Computer Science and Information Systems 18(1):23–42 https://doi.org/10.2298/CSIS200229041F

Multi-Objective Optimization of Container-Based Microservice Scheduling in Edge Computing

Guisheng Fan[1,2], Liang Chen[1], Huiqun Yu[1], and Wei Qi[1]

[1] Department of Computer Science and Engineering
East China University of Science and Technology, Shanghai, China
{gsfan,yhq}@ecust.edu.cn, chanliang_china@163.com
[2] Shanghai Key Laboratory of Computer Software Evaluating and Testing
Shanghai,China

Abstract. Edge computing provides physical resources closer to end users, becoming a good complement to cloud computing. With the rapid development of container technology and microservice architecture, container orchestration has become a hot issue. However, the container-based microservice scheduling problem in edge computing is still urgent to be solved. In this paper, we first formulate the container-based microservice scheduling as a multi-objective optimization problem, aiming to optimize network latency among microservices, reliability of microservice applications and load balancing of the cluster. We further propose a latency, reliability and load balancing aware scheduling (LRLBAS) algorithm to determine the container-based microservice deployment in edge computing. Our proposed algorithm is based on particle swarm optimization (PSO). In addition, we give a handling strategy to separate the fitness function from constraints, so that each particle has two fitness values. In the proposed algorithm, a new particle comparison criterion is introduced and a certain proportion of infeasible particles are reserved adaptively. Extensive simulation experiments are conducted to demonstrate the effectiveness and efficiency of the proposed algorithm compared with other related algorithms.

Keywords: edge computing, microservice, container orchestration, multi-objective optimization, particle swarm optimization.

1. Introduction

Recently, the emerging edge computing paradigm is seen as an effective solution to the problem of big data, which brings the processing to the edge of the network [1]. It has the advantage of shorter response time and can save bandwidth and energy required for data transmission in cloud computing [2, 3]. At the same time, microservice architecture [4] has become increasingly popular in the process of application design and development, which is commonly used to develop cloud native applications. However, there are few researches on microservice scheduling in edge computing.

As a lightweight virtualization technology, container is the perfect tool to encapsulate and deploy microservices. With the development of container technology and the widespread use of microservice architecture, some practical container scheduling strategies have been proposed. However, there are still some important problems to be solved

in container-based microservice scheduling in edge computing. Current container cluster management tools, including Docker Swarm, Apache Mesos, and Google Kubernetes, only implement simple strategies of assigning containers to physical nodes. These strategies only consider physical resources usages [5], without implementing optimization strategies for the reliability of applications, network transmission latency, etc. It is possible for researchers to obtain better results in terms of network transmission latency, reliability of microservice applications and load balancing of the cluster.

Container scheduling in edge computing is a typical NP-hard problem. Such problems must be addressed using heuristic algorithms. Particle swarm optimization (PSO) is one of the most common heuristic algorithms. Many researchers have adopted PSO to solve the problem of task scheduling or scientific workflow scheduling in distributed computing. Thus, we propose a method to implement a container resource scheduling strategy by using PSO algorithm.

In order to tackle the container-based microservice scheduling problem in edge computing, we first formulate it as a multi-objective optimization problem, in which network transmission latency among microservices, reliability of microservice applications and load balancing of the cluster can be optimized. Then we propose a latency, reliability and load balancing aware scheduling algorithm for microservice scheduling system to determine the deployment of container-based microservices. The main contributions of this paper are as follows:

- We mathematically model the container-based microservice scheduling problem in edge computing to reduce network transmission latency among microservices, improve reliability of microservice applications and balance the cluster load, the resource capacity constraints of edge nodes are also considered.
- An LRLBAS algorithm based on particle swarm optimization (PSO) is proposed to solve the multi-objective optimization problem for container-based microservice scheduling. It can be used to separate the fitness function from constraints, so that each particle has two fitness values. The new comparison criterion for particles is introduced and a certain proportion of the infeasible particles are reserved adaptively.
- Several experiments are done to evaluate the proposed algorithm. The experiment results demonstrate that our algorithm generally outperforms the other two methods in terms of objectives, fitness value and optimization speed when the number of user requests is large. And it can obtain optimization results with relatively little running overhead when the number of user requests is small.

The remainder of this paper is organized as follows. Section 2 introduces the related work. Section 3 describes the system architecture and analytical models. Section 4 provides the problem formulation. Section 5 presents the implementation of our LRLBAS algorithm. Section 6 illustrates the experimental settings and the experimental results. Section 7 summarizes this paper and raises the future work.

2. Related Work

Resource management optimization is a hot research topic in the field of distributed computing. In this paper, the related research is presented in three main parts: container orchestration, multi-objective optimization in resource management and scheduling methods based on particle swarm optimization (PSO) algorithm.

First, some related works on container orchestration are showed here. Adam et al. [6] present Two-stage Stochastic Programming Resource Allocator (2SPRA). It optimizes resource provisioning for containerized n-tier web services in accordance with fluctuations of incoming workload to accommodate predefined service-level objectives (SLOs) on response latency and reduces resource over-provisioning. Li et al. [7] propose an optimal minimum migration algorithm (OMNM) which reduces the unnecessary migration of containers. By fitting the growth rate of Docker containers in the source server, the model can estimate the growth trend of each Docker container and determine which container needs to be migrated. The algorithm aims to reduce the number of the migration and improve the utilization ratio of the resource, while ensuring the load balancing of the cluster. Kaewkasi and Chuenmuneewong [8] present a container scheduling algorithm based on Ant Colony Optimization (ACO), aiming to balance the resource usages and finally lead to the better performance of applications. Their approach is compared with the results obtained with a greedy algorithm. Guerrero et al. [9] propose a genetic algorithm approach, using the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) to optimize container allocation and elasticity management. Their approach is compared with the container management policies implemented in Google Kubernetes. Tao et al. [10] introduce a schedule algorithm based on fuzzy inference system (FIS), for global container resource allocation by evaluating nodes' statuses using FIS. The algorithm aims to derive optimal resource configurations and improve the performance of the cluster. However, only the paper [9] considers the use of microservice architecture, but Guerrero et al. do not include the network transmission latency among container-based microservices in their models.

Second, in the research of resource management optimization in distributed computing, there may exist multiple conflicting objectives, and researchers need to optimize these objectives simultaneously. Therefore, there have been many researches on multi-objective optimization methods in this field. Guerrero et al. [11] present an approach based on NSGA-II to optimize the deployment of microservice applications using containers in multi-cloud architectures. The optimization objectives are three: cloud service cost, network latency among microservices, and time to start a new microservice when a provider becomes unavailable. Azimzadeh and Biabani [12] present a multi-objective optimization method for resource management and task assignment based on genetic algorithm, in order to reduce execution time and enhance reliability of service. Langhnoja and Joshiyara [13] propose a novel scheduling algorithm called multi-objective based Integrated Task scheduling which aims to solve task scheduling problem of cloud computing, considering three optimization objectives: execution time, execution cost and load balancing. Mireslami et al. [14] propose a multi-objective resource allocation model when deploying a Web application in cloud, considering deployment cost and quality of service (QoS) simultaneously. The algorithm aims to minimize cost, maximize QoS and get a balanced trade-off between the two conflicting objectives. Zhang et al. [15] introduce an adaptive container scheduler based on integer linear programming, which considers three factors: 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. Lin et al. [16] establish a multi-objective optimization model for the container-based microservice scheduling, and propose an ant colony algorithm to solve the scheduling problem. The algorithm aims to optimize cluster service reliability, cluster load balancing, and network transmission overhead. However, all of these work

above focus on resource management in cloud, rather than the emerging edge computing paradigm. The work of Lin et al. [16] is the most similar one to our approach, but they do not provide a rigorous mathematical representation of the data transmission latency among microservices.

Third, as an intelligent algorithm, particle swarm optimization (PSO) is one of the most commonly used scheduling algorithms in the field of resource scheduling. Zhang and Yang [17] propose a task scheduling algorithm based on an improved PSO, which can schedule efficiently, shorten the task completion time and improve the utilization of resources in cloud computing. Pan and Chen [18] establish a resource-task allocation model and propose an improved PSO algorithm to achieve resource load balancing in the cloud environment. Chou et al. [19] propose the dynamic power-saving resource allocation (DPRA) mechanism based on a particle swarm optimization algorithm, aiming to improve energy efficiency for cloud data centers. Verma et al. [20] propose a hybrid PSO algorithm based on non-dominance sort for handling the workflow scheduling problem with multiple objectives in the cloud. Li et al. [21] propose a security and cost aware scheduling (SCAS) algorithm based on PSO for heterogeneous tasks of scientific workflow in cloud, aiming to minimize the total workflow execution cost while meeting the deadline and risk rate constraints. Li et al. [22] propose a PSO-based container scheduling algorithm of Docker platform, which aims to solve the problem of insufficient resource utilization and load imbalance. The algorithm distributed application containers on Docker hosts, balance resource usage, and ultimately improve application performance. However, only the paper [22] focuses on container scheduling, the paper [17], [18], [19], [20], [21] focus on task scheduling or workflow scheduling. Moreover, when solving constrained optimization problems, only the paper [21] separates the fitness function from constraints and adopts a novel comparison criterion of particles.

Despite a large number of solutions and implementations mentioned above, researches on container-based microservice scheduling in edge computing environment are still very limited. In this paper, we describe it as a multi-objective optimization problem and an LRLBAS algorithm based on PSO is implemented to solve it. This paper aims to optimize the network transmission latency among microservices, the reliability of microservice applications and the load balancing of the cluster simultaneously.

3. System Architecture and Analytical Models

As shown in Fig. 1, the system is mainly divided into two layers. The User Layer is used to send service requests to microservice applications. The Edge Cloud Layer consists of physical resources that are used to process requests from users. Users send their requests to microservice applications deployed on Edge Cloud. Then, the physical resources are allocated to related microservices encapsulated in containers by microservice scheduling system (MSM).

Container-based microservice scheduling in edge computing can be characterized by properties from three components, i.e., application model, network model, and computation model. The application model refers to the container-based microservice application being scheduled, the network model describes the infrastructure used to execute the microservice application, and the computation model corresponds to what we attempt to

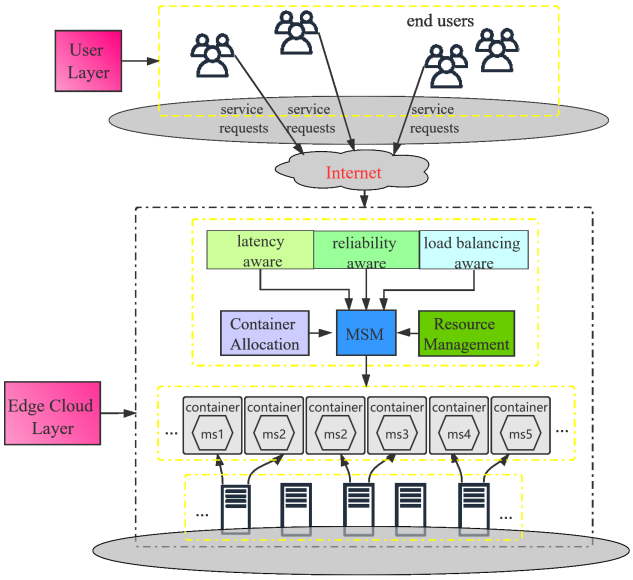


Fig. 1. System architecture

optimize. For convenience of reference, we summarize and tabulate the parameters and their descriptions used in the models in Table 1.

3.1. Application Model

We consider an application A developed by microservice architecture. A is modeled as a directed graph $G_a = \langle ms\_set, ms\_relation \rangle$, where $ms\_set = \{ms_1, ms_2, \ldots, ms_m\}$ is the set of microservices of application A; $ms\_relation$ is the set of dependencies among the microservices. When the execution of microservice ms_i requires the result generated by another microservice ms_k, the dependency between them is established, denoted by $(ms_i, ms_k) \in ms\_relation$.

Microservice ms_i is characterized as a tuple $\langle calc\_need_i, str\_need_i, max\_link_i \rangle$, where $calc\_need_i$ represents the computing resources required by one request for microservice ms_i; str_need_i is the storage resources required by one request for microservice ms_i; max_link_i is the maximum number of requests for one instance of microservice ms_i. In addition, pre_set_i is the preceeding set of microservices that provide data for microservice ms_i to execute, and when a microservice $ms_k \in pre\_set_i$, there exists $(ms_i, ms_k) \in ms\_relation$.

Application A receives service requests from users. User requests for microservice ms_k is mainly divided into two parts. One is the direct requests from users, denoted by $direct\_reqst_k$; the other is the indirect requests from other microservices. The number of indirect requests from microservice ms_i to ms_k is denoted by $link(ms_i, ms_k)$, so the total number of requests for microservice ms_k is calculated as $link_k = direct\_reqst_k +$

$\sum_{i=1 \wedge ms_k \in pre\_set_i}^{m} link(ms_i, ms_k)$; $trans(ms_i, ms_k)$ denotes the size of data transmitted in a request between microservice ms_i and ms_k. We do not consider the network transmission latency associated with direct requests from users to microservices. In addition, $scale_k$ is the number of instances of microservice ms_k in the cluster. According to the number of requests for microservice ms_k and the maximum number of requests for one instance of microservice ms_k, $scale_k$ can be calculated as $\lceil \frac{link_k}{max\_link_k} \rceil$.

3.2. Network Model

The underlying edge computing environment for running microservice applications is modeled as a fully-connected directed graph $G_e = \langle node\_set, link\_set \rangle$, where $node\_set = \{node_1, node_2, \ldots, node_n\}$ is the set of edge nodes; $link\_set$ represents the set of directed links between edge nodes. Each communication link $l_{i,j}$ between $node_i$ and $node_j$ is related to bandwidth $b_{i,j}$ and network distance $d_{i,j}$.

Edge node $node_j$ is characterized as a tuple $\langle calc_j, str_j, fail_j \rangle$, where $calc_j$ is the computing resource capacity of edge node $node_j$; str_j indicates the storage resource capacity of edge node $node_j$; $fail_j$ represents the failure rate of edge node $node_j$.

3.3. Computation Model

Network Transmission Latency among Microservices. The network transmission latency among microservices is related to four key factors: the number of requests between two interoperable microservice instances, the size of data transmission in a request between the two microservice instances, the bandwidth and the network distance between the edge nodes where the two microservice instances are allocated. Considering that both consumer microservices and provider microservices may have multiple container-based instances, we allocate the requests between the two microservices evenly among their instance pairs. This is formalized in Equation (1):

$$trans\_latency = \sum_{k=1}^{m} \sum_{q=1}^{n} y_{k,q} \sum_{i=1 \wedge ms_k \in pre\_set_i}^{m} \sum_{p=1}^{n} y_{i,p} * lc(i,k,p,q), \tag{1}$$

where:

$$lc(i,k,p,q) = \frac{link(ms_i, ms_k)}{scale_i \times scale_k} \times (\frac{trans(ms_i, ms_k)}{b_{q,p}} + \frac{d_{q,p}}{c}). \tag{2}$$

Here, $lc(i,k,p,q)$ represents the network transmission latency between one instance pair of microservice ms_i and ms_k deployed on edge node $node_p$ and $node_q$ respectively, $y_{i,p}$ denotes the number of instances of microservice ms_i deployed on edge node $node_p$, and c is the propagation rate of the electromagnetic wave over the channel, approximately $3 \times 10^8 m/s$.

Average Number of Failures for Microservice Requests. The average number of failures for microservice requests measures the reliability of microservice applications, which is related to two key factors: the number of requests for microservices and the failure rates of edge nodes. Considering that edge nodes in the cluster may break down for some reason, microservices deployed on these edge nodes will not available and user requests will

fail. This is mathematically modeled in Equation (4), which is used in [15]:

$$fail\_reqst = \sum_{j=1}^{n}\sum_{i=1}^{m} y_{i,j} \times fail_j \frac{link_i}{scale_i}. \tag{3}$$

where $y_{i,j}$ denotes the number of instances of microservice ms_i deployed on edge node $node_j$.

Imbalance Degree of Resource Usages of Edge Nodes. The imbalance degree of resource usages of edge nodes measures the load balancing of the cluster. In this paper, we consider the computing resources and storage resources simultaneously, so balancing cluster resource load is a Multi-Resource Load Balancing (MRLB) problem. To deal with the load balancing of the cluster, the standard deviations of utilization rate of physical resources in edge nodes are calculated, and then used as coefficient value for the utilization rate of corresponding resource in each node [15]. The maximum value of resource utilization rate with coefficient among edge nodes reflects the worst case about load blancing of the cluster. So, the imbalance degree of resource usages of edge nodes is formalized in Equation (4):

$$imbalance = \frac{Max(util_1, util_2, \ldots, util_j, \ldots, util_n)}{\sigma_{calc} + \sigma_{str}} \quad 1 \leq j \leq n, \tag{4}$$

where:

$$util_j = Max(\sigma_{calc} \times calc\_usage_j, \sigma_{str} \times str\_usage_j), \tag{5}$$

$$calc\_usage_j = \sum_{i=1}^{m} y_{i,j} \frac{link_i \times calc\_need_i}{scale_i \times calc_j}, \tag{6}$$

$$str\_usage_j = \sum_{i=1}^{m} y_{i,j} \frac{link_i \times str\_need_i}{scale_i \times str_j}. \tag{7}$$

Here,σ_{calc}, σ_{str} represents the standard deviation values of utilization rate of computing resources and storage resources of edge nodes in the cluster respectively; $util_j$ is the bigger value of resource utilization rate with coefficient of edge node $node_j$; $calc\_usage_j$, str_usage_j are the utilization rate values of computing resources and storage resources of edge node $node_j$.

4. Problem Formulation

4.1. Multi-Objective Optimization Model

According to the three objective functions mentioned in Section 3.3, we establish the following multi-objective optimization model of container-based microservice scheduling in edge computing.

$$min \quad trans\_latency, \tag{8}$$

$$min \quad fail\_reqst, \tag{9}$$

$$min \quad imbalance, \tag{10}$$

$subject \quad to:$

$$\sum_{i=1}^{m} y_{i,j} \frac{link_i}{scale_i} calc\_need_i - calc_j \leq 0 \quad \forall node_j, \tag{11}$$

$$\sum_{i=1}^{m} y_{i,j} \frac{link_i}{scale_i} str\_need_i - str_j \leq 0 \quad \forall node_j. \tag{12}$$

Equation (8)-(10) represent the three optimization objectives respectively: minimizing the network transmission latency among microservices, minimizing the average number of failing requests for microservices and minimizing the imbalance degree of resource usages of edge nodes. Equation (11)-(12) represent the computing and storage resource constraints of edge nodes respectively.

Table 1. Summary of parameters and their descriptions

Parameters	Descriptions
$G_a = \langle ms\_set, ms\_relation \rangle$	microservice application
m	the number of microservices in the application
ms_i	microservice with id. i
$(ms_i, ms_k) \in ms\_relation$	dependency link from microservice ms_i to ms_k
$calc\_need_i$	computing resources required by one request for microservice ms_i
str_need_i	storage resources required by one request for microservice ms_i
max_link_i	the maximum number of requests for one instance of microservice ms_i
pre_set_i	preceeding set of microservices of microservice ms_i
$direct\_reqst_i$	the number of direct requests for microservice ms_i from users
$link(ms_i, ms_k)$	the number of indirect requests from microservice ms_i to ms_k
$link_i$	the total number of requests for microservice ms_i
$trans(ms_i, ms_k)$	size of data transmitted between microservice ms_k and ms_i
$scale_i$	the number of instances of microservice ms_i
$G_e = \langle node\_set, link\_set \rangle$	edge computing environment
n	the number of edge nodes in the cluster
$node_j$	edge node with id. j
$calc_j$	computing resource capacity of edge node $node_j$
str_j	storage resource capacity of edge node $node_j$
$fail_j$	failure rate of edge node $node_j$
$l_{i,j}$	communication link between edge node $node_i$ and $node_j$
$b_{i,j}$	bandwidth of link $l_{i,j}$
$d_{i,j}$	network distance of link $l_{i,j}$

4.2. Fitness Function

Based on the aforementioned multi-objective optimization model, we use linear weighted sum method to modify the multi-objective optimization problem into a single-objective optimization problem and construct the fitness function of this paper as follows:

$$f(X) = w_1 \times \varphi(trans\_latency) + w_2 \times \varphi(fail\_reqst) + w_3 \times \varphi(imbalance). \quad (13)$$

where X is a scheduling scheme that maps microservices to edge nodes; $w_1, w_2, w_3 \geq 0$ and $w_1 + w_2 + w_3 = 1$. For the weight coefficients of optimization objectives, the most important objective generally has the maximum weight coefficient value according to the user preferences. $\varphi(l)$ is a normalized function for optimization objectives, defined as:

$$\varphi(l) = \frac{l - l_{min}}{l_{max} - l_{min}}. \quad (14)$$

In this paper, we repeatly do experiments on the three optimization objectives for 30 times, and replace the maximum and minimum values of the three objectives with their corresponding empirical constant values.

According to the analysis above, we formally define a container-based microservice scheduling problem in edge computing environment: given a directed graph structured microservice application $G_a = \langle ms\_set, ms\_relation \rangle$, a fully-connected edge computing environment $G_e = \langle node\_set, link\_set \rangle$, we wish to find a schedule X: $ms_i \rightarrow node_j$, $\forall ms_i \in ms\_set$, $\exists node_j \in node\_set$, such that minimizes the fitness function under the resource capacity of edge nodes. The problem can be formally described by Equation (15):

$$\begin{cases} find \quad X = \{x_1, x_2, \ldots, x_D\}, \\ which \quad min(f(X)), \\ subject \quad to: \sum_{i=1}^{m} y_{i,j} \frac{link_i}{scale_i} calc\_need_i - calc_j \leq 0 \quad \forall node_j, \\ \sum_{i=1}^{m} y_{i,j} \frac{link_i}{scale_i} str\_need_i - str_j \leq 0 \quad \forall node_j. \end{cases} \quad (15)$$

where D is the dimension of the schedule scheme X.

5. LRLBAS Algorithm Implementation

The above defined problem is a typical NP-hard problem. So, we consider using heuristic algorithms to obtain its near-optimal solution. Particle swarm optimization (PSO) is a frequently used heuristic algorithm, which is developed by Kennedy and Eberhart [23]. In this work, our latency, reliability and load balancing aware scheduling (LRLBAS) algorithm is based on PSO.

The basic idea of PSO is to search the optimal solution through the cooperation and information sharing among individuals in a population. Suppose that one population has N particles and the searching space is D dimensional. For a particle $P_i (i = 1, 2, \ldots, N)$, it has three typical parameters that are the position $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$, velocity $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$ and its optimal position $pbest_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$. In the k_{th}

iteration of PSO, the velocity and position of particle P_i will be updated by the following two equations:

$$V_i^k = w^k \cdot V_i^{k-1} + c_1 \cdot r_1 \cdot (pbest_i - X_i^{k-1}) + c_2 \cdot r_2 \cdot (gbest - X_i^{k-1}). \quad (16)$$

$$X_i^k = X_i^{k-1} + V_i^k. \quad (17)$$

where c_1 and c_2 denote learning factors, r_1 and r_2 are random numbers from the range of [0,1], w^k is called inertia weight that influences search capability of particles, $gbest$ is the current global optimal position.

Shi and Eberhart [24] define the inertia weight w as a decreasing function, that is

$$w^k = w_{start} - (w_{start} - w_{end}) \cdot \frac{k}{M}. \quad (18)$$

where w_{start} and w_{end} are the initial value and ending value of inertia weight w, M indicates the maximum number of iterations in PSO.

5.1. Non-linear Inertia Weight

To better balance the global and local search abilities of particles, a novel method for updating the inertia weight w is used [25], as shown in Equation (19):

$$w^k = w_{end} + (w_{start} - w_{end}) \cdot \sin(\frac{\pi}{2}\sqrt{(1 - \frac{k}{M})^3}). \quad (19)$$

Compared with the linear inertia weight in Equation (18), the non-linear inertia weight in Equation (19) is larger at the beginning period, which can promote global search in the early stage of the optimization process. When the number of iterations gradually approaches the maximum value M, the non-linear inertia weight is smaller than the linear inertia weight, which can improve local search in the late stage of the optimization process.

5.2. Constraints Handing

To deal with the constraints, we adopt a strategy similar to that in [20]. Our constraint handling strategy separates the fitness function from constraints, so that each particle has two fitness values. In addition, a new comparison criterion for particles is introduced and a certain proportion of the infeasible particles are reserved adaptively.

In this paper, the general form of our problem is expressed in Equation (20):

$$min \quad f(X) \quad s.t. \quad g_j(X) \leq 0 \quad j = 1, 2, \ldots, q \quad (20)$$

After separating the fitness function from constraints, the original problem can be transformed into Equation (21):

$$fitness(i) = f(X), \quad violation(i) = \sum_{j=1}^{q} max(0, g_j(x)) \quad i = 1, 2, \ldots, N \quad (21)$$

Here, the former formula represents the fitness value of particle P_i in a certain iteration, namely the first fitness value; the latter is the constraint violation value of particle P_i, that is, the second fitness value. The constraint violation value of a feasible solution is 0.

Then, we use the following comparison criteria for particles: firstly, a constant β is given. (1) Between two feasible particles P_i and P_j, compare their fitness values $fitness(i)$ and $fitness(j)$, the smaller one is better; (2) between two infeasible particles P_i and P_j, compare their constraint violation values $violation(i)$ and $violation(j)$, the smaller one is better; (3) between the feasible particle P_i and the infeasible particle P_j, if $violation(j)$ is smaller than β , then compare their fitness values $fitness(i)$ and $fitness(j)$, the smaller one is better; otherwise, particle P_i is better.

During the optimization process, the proportion of infeasible solutions changes dynamically. If the proportion becomes too large, most particles will move towards infeasible solutions. If the proportion becomes too small, our algorithm will not work very well and the optimization efficiency will be compromised. So, we hope the proportion of infeasible solutions can fluctuate around a fixed value p. Based on the above comparison criteria, we can know that the larger the value of constant β, the larger the proportion of infeasible solutions is likely to be. To keep the proportion around p, the following adaptive adjustment strategy for β is used: (1) when the proportion is smaller than p, $\beta = 1.5\beta$; (2) when the proportion is larger than p, $\beta = 0.5\beta$; (3) when the proportion is equal to p, the value of β does not change.

5.3. Algorithm Implementation

Based on implementation steps mentioned above, we design the LRLBAS algorithm based on PSO for microservice applications. The implementation of our algorithm is shown as the pseudo-code of Algorithm 1.

This algorithm first initializes position (i.e., the scheduling scheme), velocity of all particles and other necessary parameters (see lines 1-7). Next, update velocity, position, inertia weight and the value of β (see lines 10-14). Then, evaluate the fitness value and constraint violation value of each particle according to constraints handling, update the optimal solution of each particle and select the global optimal solution (see lines 16-19). Finally, the algorithm returns the near-optimal scheduling scheme (see line 21).

6. Performance Evaluation

6.1. Experimental Setup

In this paper, the test data set is shown in Table 2 and Table 3. The microservice application in this test data set is composed of 17 microservices.

Table 2 shows the number of requests and the amount of data transmission among microservices when the microservice application receives a unit of user service requests (represented as 1.0reqs). Here, (-,ms_i) represents users consume microservice ms_i directly. For convenience, we use $link_{i,k}$ and $trans_{i,k}$ to denote $link(ms_i, ms_k)$ and $trans(ms_i, ms_k)$, respectively. Table 3 shows the parameters of microservices in the application; $link_i$ represents the number of requests for microservice ms_i when the microservice application receives 1.0reqs; $scale_i$ is the number of instances of microservice ms_i in the cluster.

Some details about the experimental setup are shown in Table 4. Table 4(a) presents parameters of our LRLBAS algorithm. Parameter settings for the edge node cluster are described in Table 4(b).

Algorithm 1 LRLBAS algorithm based on PSO

Input: related information about the microservice application, a set of edge nodes, maximum number of iteration M, size of particle swarm N.

Output: the near-optimal scheduling solution X_{best}.

for $i = 1$ to N **do**
    Randomize the initialization of scheduling X_i, search velocity V_i and some other necessary parameters;
end for
for $i = 1$ to N **do**
    Set current position of scheduling X_i as $pbest_i$;
end for
Select the best near-optimal scheduling plan of minimum fitness from N scheduling plans as $gbest$;
for $j = 1$ to M **do**
    for $i = 1$ to N **do**
        Update the velocity of particle V_i by Equation (16);
        Update the position of particle X_i by Equation (17);
    end for
    Update the inertia weight w^k by Equation (19);
    Compute the ratio of infeasible solutions and update;
    for $i = 1$ to N **do**
        Evaluate the fitness value and the violation value of scheduling plan X_i according to constraints handling;
        Compare the current particles fitness evaluation with $pbest_i$. If current value is better than $pbest_i$, then update $pbest_i$;
    end for
    Select the best near-optimal scheduling plan of minimum fitness from N scheduling plans as $gbest$;
end for
return X_{best}

In addition, this paper assume that three objectives are equally important, so their weight coefficients w_1, w_2, w_3 are all set as 1/3.

6.2. The Comparison of both Objectives and Fitness Value

In this paper, we compare the LRLBAS algorithm with other scheduling algorithms including the original PSO (OPSO) based scheduling algorithm and the directional and non-local-convergent PSO (DNCPSO) based scheduling algorithm proposed in [25]. The main principles and steps of the two algorithms are shown below.

(1) Original PSO (OPSO). The original PSO has been described in Section 4 and proved to be a useful intelligent heuristic algorithm. It searches the optimal solution through the cooperation and information sharing among individuals in a population.
(2) Directional and non-local-convergent PSO (DNCPSO). The authors in [25] propose a directional and non-local-convergent particle swarm optimization algorithm to perform workflow scheduling in cloud-edge environment. This algorithm firstly uses

Table 2. Number of requests and amount of data transmission under 1.0reqs

(ms_i, ms_k)	$link_{i,k}$	$trans_{i,k}$(MB)	(ms_i, ms_k)	$link_{i,k}$	$trans_{i,k}$(MB)
(-,ms_1)	25	0	(ms_7, ms_{14})	5	2.1
(-,ms_3)	35	0	(ms_8, ms_{14})	8	2.1
(-,ms_6)	4	0	(ms_9, ms_5)	10	1.8
(-,ms_7)	15	0	(ms_9, ms_{11})	10	2.4
(-,ms_{10})	50	0	(ms_{10}, ms_5)	10	1.7
(-,ms_{13})	15	0	(ms_{10}, ms_9)	13	2.2
(ms_1, ms_2)	10	2.3	(ms_{10}, ms_{11})	10	2.5
(ms_1, ms_4)	5	1.6	(ms_{11}, ms_2)	10	1.6
(ms_1, ms_9)	10	2.0	(ms_{12}, ms_8)	23	3.2
(ms_2, ms_4)	5	1.8	(ms_{13}, ms_2)	10	2.3
(ms_2, ms_{12})	8	3.0	(ms_{13}, ms_8)	23	3.1
(ms_3, ms_{13})	30	0.9	(ms_{13}, ms_{16})	4	2.8
(ms_4, ms_{15})	15	2.8	(ms_{13}, ms_{17})	15	1.2
(ms_4, ms_{16})	4	2.9	(ms_{15}, ms_{16})	4	2.6
(ms_5, ms_{15})	15	2.7	(ms_{16}, ms_{14})	8	2.2
(ms_7, ms_2)	10	2.4	(ms_{17}, ms_{12})	8	3.1

Table 3. Microservices in the application

ms_i	pre_set_i	$calc\_need_i$	str_need_i	max_link_i	$link_i$	$scale_i$
ms_1	$\{ms_2, ms_4, ms_9\}$	2.1	1.4	10	25	3
ms_2	$\{ms_4, ms_{12}\}$	0.5	3.2	8	40	5
ms_3	$\{ms_{13}\}$	3.1	1.6	8	35	5
ms_4	$\{ms_{15}, ms_{16}\}$	4.7	0.2	5	10	2
ms_5	$\{ms_{15}\}$	1.8	3.1	8	20	3
ms_6	$\{\}$	2.5	5.1	4	4	1
ms_7	$\{ms_2, ms_{14}\}$	6.2	0.6	4	15	4
ms_8	$\{ms_{14}\}$	0.8	6.2	4	45	12
ms_9	$\{ms_5, ms_{11}\}$	3.9	2.3	5	23	5
ms_{10}	$\{ms_5, ms_9, ms_{11}\}$	0.2	4.8	4	50	13
ms_{11}	$\{ms_2\}$	2.8	2.6	8	20	3
ms_{12}	$\{ms_8\}$	5.3	0.9	4	15	4
ms_{13}	$\{ms_2, ms_8, ms_{16}, ms_{17}\}$	0.6	4.8	5	45	9
ms_{14}	$\{\}$	6.1	2.5	4	20	5
ms_{15}	$\{ms_{16}\}$	1.2	4.2	5	30	6
ms_{16}	$\{ms_{14}\}$	5.4	1.6	4	12	3
ms_{17}	$\{ms_{12}\}$	3.7	2.2	6	15	3

Table 4. Parameter settings

(a) Parameters of the LRLBAS algorithm

Parameter	Value
Population size	50
Maximum number of iterations	300
w_{start}	0.9
w_{end}	0.4
c_1, c_2	2
r_1, r_2	[0,1]
β	10
p	0.2

(b) Parameters of the edge node cluster

Parameter	Value
Number of edge nodes	120
$calc_i$	$\{100, 200, 400\}$
str_i	$\{100, 200, 400\}$
$fail_i$	$\{0.01, 0.02, 0.03\}$
$b_{i,j}$(Mbps)	$\{200, 400\}$
$d_{i,j}$(km)	[30,300]

non-linear inertia weight to better balance the global and local search abilities of particles. Then, it replace random search with directional search which can improve the optimization speed of the algorithm. Finally, selection and mutation operations are integrated into this algorithm, which is conducive to jump out of local optimum. So, this algorithm can get better near-optimal solution in a faster speed.

In this subsection,the performance comparisons of the three algorithms are performed in four aspects: network transmission latency, reliability of the microservice application, cluster load balancing and fitness value. We present the experimental results of three algorithms in the above four aspects under five experimental configurations. The number of user requests of the five experimental configurations varies between 1.0reqs and 3.0reqs, with an interval of 0.5reqs.

As shown in Fig. 2, the values have been normalized between 0.0 and 1.0. We can see that LRLBAS algorithm achieves better performance (smaller objective values) than OPSO in four aspects under five experimental configurations, and obtains better optimization results than DNCPSO in 12 of the total 20 scenarios.

In detail, as shown in Fig. 2(d-e), LRLBAS algorithm performs better than DNCPSO in all four aspects under 2.5reqs and 3.0reqs. However, in Fig. 2(a-c), LRLBAS algorithm obtains objective values that are slightly higher than DNCPSO in 7 scenarios. Because the proportion of infeasible solutions in the population is small when the number of user requests is small. So the infeasible solutions are not enough for LRLBAS algorithm to find better solutions.

6.3. The Comparison of Optimization Process for Fitness Value

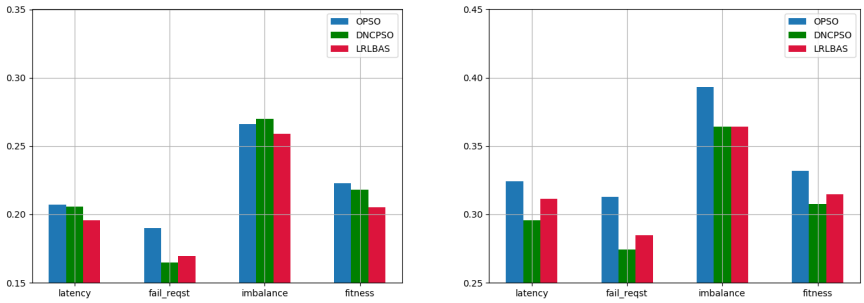
In this subsection, we compare the LRLBAS algorithm with other algorithms by the iterative trend of fitness value under five experimental configurations. The iterative trend of each algorithm for searching results includes two aspects, namely searching speed and nearest optimal solution. The searching speed indicates the fewest number of iterations that is required to find the near-optimal solution. The nearest optimal solution indicates the minimum fitness value that the algorithms can reach.

As shown in Fig. 3, LRLBAS algorithm can obtain smaller fitness values than the other two algorithms in most cases. Among the three algorithms, OPSO performs worst under five experimental configurations.

In detailed comparison, as shown in Fig. 3(c), the fitness value in DNCPSO declines significantly faster than that in our LRLBAS algorithm, indicating that DNCPSO can obtain the near-optimal solution with fewer iterations than LRLBAS algorithm. Moreover, DNCPSO obtains a smaller fitness value at the end of the iteration process. So DNCPSO performs better than LRLBAS algorithm under 2.0reqs. In addition, as shown in Fig. 3(d-e), LRLBAS algorithm performs better than DNCPSO in terms of searching speed and nearest optimal solution. The experimental results in Fig. 3 are consistent with the optimization results of fitness value in Fig. 2.

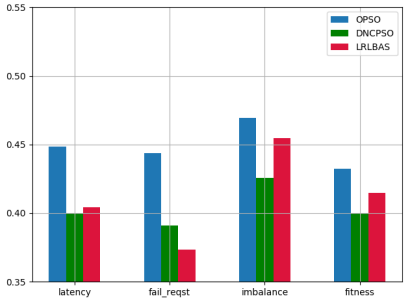
6.4. The Comparison of Sensitivity

As shown in Fig. 4, the values have been normalized between 0.0 and 1.0. We can see that our LRLBAS algorithm is the least sensitive algorithm as its curve slope has the

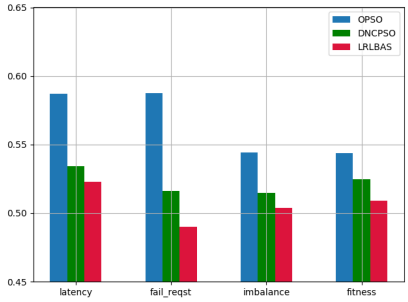


(a) 1.0reqs

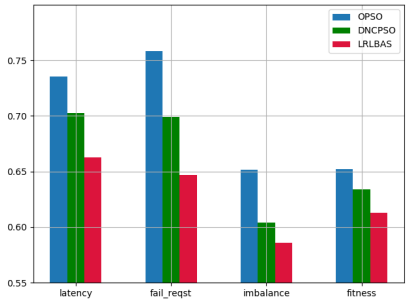
(b) 1.5reqs



(c) 2.0reqs



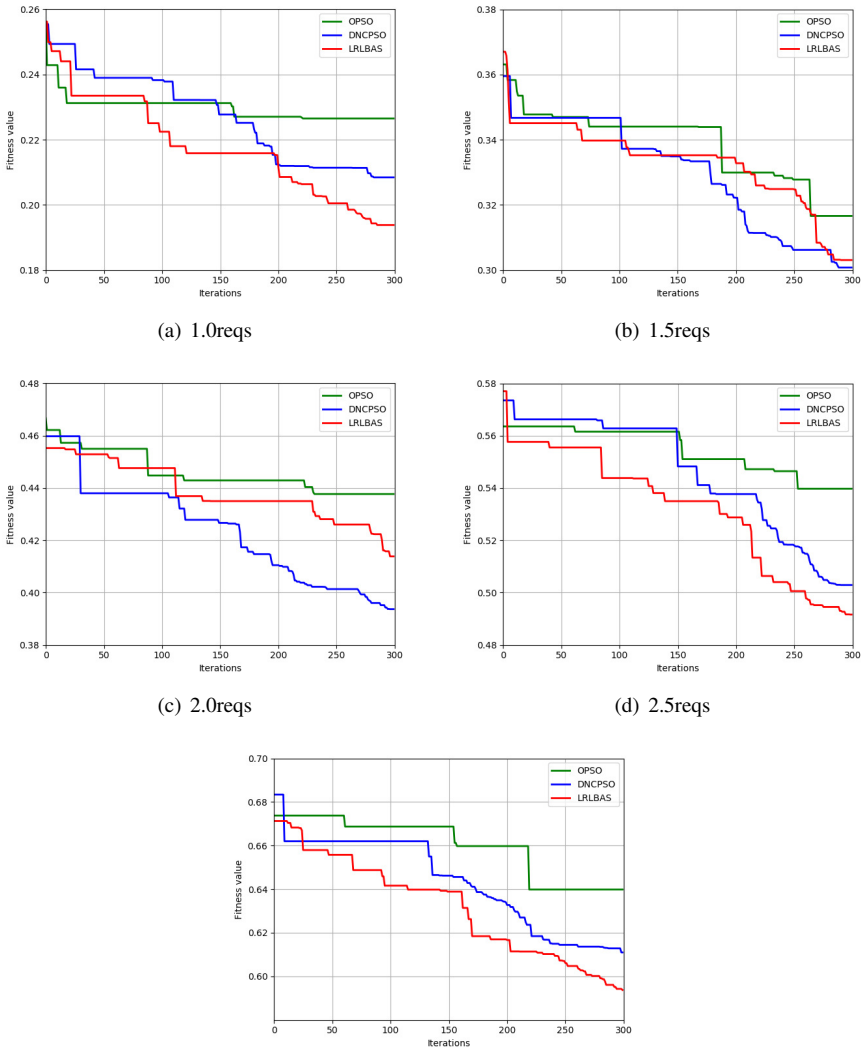
(d) 2.5reqs



(e) 3.0reqs

Fig. 2. Normalized objective and fitness values obtained with three algorithms

least obvious change in terms of both objectives and fitness value as the number of user requests increases, which has more adaptability to the situation that the number of user requests increases. Also, LRLBAS can obtain smaller objective and fitness values in most cases, which is consistent with the experimental results in Fig. 2.



(a) 1.0reqs (b) 1.5reqs

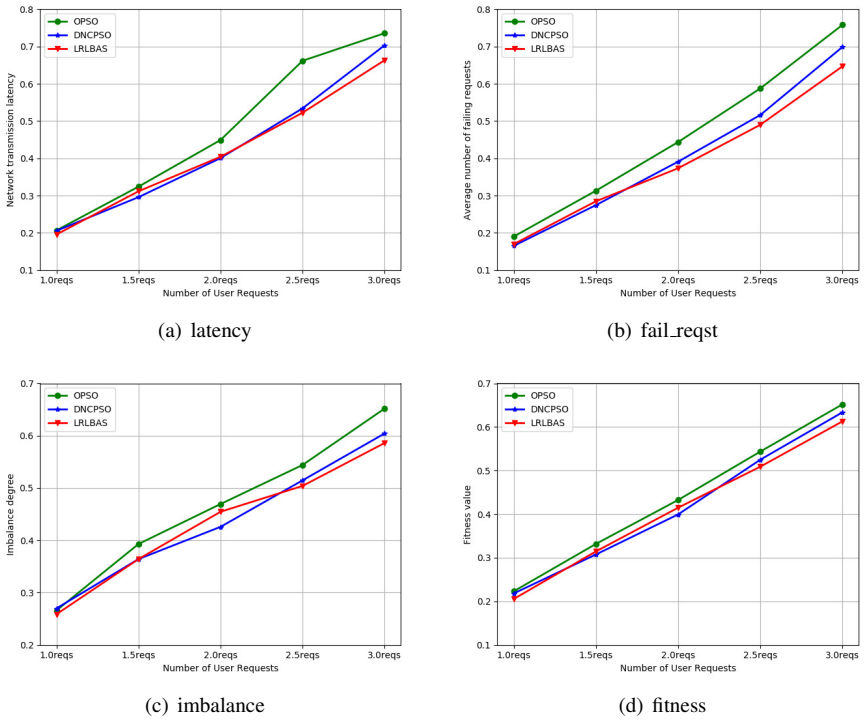
(c) 2.0reqs (d) 2.5reqs

(e) 3.0reqs

Fig. 3. The changing process of fitness value under different iterations

6.5. The Comparison of Running Overhead for Fitness Value

In this paper, the running time required to perform a optimization process for fitness value is used as the evaluation metric of algorithms running overhead. Here, The final result of the running time is calculated by running an average of 30 times. We compare our LRL-BAS algorithm with other algorithms by the running overhead under five experimental configurations.



(a) latency (b) fail_reqst

(c) imbalance (d) fitness

Fig. 4. The comparison of sensitivity among three algorithms

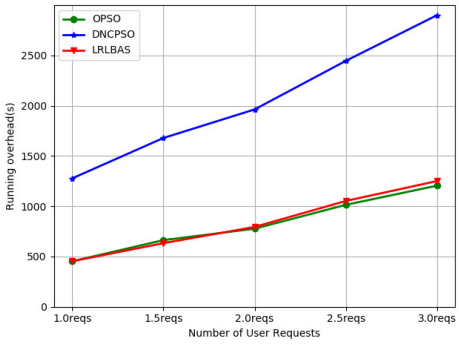


Fig. 5. Running overhead under different number of user requests

As shown in Fig. 5, we can see that the running overhead of the LRLBAS algorithm is nearly equal to that of OPSO, while the running overhead of DNCPSO is much higher than that of OPSO. As mentioned above, DNCPSO performs significantly better than our LRLBAS algorithm under 2.0reqs, but it costs significant running overhead.

Through the above several groups of experiments, it can demonstrate that LRLBAS algorithm achieves better optimization results than the other two algorithms in terms of objectives, fitness value and optimization speed when the number of user requests is large. When the number of user requests is small, although LRLBAS algorithm performs worse than DNCPSO in some cases, it consumes significantly less running overhead than DNCPSO. Therefore, it can be proved that the LRLBAS algorithm for container-based microservice scheduling in edge computing proposed in this paper is effective and efficient.

7. Conclusion

In this paper, container-based microservice scheduling in edge computing is described as a multi-objective optimization problem, aiming to reduce the network transmission latency among microservices, improve the reliability of microservice applications and balance the cluster load. We propose a latency, reliability and load balancing aware scheduling algorithm for microservice applications in edge computing. Our proposed algorithm is based on the PSO. Extensive experiments demonstrate the effectiveness and efficiency of our algorithm for microservice scheduling in edge computing.

In the future, we plan to take other optimization objectives into account. In addition, more scheduling algorithms can be added for performance comparison. Finally, we can study the results of our microservice scheduling algorithm in a real edge computing container cluster.

Acknowledgements. This work was partially supported by the NSF of China under Grant nos. 61702334 and 61772200, Shanghai Municipal Natural Science Foundation under Grant nos. 17ZR1406900 and 17ZR1429700, Action Plan for Innovation on Science and Technology Projects of Shanghai under Grant no. 16511101000, Collaborative Innovation Foundation of Shanghai Institute of Technology under Grant no. XTCX2016-20, and Educational Research Fund of ECUST under Grant no. ZH1726108.

References

1. Khan, W.Z., Ahmed, E., Hakak, S., Yaqoob, I., Ahmed, A.: Edge computing: A survey. Future Gener. Comput. Syst. 97, 219-235 (2019)
2. Mukherjee, M., Shu, L., Wang, D.: Survey of fog computing: fundamental, network applications, and research challenges. IEEE Commun. Surv. Tutorials 20(3), 1826-1857 (2018)
3. Souza, A., Wen, Z., Cacho, N., Romanovsky, A., James, P, Ranjan, R.: Using osmotic services composition for dynamic load balancing of smart city applications. In: Proceedings of 2018 IEEE 11th International Conference on Service-Oriented Computing and Applications. Paris, pp. 145-152 (2018)
4. Lewis, J., Fowler., M.: Microservices: a definition of this new architectural term. [Online]. Available: https://www.martinfowler.com/articles/microservices.html (2014)
5. Fazio, M., Celesti, A., Ranjan, R., Liu, C., Chen, L., Villari, M.: Open issues in scheduling microservices in the cloud. IEEE Cloud Comput. 3(5), 81-88 (2016)
6. Adam, O., Lee, Y.C., Zomaya., A.Y.: Stochastic resource provisioning for containerized multi-tier web services in clouds. IEEE Trans. Parallel Distrib. Syst. 28(7), 2060-2073 (2017)

7. Li, P., Nie, H., Xu, H., Dong, L.: A minimum-aware container live migration algorithm in the cloud environment. Int. J. Bus. Data Commun. Netw. 13(2), 15-27 (2017)
8. Kaewkasi, C., Chuenmuneewong, K.: Improvement of container scheduling for docker using ant colony optimization. In: 2017 9th International Conference on Knowledge and Smart Technology. Chonburi, pp. 254-259 (2017)
9. Guerrero, C., Lera, I., Juiz., C.: Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. J. Grid Comput. 16(1), 113-135 (2018)
10. Tao, Y., Wang, X., Xu, X., Chen, Y.: Dynamic resource allocation algorithm for container-based service computing. In: 2017 IEEE 13th International Symposium on Autonomous Decentralized System. Bangkok, pp. 61-67 (2017)
11. Guerrero, C., Lera, I., Juiz., C.: Resource optimization of container orchestration: A case study in multi-cloud microservices-based applications. J. Supercomput. 74(7), 2956-2983 (2018)
12. Azimzadeh, F., Biabani., F.: Multi-objective job scheduling algorithm in cloud computing based on reliability and time. In: 2017 3th International Conference on Web Research. Tehran, pp. 96-101 (2017)
13. Langhnoja, H.K., Hetal Joshiyara, P.A.: Multi-objective based integrated task scheduling in cloud computing. In: 2019 3rd International conference on Electronics, Communication and Aerospace Technology. Coimbatore, pp. 1306-1311 (2019)
14. Mireslami, S., Rakai, L., Far, B.H., Wang, M.: Simultaneous cost and QoS optimization for cloud resource allocation. IEEE Trans. Netw. Service Manage. 14(3), 676-689 (2017)
15. Zhang, D., Yan, B., Feng, Z., Zhang, C., Wang, Y.: Container oriented job scheduling using linear programming model. In: 2017 3th International Conference on Information Management. Chengdu, pp. 174-180 (2017)
16. Lin, M., Xi, J., Bai, W., Wu, J.: Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. IEEE Access 7, 83088-83100 (2019)
17. Zhang, Y., Yang, R.: Cloud computing task scheduling based on improved particle swarm optimization algorithm. In: Proceedings IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society. Beijing, pp. 8768-8772 (2017)
18. Pan, K., Chen, J.: Load balancing in cloud computing environment based on an improved particle swarm optimization. In: 2015 6th IEEE International Conference on Software Engineering and Service Science. Beijing, pp. 595-598 (2015)
19. Chou, L., Chen, H., Tseng, F., Chao, H., Chang, Y.: DPRA: Dynamic power-saving resource allocation for cloud data center using particle swarm optimization. IEEE Syst. J. 12(2), 1554-1565 (2018)
20. Verma, A., Kaushal S.: A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. Parallel Comput. 62, 1-19 (2017)
21. Li, Z., Ge, J., Yang, H., Huang, L., Hu, H., Hu, H., Luo, B.: A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds. Future Gener. Comput. Syst. 65, 140-152 (2016)
22. Li, L.W., Chen, J.X., Yan, W.Y.: A particle swarm optimization-based container scheduling algorithm of docker platform. In: Proceedings of 2018 4th International Conference on Communication and Information Processing. Qingdao, pp. 12-17 (2018)
23. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks. Perth, pp. 1942-1948 (1995)
24. Shi, Y., Eberhart R.: Empirical study of particle swarm optimization. In: Proceedings of the 1999 Congress on Evolutionary Computation. Washington, pp. 1945-1950 (1999)
25. Xie, Y., Zhu, Y., Wang, Y., Cheng, Y., Xu, R., Sani, A.S., Yuan, D., Yang, Y.: A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment. Future Gener. Comput. Syst. 97, 361-378 (2019)

Guisheng Fan received his B.S. degree from Anhui University of Technology in 2003, M.S.degree from East China University of Science and Technology (ECUST) in 2006, and Ph.D. degree from ECUST in 2009, all in computer science. He is presently a research assistant of the Department of Computer Science and Engineering at ECUST. His research interests include formal methods for complex software system, service oriented computing, and techniques for analysis of software architecture.

Liang Chen received his B.S. degree in computer science from Nanjing Tech University in 2018. He is currently a graduate student at East China University of Science and Technology (ECUST). His research interests include software engineering, edge computing and microservice.

Huiqun Yu received his B.S. degree from Nanjing University in 1989, M.S.degree from East China University of Science and Technology (ECUST) in 1992, and PhD.degree from Shanghai Jiaotong University in 1995, all in computer science. He is currently a Professor of computer science with the Department of Computer Science and Engineering at ECUST. From 2001 to 2004, he was a Visiting Researcher in the School of Computer Science at Florida International University. His research interests include software engineering, high confidence computing systems, cloud computing and formal methods. He is a senior member of IEEE, China Computer Federation.

Wei Qi received her B.S. degree in computer science from East China University of Science and Technology (ECUST) in 2018. She is currently a graduate student at ECUST. Her research interests include software engineering, cyber-physical system.

Received: February 29, 2020; Accepted: July 20, 2020.