Received May 29, 2019, accepted June 15, 2019, date of publication June 24, 2019, date of current version July 11, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2924414

Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud

MIAO LIN[1], **JIANQING XI**[1], **WEIHUA BAI**[2], **AND JIAYIN WU**[3]

[1]School of Software Engineering, South China University of Technology, Guangzhou 510006, China
[2]School of Computer Science, Zhaoqing University, Zhaoqing 526061, China
[3]School of Computer, Guangdong Vocational College of Post and Telecom, Guangzhou 510630, China

Corresponding authors: Jianqing Xi (jianqingxi@163.com) and Weihua Bai (bandwerbai@gmail.com)

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

ABSTRACT In cloud architectures, the microservice model divides an application into a set of loosely coupled and collaborative fine-grained services. As a lightweight virtualization technology, the container supports the encapsulation and deployment of microservice applications. Despite a large number of solutions and implementations, there remain open issues that have not been completely addressed in the deployment and management of the microservice containers. An effective method for container resource scheduling not only satisfies the service requirements of users but also reduces the running overhead and ensures the performance of the cluster. In this paper, a multi-objective optimization model for the container-based microservice scheduling is established, and an ant colony algorithm is proposed to solve the scheduling problem. Our algorithm considers not only the utilization of computing and storage resources of the physical nodes but also the number of microservice requests and the failure rate of the physical nodes. Our algorithm uses the quality evaluation function of the feasible solutions to ensure the validity of pheromone updating and combines multi-objective heuristic information to improve the selection probability of the optimal path. By comparing with other related algorithms, the experimental results show that the proposed optimization algorithm achieves better results in the optimization of cluster service reliability, cluster load balancing, and network transmission overhead.

INDEX TERMS Ant colony algorithm, cloud computing, container scheduling, microservices, multi-objective optimization.

I. INTRODUCTION

In recent years, as a new application development model, microservices have attracted widespread attention and have been adopted in many occasions. Based on the microservice architecture, applications are designed as a set of independent, fine-grained modular services each executing a single business task, and among microservices there use lightweight communication mechanism. The requirements of the application are realized through a group of collaborative microservices. Microservice applications are easy to deploy and update, allowing for the independent updates and redeployments of some services without restarting, and have the characteristics of easy continuous delivery [2]. The advantages of microservices have prompted many large enterprises to migrate their applications to this development framework, including Netflix [3], Amazon [4], IBM [5], Uber [6], Alibaba [7], and so on.

Docker container [8], as a lightweight virtualization technology in the operating system layer, provides an separate execution environment and file system for the running of applications. Docker container image uses incremental and hierarchical file system, containing only additional files and dependency libraries which are not in the underlying operating system, thus Docker container instance reduces the cost of virtualization [8]. Compared with virtual machines, container has the advantages of lower resource consumption, easier and faster deployment, and higher portability. As an excellent tool for encapsulating, isolating and deploying microservices, container is one of the most important technologies to

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

support microservices. Currently, the mainstream container cluster management tools include Docker Swarm, Apache Mesos and Google Kubernetes [4].

With the development of container technology and the widespread use of microservice development model, some practical container scheduling solutions are proposed, but there are still some important problems to be solved in container-based microservice scheduling and management. Since entering the era of cloud computing, scheduling methods in cloud have been widely studied. However, more research on cloud scheduling mainly focuses on resource allocation of virtual machines to achieve performance-oriented load balancing or energy-oriented load integration [9]. Research on container scheduling based on cloud is still very limited. Current container cluster management tools include Docker Swarm, Apache Mesos and Google Kubernetes, which implement simple methods of allocating containers to the physical machines. For example, Docker Swarm scheduler has three strategies: Spread, Binpack and Random. Kubernetes scheduler is divided into two stages: Predicates stage and Priorities stage. These methods only focus on the use of physical resources [2], and do not achieve optimization strategies for system performance reliability, network overhead, etc. An effective container resource allocation method not only satisfies the load requirements of the cluster, but also ensures the performance and reliability of the cluster. There is room for improvement in performance of microservice applications, reliability of cluster, and network transmission overhead among microservices, and it requires further research.

Container scheduling is a typical NP-hard problem. At present, many researchers have adopted ant colony algorithm to solve the scheduling problem of virtual machines in cloud computing. Ant colony algorithm is a probabilistic and uncertain global optimization algorithm, which can easily get the global optimal solution. In addition, it is robust and does not rely on strict mathematical optimization and the structural characteristics of the problem itself. In this paper, a multi-objective container scheduling optimization model is proposed, which considers the constraints of physical machine resources and aims at reducing network transmission overhead, balancing cluster resource load and improving the reliability of cluster services. An ant colony algorithm is proposed to solve the multi-objective optimization problem of container scheduling.

The contributions of this paper are summarized as follows.

-Firstly, this paper establishes three container-based microservice scheduling problem models, including the network transmission overhead model of microservices, the cluster load model measured by the maximum value of resource utilization rate with coefficient among the physical nodes, and the service reliability model measured by the average number of failures for microservice requests.

-Secondly, a multi-objective optimization model for container scheduling is proposed. The model takes the resource capacity of the physical nodes as constraints to optimize the network transmission overhead, cluster resource load, and the average number of failures for microservice requests.

-Finally, an ant colony optimization algorithm is proposed to solve the multi-objective optimization problem of container scheduling. The algorithm combines multi-objective heuristic information to improve the selection probability of optimal path. The algorithm also evaluates and adjusts the quality of the solution in a timely manner by using three optimization objective functions. Thus the ant colony algorithm solves the problem that the original ant colony algorithm always falls into the local optimum.

The rest of this paper is divided into following sections. Section II introduces the related work. Section III formalizes the system model and optimization objectives. In Section IV, we present our solution and the implementation of our ant colony algorithm is also given. The experimental results and analysis are presented in Section V. The last section is the conclusion.

II. RELATED WORK

Resource management is very important in cloud computing, because the quality of resource management is not only related to the needs of users, but also directly affects the load and performance of the system. Resource management optimization is a hot research topic in the field of cloud computing. The related work showed in this paper are mainly divided into three aspects: container orchestration and resource management, multi-objective optimization method, and scheduling method based on ant colony optimization (ACO) algorithm.

Firstly, some related work on container orchestration and resource management are showed here. For example, Li *et al.* [10] proposed an optimal minimum migration algorithm (OMNM) which estimates the growth trend of each Docker container and determines which container need to be migrated by fitting the growth rate of Docker containers in the source server. The algorithm aims to reduce the unnecessary migration of containers, while ensuring the load balancing of the cluster. Menouer and Cérin [12] presented a scheduling and resource management allocation system based on an economic model related to different classes for SLAs (Service Level Agreements), aiming to help companies manage a private infrastructure of machines, and to optimize the scheduling of several requests submitted online by users. Zhang *et al.* [13] proposed a container placement strategy by simultaneously taking into account the three involved entities including container, VM and PM, aiming to improve the physical resource utilization. Adam *et al.* [14] presented two-stage Stochastic Programming Resource Allocator (2SPRA), aiming to optimize resource provisioning for containerized n-tier web services in accordance with fluctuations of incoming workload to accommodate predefined SLOs on response latency. Guan *et al.* [16] designed a novel application oriented Docker container (AODC)-based resource allocation framework to minimize the application deployment cost in DCs, and to support automatic scaling while the workload of cloud applications varies. They modeled the AODC resource allocation problem considering features of Docker, various applications' requirements, and available resources in cloud data centers, and proposed a scalable algorithm for DCs with diverse and dynamic applications and massive physical resources. Although the papers above considered a part of factors for container resource scheduling, there are many other factors, for example, nature of cluster nodes including failure rate.

Secondly, in cloud resource allocation, the solutions may involve many conflicting and influential objectives, and researchers need to get these objectives as optimal as possible simultaneously. Therefore, multi-objective optimization methods are widely adopted in this field. Zhang *et al.* [11] modeled the scheduling problem as an integer linear programming, and proposed an adaptive scheduler in which the container host energy conservation, the container image pulling costs from the image registry to the container hosts, and the workload network transition costs from the clients to the container hosts are evaluated in combination. Kaur *et al.* [15] proposed a multi-objective function by using cooperative game theory, in order to reduce the energy consumption and makespan by considering different constraints such as memory, CPU, and the user's budget. Duan *et al.* [17] formulated the scheduling problem as a sequential cooperative game and proposed a communication and storage-aware multi-objective algorithm that optimizes two user objectives (execution time and economic cost) while fulfilling two constraints (network bandwidth and storage requirements). Panda and Jana [18] proposed a multi-objective task scheduling algorithm for heterogeneous multi-cloud environment which takes care both minimization of the overall completion time and that of the overall cost of the service. Zuo *et al.* [19] proposed a self-adaptive learning particle swarm optimization (SLPSO)-based multi-objective optimization method to maximize the profit of the IAAS providers, taking into account the running time, deadline and resource utilization. Aiming at the performance optimization of Docker container resource scheduling, Liu *et al.* [20] proposed a multi-objective container scheduling algorithm, namely Multiopt, considering five key factors: CPU usage of every node and memory usage of every node to balance load, the time consumption transmitting images on the network to reduce network overhead, the association between containers and nodes to improve performance of services, the clustering of containers to reduce response time. For each key factor, the authors defined a metric method and established a scoring function of objective to select the most suitable node to deploy containers. Shi *et al.* [21] considered container consolidation as one multi-objective optimization problem and presented an algorithm based on NSGA-II, with the objectives of minimizing the total energy consumption and minimizing the total number of container migrations within the certain period of time. Guerrero *et al.* [22] presented a approach based on NSGA-II to optimize the deployment of microservices-based applications using containers in multi-cloud architectures, considering three optimization objectives: cloud service cost, network latency among microservices, and time to restart a new microservice. The same authors [23] proposed a genetic algorithm approach based on NSGA-II to optimize container allocation and elasticity management. The algorithm considers four optimization objectives, including threshold distance of the container workload, the balanced use of computing resources, the reliability of the container applications, and the intercommunication overhead among related microservices. The work of paper [23] is the most similar one to our approach but with significant differences. Guerrero did not considered data transmission among microservices, neither the relation between the number of microservice requests and the reliability of the cluster services.

Thirdly, the ACO algorithm has the characteristics of positive feedback of information and heuristic search. It is essentially a heuristic global optimization algorithm in evolutionary algorithm, which is widely used in the field of cloud computing. For example, Zuo *et al.* [24] proposed an improved ACO algorithm for task scheduling, considering the makespan and the user's budget costs as the optimization objectives. Two objective functions were used to evaluate solutions, and then the quality of the solution were adjust in a timely manner based on feedback regarding the evaluation. Guo [25] proposed a task scheduling algorithm based on ACO algorithm, aiming to minimize the makespan and the total cost of the tasks, while making the system load more balanced. The work improved the initialization of the pheromone, the heuristic function and the pheromone update method in the ant colony algorithm. Zhang *et al.* [26] improved the Kubernetes scheduling model by combining ant colony algorithm and particle swarm optimization algorithm. Compared with the original model, the algorithm additionally takes into account the use of resource costs. Kaewkasi and Chuenmuneewong [27] presented an ACO-based algorithm to balance the resource usage and led to the better performance of applications. Jiao *et al.* [28] proposed a resource scheduling strategy based on the improved ACO algorithm. Based on cluster service and user quality of service preference, the strategy constructed an ant optimization model to design the parameterization definition and select the preference constraints. The fitness function of multi-dimensional quality of service was given and then the local or global update was performed accordingly. The algorithm implemented the search for Pareto optimal set of multi-objective problems. Chen *et al.* [29] proposed a resource scheduling method of cloud computing based on QoS combining ACO algorithm and Shuffled Frog Leading Algorithm (SFLA). Firstly, ACO algorithm used the quality function and convergence factor to ensure the efficiency of pheromone updating and the feedback factor was set to improve the selection of probability. Secondly, the local search efficiency of SFLA was improved by setting crossover factor and mutation factor in the SFLA. Finally, the local search and global search of the SFLA were introduced for updating in each iteration of ACO algorithm. However, except the paper [26] and [27],

TABLE 1. Summary of parameters and their descriptions.

Element	Parameter	Description
Application	MS_SET	microservice set of the application
	$\lvert MS\_SET \rvert = m$	total number of microservices in the application
	$MS\_RELATION$	a set of consumption relationships among the microservices
Microservice	$ms_i \in MS\_SET$	microservice with id. i
	$Calc\_Reqst_i$	computational resources required by a microservice request
	Str_Reqst_i	storage resources required by a microservice request
	$CONS\_SET_i$	a set of microservices consumed by a microservice ms_i
	$Link\_Thr_i$	threshold for number of requests for a microservice ms_i
	$Link_i$	total number of requests for microservice ms_i
	$Scale_i$	number of containers for a microservice ms_i in the cluster
	$(ms_i, ms_j) \in MS\_RELATION$	microservice ms_i consumes microservice ms_j
	$Link(ms_i, ms_j)$	total number of requests from microservice ms_i to microservice ms_j
	$Trans(ms_i, ms_j)$	amount of data transmitted in a request between consumer ms_i and provider ms_j
Physical node	$pm_j \in CLUSTER$	physical node with id. j
	$\lvert CLUSTER \rvert = n$	total number of the physical nodes in the cluster
	$Calc\_Resrc_j$	computational capacity of a physical node pm_j
	Str_Resrc_j	storage capacity of a physical node pm_j
	$Fail_j$	failure rate of a physical node pm_j
	$alloc(pm_j)$	a set of microservices allocated in a physical node pm_j
Network	$Dist(pm_j, pm_{j'})$	network distance between pm_j and $pm_{j'}$

the paper [24], [25], [28], and [29] were not aiming at container scheduling. Although all of these work involved the ACO algorithm, none of them had considered multi-objective heuristic information.

Different from the previous work, we consider not only the computing and storage resources of the cluster, but also the number of microservice requests and the failure rate of the physical nodes. We propose a multi-objective optimization model, with which we implement an ant colony algorithm to solve the microservice container scheduling problem. Our algorithm adopts the evaluation function of the solution to ensure the validity of pheromone updating, combining multi-objective heuristic information to improve the selection probability of the optimal path. This paper aims to solve the optimization problems of network data transmission among microservice containers, load balancing of the cluster and reliability of cluster services in cloud simultaneously.

III. PROBLEM STATEMENT

This section describes the problem model and the optimization objective models. The parameters of the models and their descriptions are summarized in Table 1.

A. SYSTEM MODEL

We consider an application *APP* based on the microservice architecture. *APP* is characterized as a tuple <*MS_SET*, *MS_RELATION*>, where *MS_SET* is the microservice set of application *APP*, and the cardinality is m; *MS_RELATION* is the set of consumption relationships among the microservices. When a microservice consumes results generated by other microservice, a consumption relationship between the two microservices is established, denoted by $(ms_{cons}, ms_{prov}) \in MS\_RELATION$. The former microservice is the consumer and the latter one is the provider.

Microservice ms_i in *MS_SET* is characterized as a tuple $<Calc\_Reqst_i, Str\_Reqst_i, CONS\_SET_i, Link\_Thr_i>$, where $Calc\_Reqst_i$ represents the computing resources consumed to satisfy a request for the microservice ms_i; Str_Reqst_i is the storage resources required to satisfy a request for the microservice ms_i; $CONS\_SET_i$ is the set of the microservices consumed by the microservice ms_i, and when the consumption relationship satisfies $(ms_i, ms_l) \in MS\_RELATION$, there exists a microservice $ms_l \in CONS\_SET_i$; and $Link\_Thr_i$ represents the threshold for number of requests for the microservice ms_i. The values of $Calc\_Reqst_i$, Str_Reqst_i, and $Link\_Thr_i$ depend on the implementation of the microservice ms_i.

We consider the application *APP* that receives service requests from users. The number of requests from the microservice ms_i to the microservice ms_l is denoted by $Link(ms_i, ms_l)$, so the total number of requests for the microservice ms_l is calculated as $Link_l = \sum_{i=1 \wedge ms_l \in CONS\_SET_i}^{m} Link(ms_i, ms_l)$. $Trans(ms_{cons}, ms_{prov})$ represents the amount of data need to be transmitted in a request between the consumer ms_{cons} and the provider ms_{prov}. When the consumers of the microservices are clients, only the number of microservice requests is considered, regardless of network transmission costs. Each microservice is encapsulated in a container, and a microservice in the cluster can have multiple container instances. $Scale_l$ is the number of containers for a microservice ms_l in the cluster. So according to the threshold for number of requests for the microservice ms_l, the number of containers for the microservice ms_l in the cluster should be calculated as $Scale_l = \left\lceil \frac{Link_l}{Link\_Thr_l} \right\rceil$. The values of $Link_l$, $Scale_l$, and $Link(ms_i, ms_l)$ depend on the number of user requests.

Cluster *CLUSTER* has n physical nodes. Each physical node is characterized as a tuple $<Calc\_Resrc_j, Str\_Resrc_j, Fail_j>$, where $Calc\_Resrc_j$ indicates the computational capacity of the physical node pm_j; Str_Resrc_j is the storage capacity of the physical node pm_j; and $Fail_j$ is the failure rate of the physical node pm_j. The physical node "failure" mentioned in this paper refers to any exception caused by hardware or software defects, incorrect design, unstable environment, or operator errors that render service or computing nodes unavailable [31]. The physical nodes are interconnected with network, and the network path between nodes pm_j and $pm_{j'}$ is characterized by their network distance $Dist(pm_j, pm_{j'})$.

Figure 1 shows a simple application example in the cluster from Alibaba Cluster Trace V2018 cluster data set [30]. The application includes five interoperable microservices,

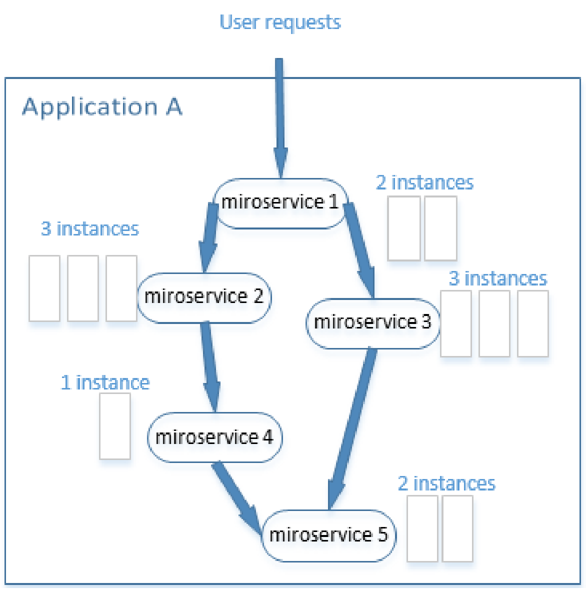


FIGURE 1. A simple application example in the cluster.

and each microservice has different number of container instances according to the requests from users or consumer microservices.

B. OBJECTIVE MODEL

The purpose of this paper is to achieve three improvements by studying the resource allocation method of microservice containers, including reducing the network transmission overhead among microservices, balancing the load of the cluster, and improving the reliability of cluster services. In order to achieve these three objectives, we establish the objective models as follows.

1) NETWORK TRANSMISSION OVERHEAD AMONG MICROSERVICES

The network transmission overhead among microservices is related to three key factors: the network distance between the physical nodes where the two interoperable microservice containers are allocated, the number of requests between the two microservices, and the amount of data transmission. Considering that both the consumer microservice and the provider microservice may have multiple container instances running at the same time, this paper uses the average network distance of all the container pairs between consumer microservice and provider microservice to calculate the data transmission overhead between two microservice containers. This is formalized in (1).

$$COMM(X) = \sum_{j=1}^{n}\sum_{i=1}^{m}\frac{x_{i,j}}{Scale_i}\sum_{l=1\wedge l\neq j}^{n}\sum_{ms_k\in CON\_SET_i} \frac{x_{k,l}}{Scale_k}Link(ms_i, ms_k)Trans(ms_i, ms_k)Dist(pm_j, pm_l) \tag{1}$$

2) MAXIMUM VALUE OF RESOURCE UTILIZATION RATE WITH COEFFICIENT AMONG THE PHYSICAL NODES

The scheduling model in this paper involves computing resources and storage resources, so load balancing of the cluster is a Multi-Resource Load Balancing (MRLB) problem. In paper [32], Wang *et al.* tackled the MRLB problem in Network Function Virtualization (NFV) by first proposing dominant load-the load of the most stressed resource on a server-as the load balancing metric, and then formulating the MRLB problem as an optimization problem to minimize the maximum dominant load of all NFV servers given the demand. Based on the paper [32], we present a improvement, adding load distribution of each resource in the cluster. For each resource in the cluster, we calculate the standard deviation of the utilization rate of the resource in the nodes and use its proportion as coefficient value for the utilization rate of corresponding resource in each node. Similar to paper [32], the maximum value of resource utilization rate with coefficient among the physical nodes reflects the worst-case load about load balancing of the cluster. The maximum value of resource utilization rate with coefficient among the physical nodes is formalized in (2).

$$RESRC\_CONS(X) = \frac{1}{\sigma_1+\sigma_2}\max_{1\leq j\leq n}\max\left(\sum_{i=1}^{m}x_{i,j}\frac{Link_i\times Cal\_Reqst_i}{Scale_i\times Cal\_Resrc_j}\sigma_1, \sum_{i=1}^{m}x_{i,j}\frac{Link_i\times Str\_Reqst_i}{Scale_i\times Str\_Resrc_j}\sigma_2\right) \tag{2}$$

Here σ_1, σ_2 are the standard deviation values of utilization rate of computing resources and storage resources of the physical nodes in the cluster respectively. Equation (2) indicates that the worst-load of the cluster is not necessarily the case of maximum resource utilization rate with relatively balanced resource load, but the case of high resource utilization rate with relatively unbalanced resource load.

3) AVERAGE NUMBER OF FAILURES FOR MICROSERVICE REQUESTS

Considering that each physical node in the cluster has a certain probability of failure due to various reasons. Microservice containers are distributed among these nodes, and there is the possibility of request failure. In this paper, the average number of request failures is indicator to measure the reliability of cluster services, which is mainly related to the number of microservice requests and the failure rate of the nodes. This is modeled in (3).

$$LINK\_FAIL(X) = \sum_{j=1}^{n}\sum_{i=1}^{m}Fail_j\times x_{i,j}\frac{Link_i}{Scale_i} \tag{3}$$

In the three equations above, $x_{i,j}$ is an decision variable, subjecting to $x_{i,j} = \begin{cases}1, & if \quad ms_i\in alloc(pm_j)\\ 0, & if \quad ms_i\notin alloc(pm_j)\end{cases}, \forall ms_i \forall pm_j.$

$alloc(pm_j)$ represents the set of microservices allocated in a physical node pm_j. $x_{i,j} = 1$ indicates that the microservices ms_i is allocated to the physical node pm_j.

C. MULTI-OBJECTIVE OPTIMIZATION MODEL

According to the three objective models mentioned above, a multi-objective optimization model for container scheduling is established in order to optimize the three objective simultaneously under the constraints of resource capacity and microservice deployment requirements.

$$\text{minimize} \quad COMM(X) \tag{4}$$

$$\text{minimize} \quad RESRC\_CONS(X) \tag{5}$$

$$\text{minimize} \quad LINK\_FAIL(X) \tag{6}$$

$$s.t. \quad \sum_{i=1}^{m} x_{i,j} \frac{Link_i}{Scale_i} Cal\_Rest_i \le Cal\_Resrc_j, \quad \forall pm_j \tag{7}$$

$$\sum_{i=1}^{m} x_{i,j} \frac{Link_i}{Scale_i} Str\_Rest_i \le Str\_Resrc_j, \quad \forall pm_j \tag{8}$$

$$x_{i,j} = \begin{cases} 1, & if \quad ms_i \in alloc(pm_j) \\ 0, & if \quad ms_i \notin alloc(pm_j) \end{cases}, \quad \forall ms_i \forall pm_j. \tag{9}$$

$$\sum_{j=1}^{n} x_{i,j} = Scale_i, \quad \forall ms_i \tag{10}$$

$$\sum_{i=1}^{m} x_{i,j} = 1, \quad \forall pm_j \tag{11}$$

Equation (4)-(6) represent the optimization of the three objectives respectively: minimizing the network transmission overhead among microservices, minimizing the maximum value of resource utilization rate with coefficient among the physical nodes, and minimizing the average number of failures for microservice requests.

Equation (7)-(11) are the constraint conditions of the optimization model. Equation (7) and (8) are the computing and storage resource constraints of physical nodes, respectively. Equation (9)-(11) are decision variable constraints. Equation (10) indicates that the allocation number of microservices must satisfies the number of their container instances; Equation (11) indicates that the container instances of the same microservice at each node are exclusive, and this constraint guarantees that in any physical node there can be at most one container instance from the same microservice, thus avoiding resource conflicts among the container instances of the same microservice.

It is difficult to solve the multi-objective optimization problem directly, especially to find the optimal solution. Ant colony algorithm has been widely used in various scheduling problems and achieved good results. It has certain advantages to solve such combinatorial optimization problems. Therefore, this paper proposes an ant colony optimization algorithm, which uses an evaluation function to estimate the quality of the solution, and feedback the quality ratio between the solution and the current optimal solution in each iteration to the next iteration through pheromone to improve the quality of the solutions to be generated, Combing multi-objective heuristic information and the feedback mechanism, the algorithm can improve the speed of finding the optimal solution and avoid falling into the local optimal solution.

IV. PROPOSED ANT COLONY OPTIMIZATION ALGORITHM

A. ALGORITHM OVERVIEW

The ant colony optimization algorithm simulates the feeding process of the ants to complete the scheduling of microservices. The algorithm is summarized as follows.

(1)Ant Ant_k is randomly placed to a microservice ms_i.

(2)Ant_k selects a path $path_{i,j}$ with a certain probability $p_{i,j}^k$ to reach the physical node pm_j, which is one of the nodes satisfying the constraint conditions of the model. $Scale_i$ is the number of containers for the microservice ms_i in the cluster, indicating that the microservice ms_i needs to be allocated $Scale_i$ times, and the selected nodes must be different every time. In other words, only when Ant_k completes the path selection from all the containers of microservice ms_i to different physical nodes, the job of container allocation for the microservice ms_i is truely completed. Then microservice ms_i can be put into tabu list $Tabu_k$ of Ant_k.

(3) Ant_k returns to the next microservice, and makes the next allocation, repeating the process of step (2).

(4) That all the ants complete the container allocation of all the microservices once, can be regarded as one iteration. The algorithm terminates until the maximum number of iterations is reached.

B. HEURISTIC INFORMATION

$\eta_{i,j}^k$ is the heuristic information which represents the expectation of Ant_k to allocate the container of the microservice ms_i to the physical node pm_j, and it is also the expectation of Ant_k to select path $path_{i,j}$. Heuristic information $\eta_{i,j}^k$ is defined as follows, according to the three objective models proposed above.

The first optimization objective is to minimize the network transmission overhead among microservices, so the heuristic information about the network transmission overhead can be formulated as:

$$\eta_{i,j}^{k(1)} = (\xi + \eta_{i,j}^{k(1)(1)} + \eta_{i,j}^{k(1)(2)})^{-1}.$$

Here ξ is a very small positive number.

$$\eta_{i,j}^{k(1)(1)} = \sum_{q=1 \wedge q \neq j}^{n} \sum_{ms_l \in alloc(pm_q)} \frac{Dist(pm_q, pm_j)}{Scale_l \times Scale_i} \times [Link(ms_l, ms_i) Trans((ms_l, ms_i) + Link(ms_i, ms_l) Trans((ms_i, ms_l)]$$

$\eta_{i,j}^{k(1)(1)}$ is the network transmission overhead between the microservice ms_i allocated to the node pm_j and the microservices already allocated to the other nodes. If the microservice ms_i does not consume the microservice ms_l, then $Link(ms_i, ms_l) = 0$ and $Trans(ms_i, ms_l) = 0$.

$$\eta_{i,j}^{k(1)(2)} = \sum_{ms_l \notin Tabu_k} \frac{Mean\_Dist_j}{Scale_i} \times [Link(ms_l, ms_i)Trans((ms_l, ms_i) + Link(ms_i, ms_l)Trans((ms_i, ms_l)]$$

$\eta_{i,j}^{k(1)(2)}$ is the average network transmission overhead between the microservice ms_i allocated to the node pm_j and the microservices not yet allocated to the nodes; $Mean\_Dist_j$ is the average network distance between the physical node pm_j and all the other nodes.

The second optimization objective is to balance the load of the cluster. The heuristic information about the second optimization objective can be formulated as:

$$\eta_{i,j}^{k(2)} = \left[\max\left(\frac{\sum_{ms_l \in alloc(pm_j)} \frac{Link_l}{Scale_l} Cal\_Reqst_l + \frac{Link_i}{Scale_i} Cal\_Reqst_i}{Cal\_Resrc_j}, \frac{\sum_{ms_l \in alloc(pm_j)} \frac{Link_l}{Scale_l} Str\_Reqst_l + \frac{Link_i}{Scale_i} Str\_Reqst_i}{Str\_Resrc_j}\right)\right]^{-1}.$$

The third optimization objective is to minimize the average number of failures for microservice requests. The heuristic information about the average number of failures for microservice requests can be formulated as:

$$\eta_{i,j}^{k(3)} = (Fail_j \times \frac{Link_i}{Scale_i})^{-1}.$$

By synthesizing the above heuristic information of the three objectives, the expectation of Ant_k to allocate the container of the microservice ms_i to the physical node pm_j can be formulated as:

$$\eta_{i,j}^{k} = \eta_{i,j}^{k(1)} \times \eta_{i,j}^{k(2)} \times \eta_{i,j}^{k(3)}. \tag{12}$$

C. THE EVALUATION FUNCTION OF SOLUTION

Once Ant_k has traversed all the microservices, the paths it has traveled generates a feasible solution to the multi-objective optimization problem. In order to ensure the quality of the solution and achieve the optimal solution as far as possible, it is necessary to evaluate the feasible solution. According to the optimization problem model established in this paper, the evaluation function of the solution can be formulated as (13).

$$Eval(X) = \frac{1}{3}\frac{COMM(X) - min_{COMM}}{max_{COMM} - min_{COMM}} + \frac{1}{3}RESRC\_CONS(X) + \frac{1}{3}\frac{LINK\_FAIL(X) - min_{LINK\_FAIL}}{max_{LINK\_FAIL} - min_{LINK\_FAIL}} \tag{13}$$

Equation (13) normalizes the tow objective functions with different dimension. max_{COMM}, min_{COMM} are the maximum and minimum values of the first optimization objective function respectively; max_{LINK_FAIL}, min_{LINK_FAIL} are the maximum and minimum values of the third optimization objective function respectively. These values can be obtained by Greedy Algorithm. As a feasible solution of the problem model, the smaller the value of $Eval(X)$ is, the closer X is to the optimum.

D. PHEROMONE UPDATING

In reality, ants leave behind pheromone every time they go along a path. The more ants go along a path per unit time, the more pheromone accumulates along the path. At the same time, due to the volatility of pheromone, pheromone on the path attenuates with time. The ACO algorithm simulates the real feeding process of ant colony. After each iteration of the algorithm, the pheromone on each path needs to be updated. Pheromone is updated as (14).

$$\tau_{i,j}(t+1) = (1-\rho)\tau_{i,j}(t) + \Delta\tau_{i,j} \tag{14}$$

Here $\tau_{i,j}(t)$ is the pheromone on path $path_{i,j}$ at time t; $\tau_{i,j}(t+1)$ is the updated pheromone. The initial value of the pheromone can be set to $\tau_{i,j}(1) = 1$. To avoid the unrestricted accumulation of the pheromone, the volatile factor of pheromone is denoted by ρ satisfying $\rho \in (0, 1)$. $(1-\rho)$ is the volatility of pheromone.

$\Delta\tau_{i,j}$ is the increment of the pheromone on path $path_{i,j}$ after one iteration. It is calculated as (15).

$$\Delta\tau_{i,j} = \sum_{k=1}^{K} \Delta\tau_{i,j}^{k} \tag{15}$$

Here K is the size of the ant colony; $\Delta\tau_{i,j}^{k}$ is the pheromone left behind by Ant_k on the path $path_{i,j}$. It is related to the evaluation function of the solution. It is calculated as (16).

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{Q}{Eval(X^k)}, & if \quad x_{i,j}^{k} = 1 \\ 0, & if \quad x_{i,j}^{k} = 0 \end{cases} \tag{16}$$

Here Q is the evaluation value of the current optimal solution; X^k is the current solution generated by the complete path traveled by Ant_k. If Ant_k went along the path $path_{i,j}$, it will contribute to the increment of the pheromone on the path. The above equation shows that the better solution has lower evaluation value, and more pheromone on the corresponding path.

E. TRANSITION PROBABILITY

Ant_k tends to select the next path with more pheromone and higher expectation from the current path, so the transition probability is related to pheromone and expectation. $RESRC_k$ represents a set of the available nodes for Ant_k satisfying the constraint conditions. The transition probability for Ant_k to select the path $path_{i,j}$ can be calculated as (17).

$$p_{i,j}^{k}(t) = \begin{cases} \frac{[\tau_{i,j}(t)]^{\alpha}[\eta_{i,j}^{k}]^{\beta}}{\sum_{r \in RESRC_k}([\tau_{i,r}(t)]^{\alpha}[\eta_{i,r}^{k}]^{\beta})}, & if \quad j \in RESRC_k \\ 0, & otherwisw \end{cases} \tag{17}$$

Here α, β are regulators for pheromone and heuristic information respectively.

F. ALGORITHM IMPLEMENTATION

In this paper, we propose an Ant Colony Optimization algorithm for Multi-objective of Container-based Microservice Scheduling (ACO_MCMS for short). The implementation of the ant colony optimization algorithm is shown as the pseudo-code of Algorithm 1.

G. COMPLEXITY ANALYSIS

The ACO_MCMS algorithm involves three optimization objectives and some constraint conditions. The complexity is mainly reflected in optimization of the network transmission overhead of microservice scheduling, which is a Quadratic Assignment Problem. An ant constructs a complete scheduling sequence, requiring $O(n^2 \times \sum_{i=1}^{m} Scale_i)$ operations. K ants require $O(K \times n^2 \times \sum_{i=1}^{m} Scale_i)$ operations in one iteration. Updating pheromone matrix after one iteration requires $O(n \times \sum_{i=1}^{m} Scale_i)$ operations. Considering the maximum iteration number N_{max}, the total time complexity of the algorithm is $O(N_{max} \times K \times n^2 \times \sum_{i=1}^{m} Scale_i)$.

V. EXPERIMENTAL EVALUATION

A. EXPERIMENT SETUP

The first step of experimental evaluation is to select test data set and set up the parameters of our model.

1) TEST DATA

Based on the analysis of the real data from Alibaba Cluster Trace V2018 cluster data set [30], the test data set for this paper is shown in Table 2 and Table 3. The test data set contains an application with 17 microservices.

Table 2 shows the number of requests and the volume of data transmission among microservices when the application receives a unit of user service requests (represented as x1.0 times user requests). For convenience, it uses (ms_i,ms_j) to denote the consumption relationship (ms_{cons},ms_{prov}) between microservices; $(0, ms_i)$ to denote that the clients or users consume a microservice ms_i; $Link_{i,j}$ to denote the number of requests $Link(ms_{cons},ms_{prov})$ between microservices ms_{cons} and ms_{prov}; and $Trans_{i,j}$ to denote the volume of data transmission $Trans(ms_{cons},ms_{prov})$.

Table 3 shows the parameters of microservices in the application. $Link_i$ is the number of requests for microservice ms_i within x1.0 times user requests, which is calculated from the data in Table 1 above; $Scale_i$ is the number of containers of a microservice ms_i in the cluster.

2) PARAMETERS SETTING

We consider a heterogeneous cluster *CLUSTER* as follows:

(1)Number of the physical nodes in the cluster:

$|CLSTER|= 100$;

(2)Physical nodes have three types which differ in the computational capacities:

$Calc\_Resrc_j$=[100.0,200.0,400.0];

Algorithm 1 Ant Colony Optimization Algorithm for Multi-Objective of Container-Based Microservice Scheduling

```
Input:
    $MS\_SET = \{ms_i | i = 1, 2, \ldots, m\}$;
    $CLUSTER = \{pm_j | j = 1, 2, \ldots, n\}$;
    $SCALE = \{Scale_i | i = 1, 2, \ldots, m\}$;
    $LINK = \{Link_i | i = 1, 2, \ldots, m\}$;
    $DIST = \{Dist(pm_j, pm_{j'})\}$;
    $TRANS = \{Trans(ms_i, ms_j)\}$;
    $max_{COMM}, min_{COMM}, max_{LINK\_FAIL}, min_{LINK\_FAIL}$;
    $K, N_{max}$;
Output:
    $PATH = \{path(i, j) | ms_i \in MS\_SET, pm_j \in CLUSTER\}$;
1:  $t \Leftarrow 1$;
2:  Initialize pheromone matrix $\tau_{i,j}(t)$;
3:  while do
4:      $K$ ants are randomly placed to $m$ microservices;
5:      for each $Ant_k$ do
6:          for each $ms_i \in MS\_SET$ do
7:              for each $pm_j \in CLUSTER$ do
8:                  Calculate the heuristic information $\eta_{i,j}^{k}$ by
                    Equation (12) according to the constraints;
9:              end for
10:             for $s = 1; s \leq Scale_i; s++$ do
11:                 for each $pm_j \in CLUSTER$ do
12:                     Calculate the transition probability $p_{i,j}^{k}$ by
                        Equation (17) according to the constraints;
13:                 end for
14:                 According to the calculated probability, select
                    a node $pm_j$, and add $path(i, j)$ to $PATH_k$;
15:                 $x_{i,j}^{k} \Leftarrow 1$;
16:             end for
17:             Put microservice $ms_i$ into $Tabu_k$;
18:         end for
19:     end for
20:     for each $Ant_k$ do
21:         Calculate the evaluation value $Eval(X^k)$ for the
            complete path $PATH_k$ by Equation (13);
22:         Update the optimal solution $PATH$ found so far;
23:     end for
24:     $t \Leftarrow t + 1$;
25:     if $t \leq N_{max}$ then
26:         for each $path(i, j)$ do
27:             Calculate pheromone increment $\Delta\tau_{i,j}$ by Equa-
                tion (15);
28:             Update pheromone matrix $\tau_{i,j}(t)$ by Equa-
                tion (14);
29:         end for
30:         for each $Ant_k$ do
31:             Empty $PATH_k$;
32:             Empty $Tabu_k$;
33:         end for
34:     else
35:         return $PATH$;
36:     end if
37: end while
```

TABLE 2. Number of microservice requests and volume of data transmission per unit user requests (x1.0 times).

(ms_i,ms_j)	$Link_{i,j}$	$Trans_{i,j}$	(ms_i,ms_j)	$Link_{i,j}$	$Trans_{i,j}$
$(0, ms_1)$	50	0	(ms_7,ms_{14})	10	4.1
$(0, ms_3)$	70	0	(ms_8,ms_{14})	15	4.2
$(0, ms_6)$	8	0	(ms_9,ms_5)	20	3.6
$(0, ms_7)$	30	0	(ms_9,ms_{11})	20	4.7
$(0, ms_{10})$	100	0	(ms_{10},ms_5)	20	3.4
$(0, ms_{13})$	30	0	(ms_{10},ms_9)	25	4.4
(ms_1, ms_2)	20	4.6	(ms_{10},ms_{11})	20	4.9
(ms_1, ms_4)	10	3.1	(ms_{11},ms_2)	20	3.2
(ms_1, ms_9)	20	4.0	(ms_{12},ms_8)	45	6.4
(ms_2,ms_4)	10	3.5	(ms_{13},ms_2)	20	4.5
(ms_2,ms_{12})	15	5.9	(ms_{13},ms_8)	45	6.1
(ms_3,ms_{13})	60	1.8	(ms_{13},ms_{16})	8	5.5
(ms_4,ms_{15})	30	5.6	(ms_{13},ms_{17})	30	2.4
(ms_4,ms_{16})	8	5.7	(ms_{15},ms_{16})	8	5.2
(ms_5,ms_{15})	30	5.3	(ms_{16},ms_{14})	15	4.3
(ms_7,ms_2)	20	4.8	(ms_{17},ms_{12})	15	6.2

TABLE 3. Microservices in the application.

ms_i	$CONS\_SET_i$	Cal_Reqst_i	Str_Reqst_i	$Link\_Thr_i$	$Link_i$	$Scale_i$
ms_1	$\{ms_2,ms_4, ms_9\}$	2.1	1.4	10	50	5
ms_2	$\{ms_4,ms_{12}\}$	0.5	3.2	8	80	10
ms_3	$\{ms_{13}\}$	3.1	1.6	8	70	9
ms_4	$\{ms_{15},ms_{16}\}$	4.7	0.2	5	20	4
ms_5	$\{ms_{15}\}$	1.8	3.1	8	40	10
ms_6	$\{\}$	2.5	5.1	4	8	2
ms_7	$\{ms_2,ms_{14}\}$	6.2	0.6	4	30	8
ms_8	$\{ms_{14}\}$	0.8	6.2	4	90	23
ms_9	$\{ms_5,ms_{11}\}$	3.9	2.3	5	45	9
ms_{10}	$\{ms_5,ms_9, ms_{11}\}$	0.2	4.8	4	100	25
ms_{11}	$\{ms_2\}$	2.8	2.6	8	40	5
ms_{12}	$\{ms_8\}$	5.3	0.9	4	30	8
ms_{13}	$\{ms_2,ms_8, ms_{16},ms_{17}\}$	0.6	4.8	5	90	18
ms_{14}	$\{\}$	6.1	2.5	4	40	10
ms_{15}	$\{ms_{16}\}$	1.2	4.2	5	60	12
ms_{16}	$\{ms_{14}\}$	5.4	1.6	4	24	6
ms_{17}	$\{ms_{12}\}$	3.7	2.2	6	30	5

TABLE 4. Parameter setup of ACO_MCMS algorithm.

Parameter	K	N_{max}	α	β	ρ
Value	20	100	2	3	0.05

(3)Three types of different storage capacities corresponding to the computational capacities:

Str_Resrc_j=[100.0, 200.0, 400.0];

(4)Failure rate of the physical nodes:

$Fail_j$ is a random number between 0.01 and 0.03;

(5)Network distance among the physical nodes:

$Dist(pm_j, pm_{j'})$=[1.0, 4.0].

The parameter setup of the ACO_MCMS algorithm are shown in Table 4.

B. RELATED ALGORITHMS FOR COMPARISONS

This paper studies container-based microservice scheduling, which is different from traditional virtual machine-based task or job scheduling. To validate the effectiveness of the ACO_MCMS algorithm, the experiment compares the ACO_MCMS algorithm with the Multiopt algorithm from the paper [20] (mentioned in related work), the GA_MOCA algorithm (GA-based Multi-objective Optimization of Container Allocation) from the paper [23] (mentioned in related work), and the Spread algorithm implemented in Docker Swarm.

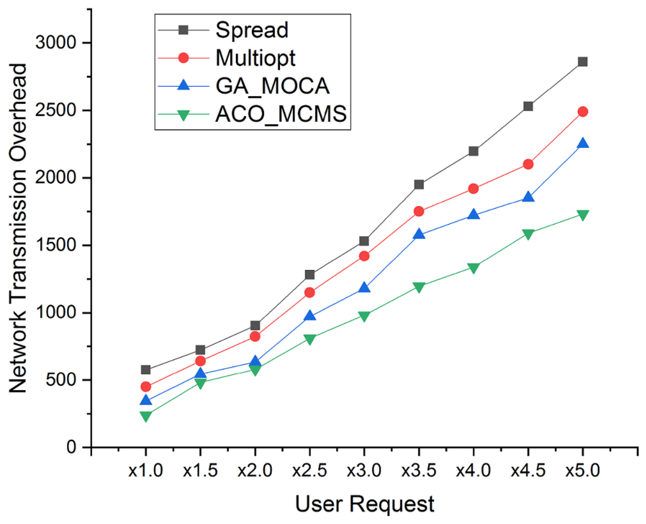


FIGURE 2. Comparative results of network transmission overhead.

-Multiopt algorithm adopts a multi-objective container scheduling method. According the work of this paper, we only consider four related key factors in the Multiopt algorithm, including CPU usage of every node, memory usage of every node, the association between containers and nodes, and the clustering of containers.

-GA_MOCA algorithm is a multi-objective optimization approach based on NSGA-II. The GA_MOCA algorithm considers four optimization objectives, including threshold distance of the container workload, the balanced use of computing resources, the reliability of the container applications, and the intercommunication overhead among related microservices.

-Spread algorithm selects the node with the least number of microservice container instances to place new containers.

C. EXPERIMENTAL RESULTS AND ANALYSIS

The performance comparisons of the four algorithms are performed in three aspects: network transmission overhead, cluster load balancing and reliability of cluster services. We set up nine experimental configurations to analyze and compare the experimental results of four algorithms in the above three aspects. The number of user requests of the nine experimental configurations vary between $\times 1.0$ times and $\times 5.0$ times, and 0.5 times user requests is a span.

1) NETWORK TRANSMISSION OVERHEAD

Network transmission overhead is one of the indicators to evaluate and verify the scheduling effect of the four algorithms.

Figure 2 shows the comparative results of the network transmission overhead for the four algorithms with different number of user requests. The Spread algorithm only considered the resource utilization of the physical nodes, and distributed the microservice containers to each physical node as evenly as possible. The Spread algorithm increased the probability of cross-network transmission among microservices, thus increasing the network transmission overhead between microservices. To reduce network transmission overhead among containers, the Multiopt algorithm allocated the interrelated containers to the same node as much as possible by the clustering of microservice containers. However, the clustering of containers was accomplished by the operation of feature set relation among containers in the nodes, and the algorithm adopted a sequential container scheduling method, in which scheduling sequence of containers has a great influence on the effect of scheduling. Similar to the Multiopt algorithm, the GA_MOCA algorithm just allocated interrelated microservice containers to the physical nodes with short network distances. Neither of the three algorithms above considered the network data transmission among containers. From Figure 2, we can see that the Spread algorithm performed the worst; the performance of the Multiopt algorithm and the GA_MOCA algorithm took the second place; the ACO_MCMS algorithm was the best. It is because the ACO_MCMS algorithm took full account of the network data transmission among microservices and the network distance among the physical nodes where the microservices were allocated, and optimized the scheduling by ant colony optimization method.

2) RESOURCE LOAD OF THE CLUSTER

Standard deviation σ_1 and σ_2 are used to evaluate the load imbalance degree of computing resources and storage resources among the physical nodes, respectively. $\sigma_{cluster}$ captures the load imbalance degree of cluster. Formula for $\sigma_{cluster}$ is as follows.

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

The experimental results are shown in Figure 3, 4, and 5. The standard deviation of computing resource usage fluctuated between 0.13 and 0.25 using the GA_MOCA algorithm as scheduling strategy for microservice containers and that of storage resource usage was between 0.3 and 0.35. For the ACO_MCMS algorithm, they were between 0.14 and 0.27, between 0.17 and 0.25, respectively. For the Spread algorithm, they were between 0.21 and 0.31, between 0.23 and 0.3, respectively. For the Multiopt algorithm, they were between 0.17 and 0.32, between 0.22 and 0.31, respectively. The results show that the GA_MOCA algorithm had the best performance in the load distribution of computing resources but the worst performance in the load distribution of storage resources. It is because the GA_MOCA algorithm only considered the optimization of computing resource usage. For the Spread algorithm and the Multiopt algorithm, in each round of sequential scheduling process, the most suitable node was selected, according to the resource requirements of the microservice container to be deployed and the number

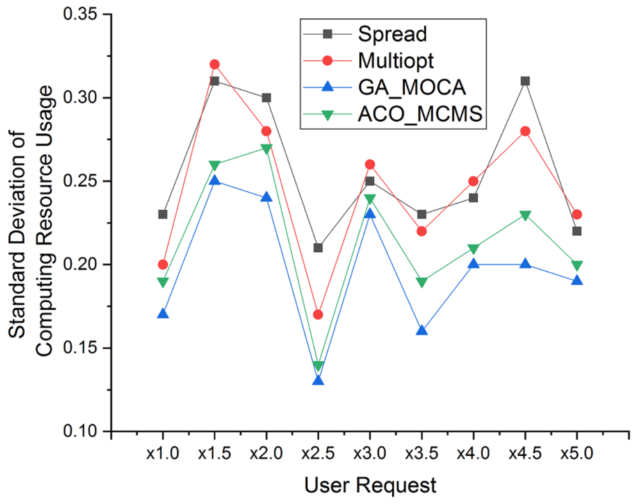


FIGURE 3. **Comparative results of standard deviation of computing resource usage.**

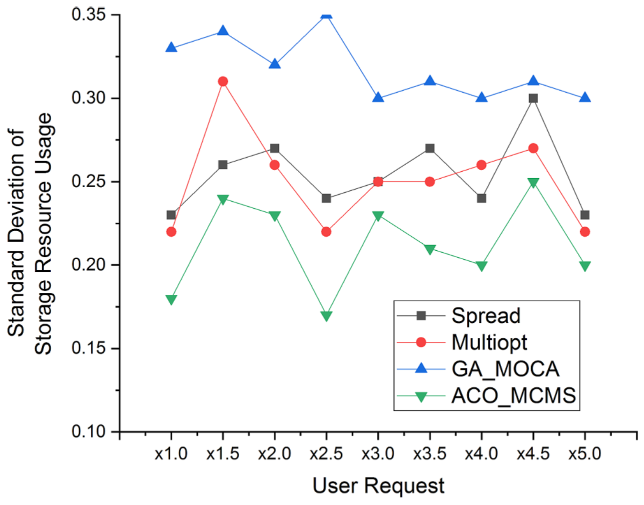


FIGURE 4. **Comparative results of standard deviation of storage resource usage.**

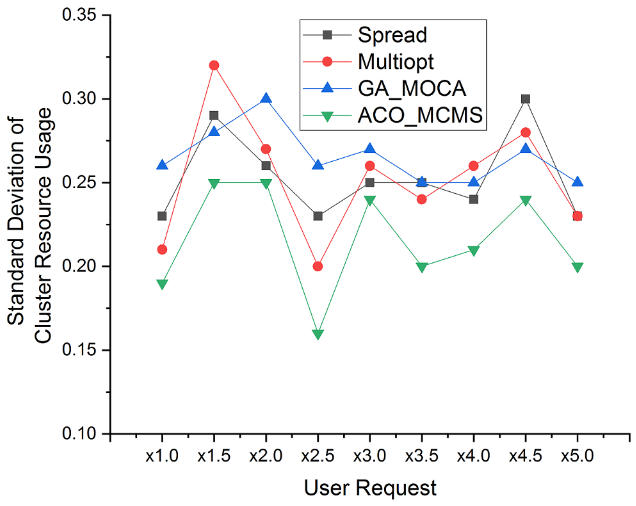


FIGURE 5. **Comparative results of standard deviation of cluster resource usage.**

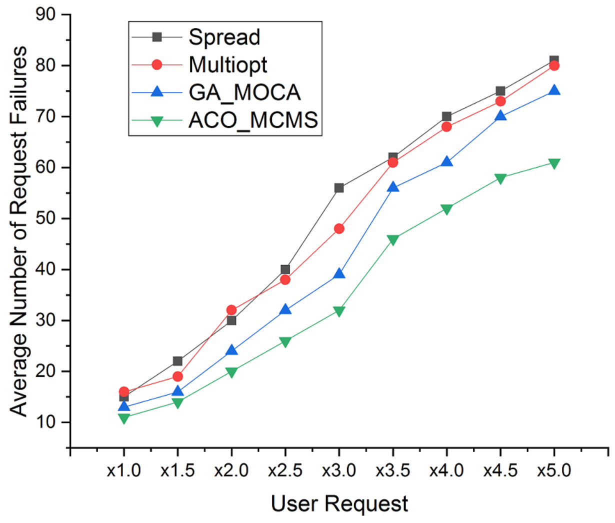


FIGURE 6. Comparative results of the average number of failures for microservice requests.

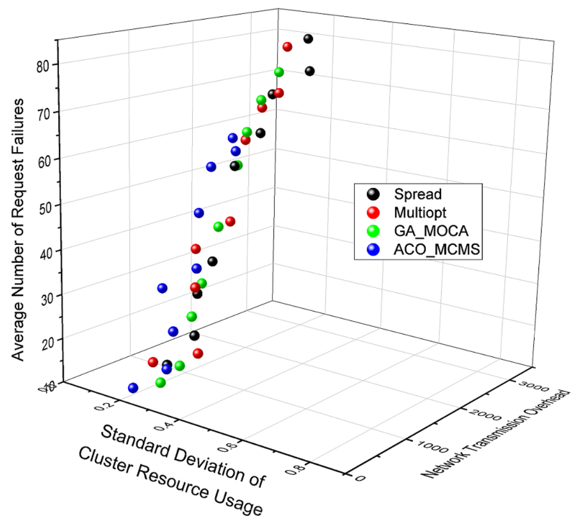


FIGURE 7. Comparative results of multiple indicators.

of deployed containers or resource utilization of the physical nodes in the cluster. However, the ACO_MCMS algorithm considered both the resource utilization rate of the nodes and load distribution of each resource in the cluster through the maximum value of resource utilization rate with coefficient among the physical nodes, and achieved the optimal solution by ACO. From Figure 5, we can see that the ACO_MCMS performed the best in the load balancing of the cluster overall under the various settings of user requests, respectively.

3) RELIABILITY OF CLUSTER SERVICES

The reliability of cluster services is measured by the average number of failures for microservice requests.

Figure 6 shows the average number of failures for microservice requests for four algorithms with different number of user requests. The experimental results of the Spread algorithm and the Multiopt algorithm show an approximate linear growth with the increase of user requests, while that of the GA_MOCA algorithm and the ACO_MCMS algorithm show a slow growth when user requests were low (below $\times 3.0$ times user requests), which were better than the Spread algorithm and the Multiopt algorithm. It is because the Spread algorithm and the Multiopt algorithm did not consider the failure rate of nodes, focusing on load balancing among nodes. The GA_MOCA algorithm considered the failure rate of nodes and microservices, but still did not consider the impact of the number of requests for microservices on the reliability of cluster services. The ACO_MCMS used the average number of failures for microservice requests to measure the reliability of services, and allocated the microservice containers with more requests to the nodes with lower failure rate, so as to reduce the average number of failures for microservice requests and improved the reliability of the cluster services overall. Figure 6 shows that our algorithm performed the best.

4) THREE-DIMENSIONAL COMPARISON OF MULTIPLE INDICATORS

From Figure 7, we can see that the ACO_MCMS algorithm outperformed the Multiopt algorithm, the GA_MOCA algorithm and the Spread algorithm overall by comparing the three performance indicators at the same time. It is because the ACO_MCMS algorithm proposed in this paper is based on the scheduling problem model to optimize the three objectives, so that the three objectives can be optimal simultaneously. However, Spread algorithm only focused on load balancing among the nodes, lacking other optimization strategy. The Multiopt algorithm and the GA_MOCA algorithm ignored the impact of the network data transmission among microservices and the number of microservice requests on scheduling.

VI. CONCLUSION

In this paper, we propose three optimization objectives and establish a multi-objective optimization model, including reducing the network transmission overhead among microservices, balancing the load of the physical nodes in the cluster, and improving reliability of cluster services. A multi-objective optimization method based on ant colony algorithm is proposed to solve the scheduling problem of microservice containers in cloud. In order to prevent ant colony algorithm from falling into local optimal solution and obtain global optimal solution as soon as possible, our method firstly constructs heuristic information to improve the selection probability of the optimal path, by using three optimization objective models. Secondly, it evaluates the quality of solution by using three objective functions in combination, and feeds back the quality ratio between the solution and the current optimal solution to pheromone updating. The quality of the solution in next generation can be improved as much as possible through the mechanisms of evaluation and feedback. Compared with other related algorithms, the proposed optimization algorithm has obvious advantages in reducing network transmission overhead, balancing the load of clusters and improving the reliability of microservices in clusters.

In future work, we plan to apply our scheduling algorithm in a real cloud container cluster and try to reduce the time complexity of the algorithm. In addition, it can also include other optimization objectives, as well as other intelligent optimization algorithms, such as genetic algorithm. Finally, we can study the consumption relationship and resource usage among microservice containers, and use group scheduling instead of single container scheduling.

REFERENCES

[1] A. Balalaie, A. Heydarnoori, and P. Jamshidi, ''Microservices architecture enables devOps: Migration to a cloud-native architecture,'' *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May /Jun. 2016.

[2] 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.

[3] J. Thönes, ''Microservices,'' *IEEE Softw.*, vol. 32, no. 1, p. 116, Jan. /Feb. 2015.

[4] S. Newman, ''Microservices,'' in *Building Microservices: Designing Fine-Grained Systems*. Sebastopol, CA, USA: O'Reilly, 2015, ch.1, sec.1, p. 6.

[5] S. Daya, N. Van Duy, K. Eati, C. M. Ferreira, and D. Glozic, *Microservices From Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. Washington, DC, USA: ITSO, Aug. 2015. [Online]. Available: http://www.redbooks.ibm.com/abstracts/sg248275.html

[6] T. Hoff, *Lessons Learned From Scaling Uber To 2000 Engineers, 1000 Services, and 8000 Git Repositories*. Accessed: Oct. 12, 2016. [Online]. Available: http: highscalability.com/blog/2016/10/12/lessons-learned-from-scaling-uber-to-2000-engineers-1000-ser.html

[7] Z. Ren, W. Wang, G. Wu, C. Gao, W. Chen, J. Wei, and T. Huang, ''Migrating Web applications from monolithic structure to microservices architecture,'' in *Proc. 10th Asia–Pacific Symp. Internet Ware*, Beijing, China, Sep. 2018, p. 7.

[8] D. Merkel, ''Docker: Lightweight Linux containers for consistent development and deployment,'' *Linux J.*, vol. 2014, no. 239, Mar. 2014, Art. no. 2.

[9] H. Yuan, C. Li, and M. Du, ''Optimal virtual machine resources scheduling based on improved particle swarm optimization in cloud computing,'' *J. Softw.*, vol. 9, no. 3, pp. 705–708, Mar. 2014.

[10] P. Li, H. Nie, H. Xu, and L. Dong, ''A minimum-aware container live migration algorithm in the cloud environment,'' *Int. J. Bus. Data Commun. Netw.*, vol. 13, no. 2, pp. 15–27, 2017.

[11] D. Zhang, B. Yan, Z. Feng, C. Zhang, and Y. Wang, ''Container oriented job scheduling using linear programming model,'' in *Proc. 3rd Int. Conf. Inf. Manage. (ICIM)*, Apr. 2017, pp. 174–180.

[12] T. Menouer and C. Cérin, ''Scheduling and resource management allocation system combined with an economic model,'' in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl. IEEE Int. Conf. Ubiquitous Comput. Commun. (ISPA/IUCC)*, Dec. 2017, pp. 807–813.

[13] R. Zhang, A.-M. Zhong, B. Dong, F. Tian, and R. Li, ''Container-VM-PM architecture: A novel architecture for docker container placement,'' in *Proc. Int. Conf. Cloud Comput.*, Seattle, WA, USA, Jun. 2018, pp. 128–140.

[14] O. Adam, Y. C. Lee, and A. Y. Zomaya, ''Stochastic resource provisioning for containerized multi-tier Web services in clouds,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 2060–2073, Jul. 2017.

[15] 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.

[16] X. Guan, X. Wan, B.-Y. Choi, S. Song, and J. Zhu, ''Application oriented dynamic resource allocation for data centers using docker containers,'' *IEEE Commun. Lett.*, vol. 21, no. 3, pp. 504–507, Mar. 2017.

[17] R. Duan, R. Prodan, and X. Li, ''Multi-objective game theoretic scheduling of bag-of-tasks workflows on hybrid clouds,'' *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 29–42, Jan./Mar. 2014.

[18] S. K. Panda and P. K. Jana, ''A multi-objective task scheduling algorithm for heterogeneous multi-cloud environment,'' in *Proc. Int. Conf. Electron. Design, Comput. Netw. Automated Verification (EDCAV)*, Shillong, INDIA, Jan. 2015, pp. 82–87.

[19] X. Zuo, G. Zhang, and W. Tan, ''Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud,'' *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 564–573, Apr. 2014.

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

[21] T. Shi, H. Ma, and G. Chen, ''Multi-objective container consolidation in cloud data centers,'' in *Proc. Australas. Joint Conf. Artif. Intell.*, Nov. 2018, pp. 783–795.

[22] C. Guerrero, I. Lera, and C. Juiz, ''Resource optimization of container orchestration: A case study in multi-cloud microservices-based applications,'' *J. Supercomput.*, vol. 74, no. 7, pp. 2956–2983, Jul. 2018.

[23] 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.

[24] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, ''A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing,'' *IEEE Access*, vol. 3, pp. 2687–2699, 2015.

[25] Q. Guo, ''Task scheduling based on ant colony optimization in cloud environment,'' in *Proc. AIP Conf.*, Apr. 2017, vol. 1834, no. 1, Art. no. 040039.

[26] W. G. Zhang, X. L. Ma, and J. Z. Zhang, ''Research on kubernetes' resource scheduling scheme,'' in *Proc. 8th Int. Conf. Commun. Netw. Secur.*, Qingdao, China, Nov. 2018, pp. 144–148.

[27] C. Kaewkasi and K. Chuenmuneewong, ''Improvement of container scheduling for docker using ant colony optimization,'' in *Proc. 9th Int. Conf. Knowl. Smart Technol. (KST)*, Chonburi, Thailand, Feb. 2017, pp. 254–259.

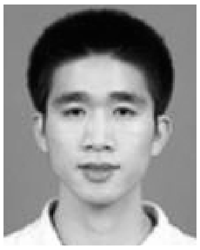
[28] H. Jiao, J. Li, and J. Li, ''The cloud parameters specification and scheduling optimization on multidimensional Qos constraints,'' in *Proc. 15th Int. Comput. Conf. Wavelet Active Media Technol. Inf. Process. (ICCWAMTIP)*, Chengdu, China, Dec. 2018, pp. 22–26.

[29] X. Chen, J. W. Xu, and D. Long, ''Resource scheduling algorithm of cloud computing based on ant colony optimization-shuffled frog leading algorithm,'' *J. Comput. Appl.*, vol. 38, no. 6, pp. 1670–1674, Jun. 2018.

[30] Alibaba Corp. *Alibaba Cluster Trace V2018*. Accessed: Dec. 3, 2018. [Online]. Available: https://github.com/alibaba/clusterdata

[31] S. Fu, ''Failure-aware resource management for high-availability computing clusters with distributed virtual machines,'' *J. Parallel Distrib. Comput.*, vol. 70, no. 4, pp. 384–393, 2010.

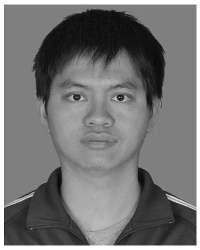
[32] T. Wang, H. Xu, and F. Liu, ''Multi-resource load balancing for virtual network functions,'' in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Atlanta, GA, USA, Jun. 2017, pp. 1322–1332.



MIAO LIN received the B.S. degree from the Department of Information and Computing Science, Northeastern University, in 2011, and the M.S. degree from the College of Information Science and Engineering, Northeastern University, in 2013. He is currently pursuing the Ph.D. degree with the School of Software Engineering, South China University of Technology. His research interests include cloud computing, parallel processing, and high-performance computing.

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

WEIHUA BAI received the M.S. 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, in 2017. He is currently an Associate Professor with Zhaoqing University. His research interests include cloud computing, parallel scheduling, and software architectures. He is also a member of the China Computer Federation.



JIAYIN WU received the B.S. degree from the South China University of Technology, in 2007, and the M.S. degree from the School of Physics and Engineering, Sun Yat-sen University, in 2010. He is currently a teacher with Guangdong Vocational College of Post and Telecom. His research interests include cloud computing and its application on scientific computing.