based on a non-dominated sorting genetic algorithm (NSGA-II). The optimization objectives of this strategy are the threshold distance between containers, network distance between containers, execution reliability of microservices, and load balancing of computing resources.
5. The Spread algorithm is a classic deployment strategy implemented by Docker Swarm. This strategy mainly deploys microservices on physical nodes with the least number of containers and evenly distributes service containers on each physical node. The goal is to improve the resource utilization.

5.3 | Analysis and comparison

The size of the server cluster and the number of user requests in this experiment are: M=[120, 240], $sizeof(Ureq) = 6$. This section evaluates the Pareto solution set of the AF-CSDS deployment strategy according to the proposed optimization objectives and Equation (26). Here, set $\beta_i = 1/3$. The comparison parameters are the matching degree (MD) between the containers and server nodes, load balancing of clusters (CLB), service execution failure(SEF), and resource utilization of computing and storage clusters. In addition, the running time of each deployment strategy was also considered.

In the test result dataset, the result data in bold black font are superior to the other results. Because the running time of the spread algorithm is less than that of the meta-heuristic strategies (AF-CSDS, APSO-TSDS, MSG-NSGA-III, ACO-MCMS, and GA-NSGA-II), its running time is not listed.

5.3.1 | Experiment A

The average experimental results of Experiment A are listed in Table 7. This test implemented six different user requests in a cluster comprising $M = 120$ server nodes. The user request scope is $Ureq_i \in [1.0, 6.0]$ and $1 \leq i \leq 6$, that is, the user request IDs are from Ureq1 to Ureq6.

For the matching degree (MD) between the container and server node, the smaller the value, the better the compatibility between the container and server node. It can be seen from Table 7 that the AF-CSDS strategy performs best, and its MD value ranges from 0.0339 to 0.1962. This is because our strategy considers type matching between the containers and server nodes, as well as aggregation between containers. The Spread strategy exhibited the worst performance. Its MD value ranged from 0.1001 to 0.6505. The main reason is that the strategy considers that containers are evenly distributed among server nodes and did not consider the type-matching relationship between the containers and server nodes.

For the rate of the service execution failure (SEF), compared to APSO-TSDS, MSG-NSGA-III, ACO-MCMS, GA-NSGA-II, and Spread, AF-CSDS improved by 17.75%, 18.23%, 13.09%, 18.33%, and 18.15%, respectively. The ACO-MCMS strategy takes second place in performance because it considers the rate of the service execution failure on the server node, but did not consider the factor of the platform failure rate where the service is located. Similarly, for load balancing of clusters (CLB), the AF-CSDS strategy performs best, and its CLB value ranges are 0.0130∼0.0798. The ACO-MCMS performance takes second place, with a value range of 0.0131∼0.0887.

In terms of running time, a comparison of the whole running time in Experiment A is shown in Figure 2A.

As shown in Figure 2A, the AF-CSDS strategy was far superior to the APSO-TSDS, ACO-MCMS, and GA-NSGA-II strategies. However, the running time of the ACO-MCMS strategy was the longest at 610.30% higher than that of the AF-CSDS strategy. In addition, the running time of the APSO-TSDS strategy occurs when the user requests Ureq=×5.0 exceeds ACO-MCMS strategy.

The experimental results show that the proposed AF-CSDS deployment strategy achieves better feasible solutions for the target values of MD, SEF, and CLB and has the best performance compared with the other five deployment strategies. In addition, it is superior to other meta-heuristic strategies in terms of running time.

5.3.2 | Experiment B

To verify the effectiveness of the AF-CSDS deployment strategy further, we designed Experiment B, which implemented six different user requests in a larger server cluster (M=240). The results of Experiment B are presented in Table 8.

TABLE 7 Statistical data for Experiment A

Obj.	Strategies	Ureq1	Ureq2	Ureq3	Ureq4	Ureq5	Ureq6
MD	AF-CSDS	**0.1962**	**0.1013**	**0.0675**	**0.0509**	**0.0405**	**0.0339**
	APSO-TSDS	0.6657	0.2793	0.1843	0.1378	0.1108	0.0911
	MSG-NSGA-III	0.4857	0.3073	0.2182	0.1455	0.1181	0.0890
	ACO-MCMS	0.5685	0.3184	0.1894	0.1271	0.1085	0.0926
	GA-NSGA-II	0.5401	0.2856	0.1687	0.1415	0.1055	0.0905
	Spread	0.0605	0.3228	0.2165	0.1636	0.1257	0.1001
SEF	AF-CSDS	**3.4144**	**7.1105**	**10.7428**	**14.7622**	**18.6654**	**22.6684**
	APSO-TSDS	4.3531	8.5179	12.7287	16.9650	21.3938	25.2031
	MSG-NSGA-III	4.3205	8.5759	12.8555	17.0704	21.2381	25.2031
	ACO-MCMS	3.9076	8.1873	12.2122	16.6590	20.7824	25.1931
	GA-NSGA-II	4.3025	8.6494	12.8662	17.1328	21.1971	25.6106
	Spread	4.3333	8.6080	12.7881	17.0281	21.2965	25.5016
CLB	AF-CSDS	**0.0130**	**0.0260**	**0.0391**	**0.0530**	**0.0658**	**0.0798**
	APSO-TSDS	0.0179	0.0309	0.0462	0.0642	0.0789	0.0948
	MSG-NSGA-III	0.0144	0.0295	0.0481	0.0623	0.0786	0.0956
	ACO-MCMS	0.0131	0.0270	0.0425	0.0579	0.0733	0.0887
	GA-NSGA-II	0.0149	0.0303	0.0450	0.0618	0.0757	0.0948
	Spread	0.0157	0.0316	0.0475	0.0633	0.0791	0.0949
Time(s)	AF-CSDS	**2.3391**	**2.3855**	**3.2492**	**3.4341**	**4.2480**	**5.3101**
	APSO-TSDS	3.5218	6.4394	9.1172	12.0789	15.6691	18.7916
	MSG-NSGA-III	2.9574	3.5348	4.1683	4.7484	5.2269	5.8179
	ACO-MCMS	20.5724	20.8911	21.1701	21.4388	21.8104	22.0795
	GA-NSGA-II	4.0279	5.3678	6.7379	8.2532	9.7313	11.1732

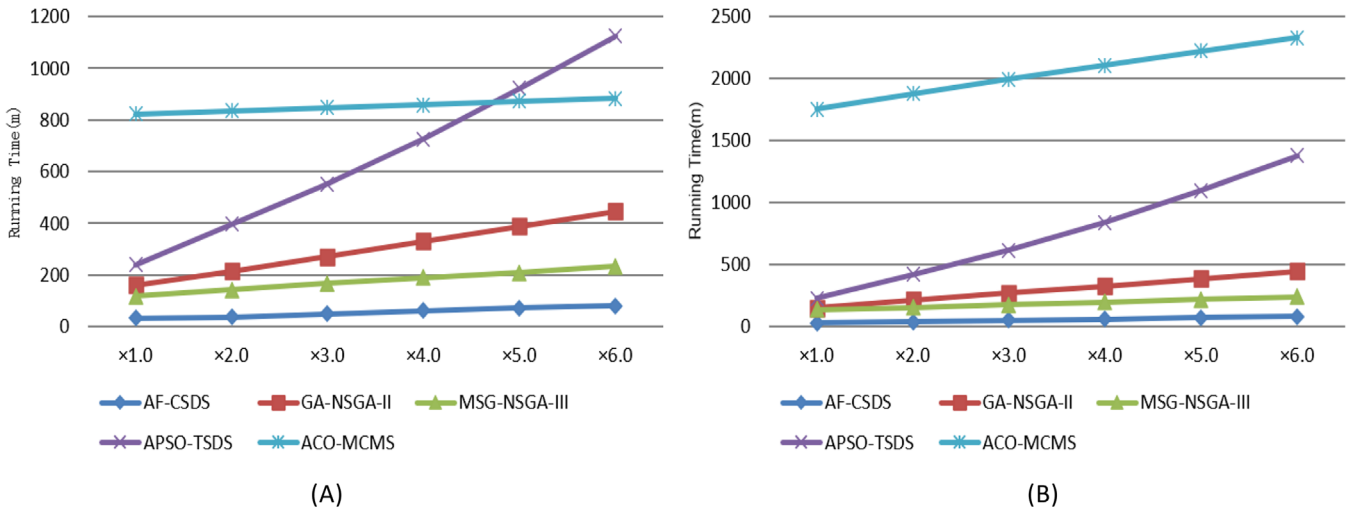

FIGURE 2 Comparison result of running time. (A) Experiment A. (B) Experiment B

TABLE 8 Statistical data for Experiment B

Obj.	Strategies	Ureq1	Ureq2	Ureq3	Ureq4	Ureq5	Ureq6
MD	AF-CSDS	**0.2029**	**0.1031**	**0.0705**	**0.0523**	**0.0415**	**0.0346**
	APSO-TSDS	0.5973	0.2965	0.1840	0.1404	0.1148	0.0903
	MSG-NSGA-III	0.5261	0.2833	0.2148	0.1422	0.1126	0.1059
	ACO-MCMS	0.6288	0.3095	0.1738	0.1399	0.1146	0.0908
	GA-NSGA-II	0.6735	0.2614	0.1822	0.1426	0.1114	0.0965
	Spread	0.6591	0.3302	0.2192	0.1669	0.1319	0.1090
SEF	AF-CSDS	**3.3280**	**6.9471**	**10.7634**	**14.7458**	**18.5056**	**22.7884**
	APSO-TSDS	4.3622	8.8215	13.1181	17.7279	21.8413	26.2933
	MSG-NSGA-III	4.4305	8.8626	13.2024	17.6698	22.0589	26.2987
	ACO-MCMS	4.1717	8.2909	12.8251	16.0234	20.4330	25.6380
	GA-NSGA-II	4.6090	8.6748	13.1124	17.6785	22.0045	26.2545
	Spread	4.4634	8.8533	13.1726	17.5634	21.9590	26.2877
CLB	AF-CSDS	**0.0061**	**0.0129**	**0.0194**	**0.0258**	**0.0318**	**0.0388**
	APSO-TSDS	0.0075	0.0152	0.0229	0.0302	0.0392	0.0467
	MSG-NSGA-III	0.0069	0.0141	0.0227	0.0305	0.0387	0.0465
	ACO-MCMS	0.0065	0.0135	0.0207	0.0281	0.0357	0.0431
	GA-NSGA-II	0.0076	0.0156	0.0234	0.0302	0.0372	0.0466
	Spread	0.0077	0.0155	0.0233	0.0311	0.0388	0.0465
Time(s)	AF-CSDS	**1.8965**	**2.5988**	**3.2301**	**3.7543**	**4.7097**	**5.9164**
	APSO-TSDS	3.6492	6.4238	6.9855	12.6047	15.9167	18.9035
	MSG-NSGA-III	3.3883	3.8648	4.3937	4.9426	5.4496	6.0332
	ACO-MCMS	43.7969	46.9315	49.8780	52.6925	55.6079	58.2428
	GA-NSGA-II	3.7885	5.3427	6.7133	8.1833	9.7049	11.2135

It can be seen from Table 8 that for MD, AF-CSDS still has the best performance, with an MD value range of 0.03461~0.2029, and The Spread has the worst performance, with a value range of 0.10901~0.6591.

For SEF, compared with APSO-TSDS, MSG-NSGA-III, ACO-MCMS, GA-NSGA-II and Spread strategies, AF-CSDS improved by 22.26%, 22.96%, 17.93%, 23.19%, and 22.84%, respectively. The performance of the ACO-MCMS was second. Similarly, for CLB, the AF-CSDS strategy has the best performance, and its CLB range is 0.0061~0.0388; the ACO-MCMS performance takes second place, and the value range is 0.0065~0.0431.

For the running time, a comparison of the whole running time of Experiment B is shown in Figure 2B.

As shown in Figure 2B, the AF-CSDS strategy is much better than the APSO-TSDS and ACO-MCMS strategies, while the MSG-NSGA-III strategy runs slightly longer than the AF-CSDS strategy. However, with the expansion of the server node scale, the whole running time of ACO-MCMS strategy is up to 4298.4 min (71.64 h). That is, this strategy spends a lot of time searching for appropriate server nodes to deploy containers, and has little application value in actual deployment. In addition, the running time of the APSO-TSDS strategy is significantly lower than that of the ACO-MCMS, but the running time is still very long (14.03 h).

From the comparison results in Table 8 and Figure 2B, the proposed AF-CSDS strategy is superior to the other five strategies in terms of the MD, SEF, CLB. Furthermore, it is superior to other meta-heuristic strategies in terms of running time.

5.3.3 | Resource utilization of computing and storage nodes

Here, we only observed the resource utilization of computing and storage server clusters to verify the effectiveness of the proposed AF-CSDS algorithm, while the resource utilization of common server clusters was not considered.

To analyze the performance of the proposed AF-CSDS strategy and the other five strategies on resource utilization, we collected sample data on the resource utilization of the computing and storage server clusters in 10 runs for Experiments A and B. The box graphs of the resource utilization distributions of Experiments A and B are shown in Figures 3 and 4, respectively.

In Figure 3, the resource utilization of the computing and storage clusters for the AF-CSDS strategy and spread strategy are almost all distributed at one point; that is, the data distribution is concentrated in the upper and lower quartiles, indicating that the resource utilization has little change, and the performance is stable. In Figure 3A, the data distribution of GA-NSGA-III strategy is relatively scattered, indicating that its performance is relatively unstable and changes greatly; GA-NSGA-II strategy has the largest data distribution change and the worst performance stability; Compared with GA-NSGA-III and GA-NSGA-II strategies, APSO-TSDS and ACO-MCMS strategy data are more centralized and stable. However, in Figure 3B, the data distribution of GA-NSGA-II is relatively scattered and unstable, and the data distribution of the GA-NSGA-III strategy is the worst.

In Figure 4, the resource utilization data of the AF-CSDS and spread strategies are still distributed in the upper and lower quartiles, indicating that their performance is stable. As shown In Figure 4A, when ureq= 1.0, the upper quartile data of the GA-NSGA-II strategy are almost at the same level as that of the AF-CSDS strategy, but the median is significantly lower than that of the AF-CSDS strategy, indicating that the resource rate of the GA-NSGA-II strategy is higher than that of the other strategies. However, GA-NSGA-II exhibits the worst performance stability as the number of requests increases, and the data distribution becomes more dispersed. Similarly, compared to the GA-NSGA-III and GA-NSGA-II strategies, the ACO-MCMS strategy data distribution is more centralized and stable. However, in Figure 4B, the data distribution of the GA-NSGA-III strategy is relatively stable, its resource utilization is also higher than that of other strategies, and the data distribution of GA-NSGA-II is still the worst.

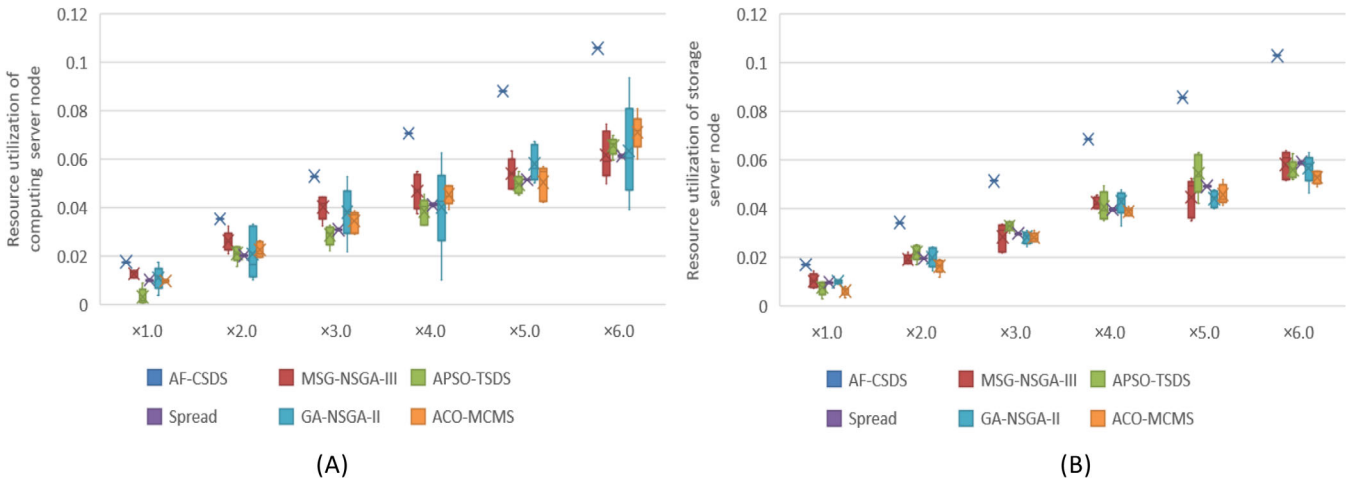

FIGURE 3 Comparison results of server cluster resource utilization in Experiment A. (A) Computing server cluster. (B) Storage server cluster

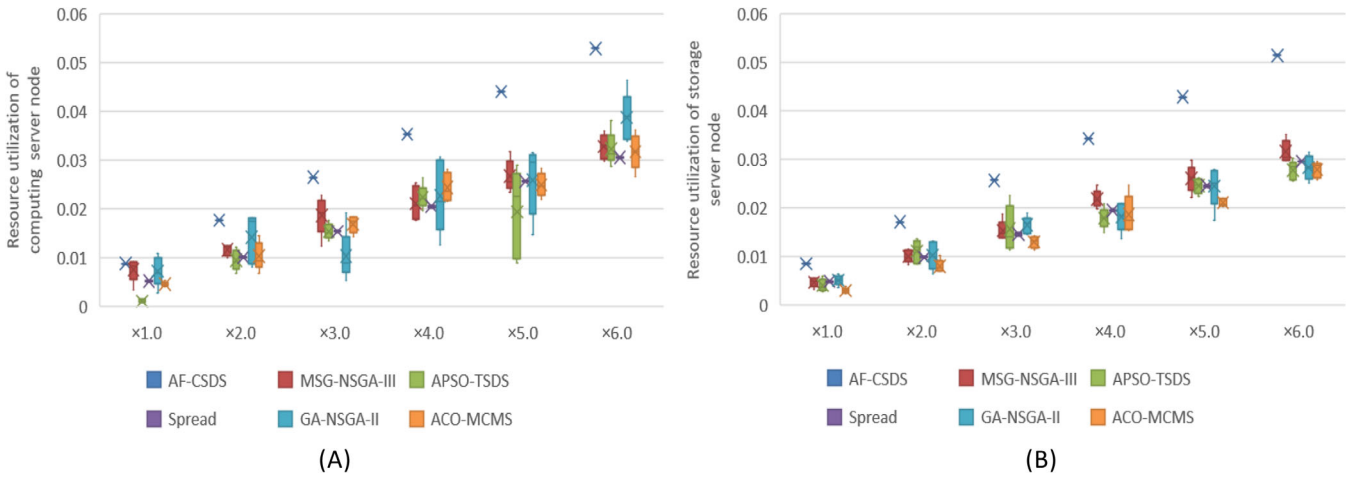

FIGURE 4 Comparison results of server cluster resource utilization in Experiment B. (A) Computing server cluster. (B) Storage server cluster

As shown in Figures 3 and 4, the resource utilization of the proposed AF-CSDS strategy is significantly higher than the median of the resource utilization of the other strategies, the data distribution is stable, and the performance is optimal.

To facilitate discussion and understanding, the statistics of the average resource utilization of the computing and storage server nodes of the six algorithms are presented in Table 9. In Experiment A, the resource utilization of the computing server node cluster: The AF-CSDS strategy is superior to APSO-TSDS, MSG-NSGA-III, ACO-MCMS, GA-NSGA-II and Spread strategies by 50.41%, 36.61%, 37.98%, 33.16% and 41.68%, respectively; The resource utilization of the storage cluster has increased by 44.32%, 40.37%, 49.85%, 42.98% and 42.54%, respectively.

In Experiment B, the resource utilization of the computing server cluster increased by 52.61%, 36.27%, 42.15%, 37.25%, 42.09%, respectively; The resource utilization of the storage server cluster has increased by 44.76%, 38.51%, 53.59%, 44.01% and 42.71%, respectively. The results of the two experiments show that AF-CSDS has the best resource utilization and the most stable performance under the load balancing constraint.

5.3.4 | Parameter analysis

In the process of solving practical multi-objective problems (MOPs), parameters have a significant impact on the performance of the algorithm, and their settings and choices often depend on the experience of the researchers. The visual range of an artificial fish is the key factor in determining the neighborhood and distance, and an appropriate visual range value can balance global and local searches. To verify the effectiveness and performance of the parameters of the AF-CSDS strategy, we tested the visual sensitivity of artificial fish in the test environment of Experiment B without losing generality.

As described in Section 4.2, the encoding length of each artificial fish is the position length $K = \sum_{j=1}^{N} Cins_j$, where N is the number of services. That is, the K value is related to the number of service execution container instances *Cins*. The Equation for calculating *Cins* is as follows:

TABLE 9 Statistical analysis of the average resources utilization

	Experiment A		Experiment B	
Strategies	**CPU utilization**	**Store utilization**	**CPU utilization**	**Store utilization**
AF-CSDS	61.80%	60.03%	30.91%	30.01%
APSO-TSDS	34.08%	34.14%	16.27%	17.05%
MSG-NSGA-III	37.29%	35.58%	18.56%	18.46%
ACO-MCMS	38.61%	30.72%	18.68%	14.62%
GA-NSGA-II	40.29%	33.77%	19.53%	16.70%
Spread	35.96%	34.44%	17.89%	17.21%

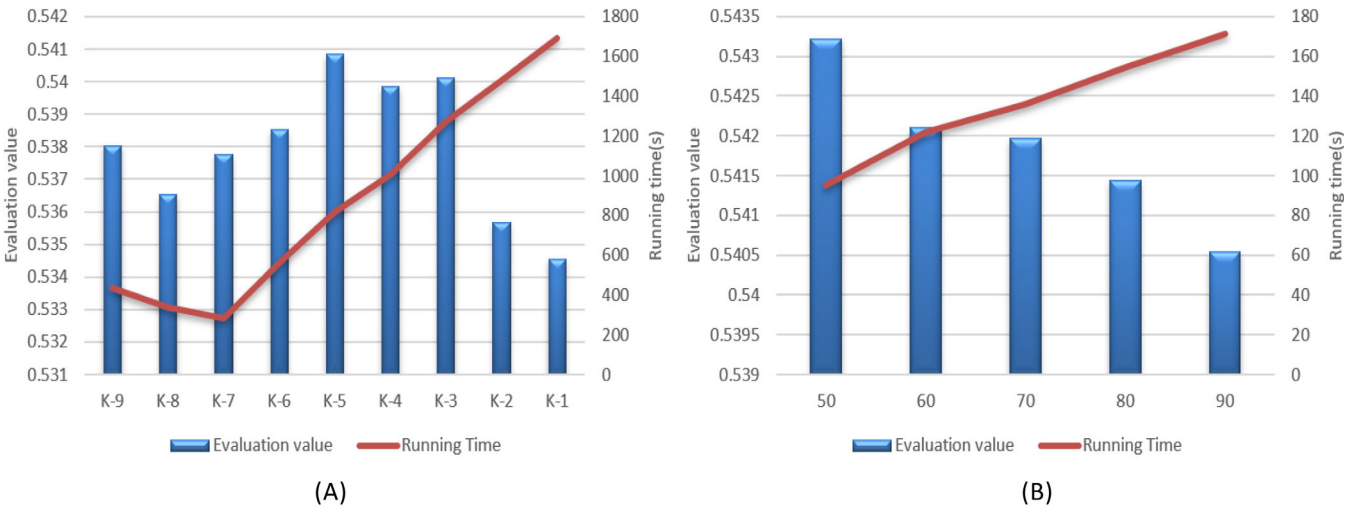

FIGURE 5 Effect of different (A) *visual* and (B) *F* values on Experiment B

$$Cins = \left\lceil \frac{Ureq \times Tsreq}{Thr} \right\rceil \tag{27}$$

To select an appropriate *Visual* value, we used K as the reference object and 1 as the step size to test the distribution value.

It can be observed from Figure 5A that with an increase in the *Visual* value, the running time of the AF-CSDS algorithm shows a downward trend, reaching the lowest point when $Visual = K - 7$. However, when $Visual > K - 7$, its running time rises sharply in a linear manner, indicating that parameter *Visual* has an important impact on the running time. The fundamental reason is that the search range of feasible solutions expands with an increase in the visual range factor. For the evaluation value, the evaluation value is the normalized calculated value of the three objective functions. When $Visual = K - 5$, the evaluation value reaches the peak value; When $Visual = K - 1$, the evaluation value is the minimum, which indicates that with an increase in the *Visual* range, the performance of the AF-CSDS strategy on the three objective functions improves, but the running time increases sharply. Weigh the runtime and performance, so we set *Visual* to $K - 8$.

Similarly, to test the effect of fish swarm size F, we set the value of parameter F from 50 to 90, and the experimental results are shown in Figure 5B. As can be seen in Figure 5B, with an increase in F, the evaluation value shows a downward trend. When $F = 90$, the AF-CSDS strategy has the best performance on the three objective functions; however, the runtime increases linearly. Considering the balance between performance and runtime, we set parameter F to 80.

6 | CONCLUSIONS

This study designs a deployment strategy for service execution containers to address computing- and storage-intensive microservice deployment problems. Based on the similarity between the microservices of different business types and servers, three objective functions are proposed: the matching degree (MD) between the containers and server nodes, load balancing of clusters (CLB), service execution failure(SEF). We used an artificial fish swarm algorithm and crowding distance, to obtain the solution of the problem model by optimizing three objective functions to deploy the execution containers of different types of microservices to the corresponding server nodes. Comprehensive experiments show that, compared with other deployment strategies, this strategy achieved good improvement rates in MD, CLB, and SFC. Under the constraint of load balancing, the resource utilization and service execution reliability of computing and storage server nodes are significantly improved. In addition, this strategy significantly shortens the running time by optimizing the performance parameters and provides a competitive objective function value.

Although the proposed AF-CSDS strategy shows great optimization potential in dealing with the deployment of microservices of different business types, it is only for computing- and storage-intensive services. Future work will extend to other business types to adapt to more types of microservice deployment problems. In addition, in a heterogeneous cluster environment, the optimization of the task scheduling performance of microservices is related to the interests of providers and cloud users. We will adopt deep learning to solve this problem.

ACKNOWLEDGMENTS

This work was supported in part by the Science and Technology Plan Project of Guangdong Province, China, under grants 2014B010112007 and 2016B010124010.

CONFLICT OF INTEREST STATEMENT

The authors declare no potential conflict of interests.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ORCID

Mingxue Ouyang https://orcid.org/0000-0002-1205-4730
Weihua Bai https://orcid.org/0000-0001-8333-7415

REFERENCES

1. Armbrust M, Fox A, Griffith R, et al. A view of cloud computing. *J Commun ACM*. 2010;53(4):50-58.
2. Sharma P, Chaufournier L, Shenoy P, Tay YC. Containers and virtual machines at scale: a comparative study. Proceedings of the 17th International Middleware Conference, ACM; 2016:1-13.
3. Kang H, Le M, Tao S. Container and microservice driven design for cloud infrastructure devops. Paper presented at: 2016 IEEE International Conference on Cloud Engineering (IC2E), IEEE; 2016:202-211.
4. Nadareishvili I, Mitra R, McLarty M, Amundsen M. *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, Inc; 2016.

5. Ouyang M, Xi J, Bai W, Li K. Band-area resource management platform and accelerated particle swarm optimization algorithm for container deployment in internet-of-things cloud. *IEEE Access*. 2022;10:86844-86863.
6. Zhang F, Liu G, Fu X, Yahyapour R. A survey on virtual machine migration: challenges, techniques, and open issues. *J IEEE Commun Surv Tutor*. 2018;20(2):1206-1243.
7. Pahl C, Brogi A, Soldani J, Jamshidi P. Cloud container technologies: a state-of-the-art review. *J IEEE Trans Cloud Comput*. 2017;7(3):677-692.
8. Rad BB, Bhatti HJ, Ahmadi M. An introduction to docker and analysis of its performance. *Int J Comput Sci Netw Secur*. 2017;17(3):228-234.
9. Ahmad I, AlFailakawi MG, AlMutawa A, Alsalman L. Container scheduling techniques: a survey and assessment. *J King Saud Univ-Comput Inf Sci*. 2021;34(7):3934-3947.
10. Bai W, Xi J, Zhu J. Performance analysis of heterogeneous data centers in cloud computing using a complex queuing model. *J Math Prob Eng*. 2015;2015(PT.12):1-15.
11. Docker Inc. *Docker Swarm Strategies*. Docker Inc; 2022. https://docs.docker.com/engine/swarm/
12. Mousa MH, Hussein MK. Efficient UAV-based mobile edge computing using differential evolution and ant colony optimization. *J Peer J Comput Sci*. 2022;8(e870):1-24.
13. Li XL, Qian JX. Studies on artificial fish swarm optimization algorithm based on decomposition and coordination techniques. *J Circuits Syst*. 2003;1(2003):1-6.
14. Li XL, Lu F, Tian GH, Qian JX. Applications of artificial fish school algorithm in combinatorial optimization problems. *J Shandong Univ (Eng Sci)*. 2004;34(5):64-67.
15. Lei D, Yan X. *Multi-Objective Intelligent Optimization Algorithm and Its Application*. Science Press; 2009:300-308.
16. Madni SHH, Latiff MSA, Coulibaly Y, Abdulhamid SIM. Recent advancements in resource allocation techniques for cloud computing environment: a systematic review. *J Clust Comput*. 2017;20(3):2489-2533.
17. Guerrero C, Lera I, Juiz C. Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. *J Grid Comput*. 2018;16(1):113-135.
18. Ullah A. Artificial bee colony algorithm used for load balancing in cloud computing. *IAES Int J Artif Intell*. 2019;8(2):156-167.
19. Lin M, Xi J, Bai W, Wu J. Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. *IEEE Access*. 2019;7:83088-83100.
20. Ma W, Wang R, Gu Y, et al. Multi-objective microservice deployment optimization via a knowledge-driven evolutionary algorithm. *J Complex Intell Syst*. 2021;7(3):1153-1171.
21. Muniswamy S, Vignesh R. DSTS: a hybrid optimal and deep learning for dynamic scalable task scheduling on container cloud environment. *J Cloud Comput*. 2022;11(1):1-19.
22. Zhang D, Yan BH, Feng Z, Zhang C, Wang YX. Container oriented job scheduling using linear programming model. Paper presented at: 2017 3rd International Conference on Information Management (ICIM), IEEE; 2017:174-180.
23. Liu B, Li P, Lin W, Shu N, Li Y, Chang V. A new container scheduling algorithm based on multi-objective optimization. *J Soft Comput*. 2018;22(23):7741-7752.
24. Liu B, Li J, Lin W, Bai W, Li P, Gao Q. K-PSO: an improved PSO-based container scheduling algorithm for big data applications. *J Int J Netw Manag*. 2021;31(2):e2092.
25. Xie Y, Wang Y, Jiang Y, Peng Z, Wang Y. Multi-objective task scheduling algorithm based on harmony search for grid microservice optimization. Paper presented at: 2020 3rd International Conference on Modeling, Simulation and Optimization Technologies and Applications (MSOTA), IOP Publishing; 2020.
26. Zhu L, Huang K, Hu Y, Tai X. A self-adapting task scheduling algorithm for container cloud using learning automata. *IEEE Access*. 2021;9:81236-81252.
27. Fan G, Chen L, Yu H, Qi W. Multi-objective optimization of container-based microservice scheduling in edge computing. *J Comput Sci Inf Syst*. 2021;18(1):23-42.
28. Chen X, Xiao S. Multi-objective and parallel particle swarm optimization algorithm for container-based microservice scheduling. *J Sensors*. 2021;21(18):6212-6240.
29. Liu R, Yang P, Lv H, Li W. Multi-objective multi-factorial evolutionary algorithm for container placement. *J IEEE Trans Cloud Comput*. 2021;2021:1-16.
30. Gupta D, Khanna A, Shankar K, Furtado V, Rodrigues JJ. Efficient artificial fish swarm based clustering approach on mobility aware energy-efficient for MANET. *J Trans Emerg Telecommun Technol*. 2019;30(9):e3524.
31. Feng Y, Zhao S, Liu H. Analysis of network coverage optimization based on feedback K-means clustering and artificial fish swarm algorithm. *IEEE Access*. 2020;8:42864-42876.
32. Ajitha KM, Indra NC. Bivariate correlative oppositional based artificial fish swarm resource optimized task scheduling in cloud. *J Int J Next-Gener Comput*. 2020;11(2):163-177.
33. Albert P, Nanjappan M. An efficient kernel FCM and artificial fish swarm optimization-based optimal resource allocation in cloud. *J Circuits Syst Comput*. 2020;29(16):1-16.
34. Li T, Yang F, Zhang D, Zhai L. Computation scheduling of multi-access edge networks based on the artificial fish swarm algorithm. *IEEE Access*. 2021;9:74674-74683.
35. Uma J, Vivekanandan P, Mahaveerakannan R. A heuristic algorithm for deadline-based resource allocation in cloud using modified fish swarm algorithm. *Proc. Inventive Computation and Information Technologies*. Springer; 2021:1-13.
36. Manikandan N, Divya P, Janani S. BWFSO: hybrid black-widow and fish swarm optimization algorithm for resource allocation and task scheduling in cloud computing. Proc. 1st International Conference on Innovative Technology for Sustainable Development (ICITSD); 2022:4903-4908. Materials Today: Proceedings.
37. Ouyang M, Xi J, Bai W, Li K. Band-area application container and artificial fish swarm algorithm for multi-objective optimization in internet-of-things cloud. *IEEE Access*. 2020;10:16408-16423.
38. Pourpanah F, Wang R, Lim CP, Wang XZ, Yazdani D. A review of artificial fish swarm algorithms: recent advances and applications. *J Artif Intell Rev*. 2022;1:1-37.

39. Marler RT, Arora JS. The weighted sum method for multi-objective optimization: new insights. *J Struct Multidiscip Optim*. 2010;41(6):853-862.
40. Alibaba Corp. Alibaba Cluster Trace V2018; 2021. https://github.com/alibaba/clusterdata

How to cite this article: Ouyang M, Xi J, Bai W, Li K. A container deployment strategy for server clusters with different resource types. *Concurrency Computat Pract Exper*. 2023;35(10):e7665. doi: 10.1002/cpe.7665