Particle Swarm Optimization Algorithm for Container Deployment

Libin WU¹,², Hongbin XIA¹,²,*

¹ School of Digital Media, Jiangnan University, Wuxi, Jiangsu 214122, China
² Jiangsu Key Laboratory of Media Design and Software Technology, Wuxi, Jiangsu 214122, China

*Corresponding author’s e-mail: hbxia@163.com

Abstract: In recent years, with the development of cloud computing, virtualization technology has received widespread attention. As a new representative of virtualization technology, containers have been widely used in software development, operation and maintenance, testing and other aspects, such as microservices and Docker Cloud. In cloud data centers, containers have gradually replaced virtual machines (VMs) as a new carrier for cloud tasks. However, with the increasing number of cloud products, the scale of tasks requested by users in cloud data centers continues to expand. The economic cost of tasks in the process of containerized deployment has become a concern of various cloud service vendors. The container deployment cost usually includes the data exchange cost, the image pull cost and the server energy cost. In order to solve the containerized deployment of application tasks in the container cloud environment with the lowest possible container deployment cost, this paper proposes a new cost calculation model in a container cloud environment, and then presents an improved particles swarm algorithm, namely a particle swarm optimization (PSO) algorithm for container deployment (CD-PSO) to provide the best solution for application task loading. Experimental results show that the proposed algorithm has a lower deployment cost than other scheduling algorithms.

1. Introduction

Containers are a group of processes running on the host operating system to provide isolated environment technology [1], [2]. It is based on the Linux kernel technology implementation [3], in which Namespce is used to isolate the environment required for program running; Cgroups implements resource control; Rootfs implements file system isolation and some container engines to manage its own life cycle. As an emerging virtualization technology, containers have many advantages in providing a working platform, such as efficient portability, small footprint and startup time of physical nodes, scalable computing resources to a smaller granularity, and simpler management. Nowadays, more and more enterprises are willing to put their business on the cloud. With its unique advantages, containers provide more efficient services for customers in terms of task execution. For large-scale, distributed data processing, however, deployment of a large number of containers on suitable servers to achieve the containerized deployment of user tasks at the lowest possible cost remains a research topic in the scientific community.

The containerized deployment of tasks refers to assigning user tasks to different server hosts in the cluster through a reasonable scheduling algorithm or policy and meeting various constraints. This consists of completing the mapping from tasks to hosts, requiring hosts to complete the pull of image
files, configuration of requested resources, and to start daemons, etc. to achieve containerization of tasks on the host. In this process, the proper scheduling of tasks is the key step for container deployment. In cloud computing, many research works on task scheduling, including applications of scheduling algorithms on container deployment, have been presented. For example, Ye Tao et al. [4] proposed a dynamic resource scheduling algorithm based on container service computing to provide the best resource allocation for each service scheduling; Kaewkasi et al. [5] proposed improvement of Container Scheduling for Docker using Ant Colony Optimization to balance use of cluster node resources; Bhiman et al. [6] developed a new docker controller for scheduling workload containers of different types of applications to help speed-up the overall execution of multiple applications on SSDs; Wu et al. [7] proposed a flexible deferrable server scheduler to improve the performance of Docker containerized mixed-criticality real-time system, which might run applications with different criticalities as well as timing constraints; Zhang et al. [8] proposed a dynamic load balancing-based scheduling strategy for container clusters; Cerin et al. [9] proposed a new scheduling strategy based on different Service Level Agreement (SLA) classes. These scheduling algorithms and strategies solve the problems of node load imbalance, low cluster scalability, and low fault tolerance during container deployment. Aiming at reducing the cost of cloud data centers, Canosa et al. [10] proposed a scheduling strategy with energy consumption and quality of service optimization; Cheng et al. [11] used automatic scaling technology to achieve the dynamic allocation of container resources to reduce the cost of container adjustment; Wu et al. [12] proposed a lightweight cloud platform (Rattrap) and developed a new operating environment (Cloud Android Container) for mobile computing offloading, thus replacing the heavy weight virtual machines (VMs) and saving the uninstall cost caused by traditional cloud platforms. Our research also aims to reduce the cost of container data center. The main contributions of this article are two-fold. Firstly, we build a cluster environment consisting of clients, management host, and execution hosts by combining them with the cloud environment. In addition, we propose a calculation model for container deployment costs on this cluster. Secondly, we present an improved particle swarm optimization algorithm for container deployment (CD-PSO), where the novel cost model is used for performance measurement. Finally, the CD-PSO is applied to the containerized deployment of tasks.

The rest of the paper is organized as follows. Section II introduces the related work and background. Section III presents the objective function and formulation of the container deployment problem. Section IV presents the CD-PSO algorithm. Section V discusses the experiment results. Section VI concludes the work and provides possible future directions.

2. Related Work And Background

This section introduces the related work of reducing economic costs in the cloud environment and the background of the particle swarm optimization (PSO) algorithms.

2.1. Related work

Of long standing it remains a problem as to how to reduce the economic cost of cloud data centers. The study in [13] finds that the biggest economic cost in the data center is caused by the energy consumption of the server, while the energy consumption of the server is related to the allocation of the task carrier. In data centers that use VMs as their primary carrier, Kurpicz et al. [14] introduced a model for Energy-Proportional Accounting in VM-based Environments; Dabbagh et al. [15] proposed an energy-efficient resource allocation framework for overcommitted clouds, which monitors the VM resource usage and reducing the number of active PMs via efficient VM migration and placement; The method proposed by Sonkar et al. [16] not only improves the overall utilization of server resources, achieves the benefit of reducing costs, but also helps to combine various types of workloads; Tian et al. [17] research focuses on the energy-efficient scheduling of VM reservations in a cloud data center. On the other hand, in the cloud environment which is based on the container technology, researchers have conducted the following research to reduce costs. Al-Rakhami et al. [18] proposed a container technology named Raspberry Pi devices, which is applied to the recognition of human activities; Shi et al. [19] proposed a two-stage multi-type particle swarm optimization method (TMPSO) for
energy-aware container consolidation in cloud data centers. This method can save program execution time while optimizing, but the effect of energy consumption optimization is not obvious; Hanafy et al. [20] proposed a novel container and host selection policies for cloud deployment models, and experimentally demonstrated that the proposed policies could achieve better energy and services level agreement compared to the other competitors; Guan et al. [21] argued that the deployment cost of the same container might be different on different nodes considering image files reusing. They presented a framework for Application Oriented Docker Container (AODC) to reduce deployment costs. This solution starts a tray container for each application to better manage task execution, but it tends to occupy extra resources; Zhang et al. [22] combined container image pulling costs, the cost of node energy consumption and the workload network transition costs between clients and hosts to form an optimization problem. A linear optimization algorithm (LP) was used to find the task allocation scheme with the lowest cost. However, this solution lacks a management host to uniformly handle task scheduling, instead, it allows the client to directly contact the server.

Combining the existing scheduling strategies and the characteristics of the task containerization process, we summarize several types of costs that are easy to incur in the container deployment process: 1) The data exchange costs generated by the large amount of data transfer between hosts in the data center; 2) The image pull cost incurred in the process when a host pulls the image files from the image registry; 3) The energy cost of a host server on which the container runs. In this paper, we refer to the sum of the costs incurred during the task containerization process as the container deployment cost, which is composed of the aforementioned types of costs. This work is to explore the means of achieving the containerization of user-submitted tasks with the lowest container deployment cost. We deal the work as an optimization problem.

2.2. Background of PSO
Particle swarm algorithm [23] is a swarm intelligence optimization algorithm proposed by Ebehart and Kennedy, which is derived from the research of bird swarm predation behavior. For each particle, which has a position and velocity like a bird, we use vectors $\mathbf{X}_j = (x_{j1}, x_{j2}, \ldots, x_{jn})$ and $\mathbf{V}_j = (v_{j1}, v_{j2}, \ldots, v_{jn})$ to represent them. When the algorithm is initialized, the number of N particles are randomly generated as the effective solution of the problem in search space. With each iterative search, the particles are based on their individual optimal positions $\mathbf{P}_{bestj} = (pb_{j1}, pb_{j2}, \ldots, pb_{jn})$ and the global optimal position of the current group $\mathbf{G}_{best} = (Gb_{1}, Gb_{2}, \ldots, Gb_{n})$ to update their speed and position, and after a certain number of iterations, get the final optimization result of the problem.

The speed and position update of particle $j$ satisfy the following formula: 

$$V_j(k+1) = W V_j(k) + c_1 r_1 (\mathbf{P}_{bestj} - \mathbf{X}_j) + c_2 r_2 (\mathbf{G}_{best} - \mathbf{X}_j) \tag{1}$$

$$X_j(k+1) = X_j(k) + V_j(k+1) \tag{2}$$

In the formula (1) and (2), $k$ represents the current number of iterations of the algorithm, and $X_j(k)$ and $V_j(k)$ represent the position and velocity of particle $j$ at the $k$-th iteration, respectively. $W$ is the inertia weight, which is usually initialized to 0.9, $c_1$ and $c_2$ are the learning factors, which control the particle’s tendency to the global optimal solution. Usually, their values are 2, and $r_1$ and $r_2$ are random numbers uniformly distributed between 0 and 1.

3. Problem Formulation
This section describes a container deployment cost calculation model in a cluster environment, and then the formulation of container deployment problem is discussed.

3.1. Deployment cost model
A typical scenario in a container cloud environment is shown in Figure 1. One or more users submit resource requests (Here mainly refers to computing resources) from clients to the management host, and the management host divides the request task into one or more subtasks. After that, it is queued into a queue $tq$, waiting for the task allocation plan made by the scheduling module, loading the task on the
designated host, and completing the containerized deployment of the task.

![Diagram of container cloud environment](image-url)

**Figure 1.** container cloud environment diagram.

The key issue in the entire process described above is the design of the task scheduler, which dispatches the task load to the execution host, to meet the following criterions.

1) The task loads scheduled to a specific host cannot exceed the host’s computing capacity.
2) The total number of scheduled task loads should remain fixed during the containerization process.

To meet criterion 1, denote $x_i$ as the task loads on node $i$, $node(i)_{capacities}$ is the amount of computing resources on node $i$, then a constraint is set as:

$$0 \leq x_i \leq node(i)_{capacities}$$  

(3)

This constraint ensures that all the task loads running on host $i$ should not exceed its computing capacity.

To meet criterion 2, denote $node_{number}$ as the number of execution hosts in the cluster, another constraint is set as:

$$\sum_{i=1}^{node_{number}} x_i = T$$  

(4)

The value of $T$ is a fixed value. This constraint ensures that the total task load on the cluster server remains the same during the particle optimization process.

Data exchange costs occur between the management host and the execution host. The management host needs to load tasks to the execution host, causing data transfer. In [21], the link cost model in the AODC strategy is used to calculate the cost of data exchange between two hosts. Based on this, the costs of data exchange between the management host and the execution hosts is defined as:

$$data_{cost} = \sum_{i=1}^{node_{number}} distance(i) * traffic_{cost} * x_i$$  

(5)

$distance(i)$ is defined as the distance between the execution host $i$ and the management host, and $traffic_{cost}$ is the cost of traffic through this path.

When the task is started in the container of the execution host, the execution host needs download the image from the image registry, it will generate the image pull cost. The image is composed of multiple layer files and most files are basic files that can be shared [24]. If the image files required for all tasks in the container have been downloaded on the host, there is no need to pull costs, but if the execution host
contains only the basic files required for some tasks, it is necessary to pull the missing files, as shown in Figure 2. The image pulling cost is defined as:

$$\text{im}_\text{cost} = \sum_{i=1}^{\text{node\_number}} \text{pct}_i \cdot x_i \cdot \text{pull\_cost} \cdot y$$  \hspace{1cm} (6)

Because the missing files can only be determined by combining the storage situation of the host image file and the task requirements, we use a random strategy to estimate the percentage of missing files, $\text{pct}_i$, is defined as:

$$\text{pct}_i = \text{rand}(0,1)$$  \hspace{1cm} (7)

$\text{pull\_cost}_i$ is defined as the unit file pulling cost of host $i$. $y \in (0,1)$ indicates whether node $i$ has a task load.

![Image Pulling Diagram](image.png)

Figure 2. Pull images diagram.

In addition, when the container is running on the host (usually a execution node is a server in a container cloud environment), there will also be an energy cost. According to the server energy consumption model proposed by E. Feller et al. [25], we use the utilization of computing resources to estimate the energy cost.

$$\text{energy}_\text{cost} = \begin{cases} 
\sum_{i=1}^{\text{node\_number}} (f_{i\text{min}} + (f_{i\text{max}} - f_{i\text{min}}) \cdot u \cdot \text{power\_cost} \cdot x_i), & \text{if } x_i > 0 \\
\sum_{i=1}^{\text{node\_number}} 0, & \text{if } x_i = 0 
\end{cases}$$  \hspace{1cm} (8)

$f_{i\text{min}}$ is defined as the lowest working energy consumption value of server $i$, in our scenario, the lowest working energy consumption means that no container is running in the server, and $f_{i\text{max}}$ is defined as the energy consumption value of server $i$ under full load. $\text{power\_cost}$ is defined as the cost of unit energy consumption.

The character $u$ in function (8) means the utilization of computing resources on server $i$ and it can be generated by:

$$u = \frac{x_i}{\text{node}(i) \cdot \text{capacities}}$$  \hspace{1cm} (9)
For the purpose of global optimization, the energy cost generated by all the hosts, the image pulling costs and the costs of data exchange between the management host and the execution host should be minimized. Take functions (5), (6), and (8) into consideration, the cost of container deployment function is introduced as:

$$f_{\text{cost}} = \sum_{i=1}^{\text{node number}} \alpha \cdot \text{data}_{\text{cost}} + \beta \cdot \text{energy}_{\text{cost}} + \gamma \cdot \text{im}_{\text{cost}}$$  \hspace{1cm} (10)$$

where the weighting factors $\alpha$, $\beta$, $\gamma$ are introduced to balance the three costs.

### 3.2. Problem formulation

Assume that in cluster environment, there is a set of execution nodes $\text{Enodes} = \{E1, \ldots, E_i, \ldots\}$, and the computing resources for execution node $i$ are fixed, defined as $\text{Enode}(i)$ capacities. Each time task scheduling occurs, subtasks in the task queue are loaded from the same management node to multiple execution nodes. Different task loading schemes on the execution nodes in the cluster are considered as different solutions.

The above work aims to dispatch the tasks loads from the management node to execution nodes to achieve the minimal costs of container deployment cost. This is reflected in the objective function (11):

$$f = \text{Min} \sum_{i=1}^{\text{node number}} \alpha \cdot \text{data}_{\text{cost}} + \beta \cdot \text{energy}_{\text{cost}} + \gamma \cdot \text{im}_{\text{cost}}$$  \hspace{1cm} (11)$$

Subject to the following constraints:

$$0 \leq x_i \leq \text{Enode}(i)_{\text{capacities}}$$  \hspace{1cm} (12)$$

$$\forall i \in \{1\ldots\text{node number}\}$$  \hspace{1cm} (13)$$

$$\forall j \in \{1\ldots N\}$$  \hspace{1cm} (14)$$

$$y$$ is binary.  \hspace{1cm} (15)$$

$$\sum_{i=1}^{\text{node number}} x_i = T$$  \hspace{1cm} (16)$$

Thereby, a PSO programming representation is introduced by the objective function (11) and the constrains (12), (13), (14), (15), and (16). Then the optimal solutions to the $\text{Gnode} = \{g_1, g_2, \ldots\}$ $\text{gnode number}$ can be reached. The complete algorithm procedure is presented in section IV.

### 4. CD-PSO Algorithm

In this section we present the improvement of the CD-PSO algorithm on the PSO algorithm and the specific implementation of the CD-PSO algorithm.

The role of the CD-PSO algorithm is to find the lowest cost task loading scheme during the containerized deployment of tasks. In the initialization phase of the algorithm, $N$ kinds of task load distribution schemes $X = \{X_1, X_2, \ldots, X_j\}$ are randomly generated under constraints. The vector sets of $X$ and $V$ are described as follows.

$$X = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1u} \\ x_{21} & x_{22} & \ldots & x_{2u} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1j} & x_{j2} & \ldots & x_{jv} \end{bmatrix}$$  \hspace{1cm} (17)$$
Among $X$, each line is a vector that represents a loading solution and also represents a particle position in the search space, $x_{ji}$ represents the task load of the $i$-th node in the $j$-th scheme. For vector set $V$, each line is a vector that represents a particle velocity, and the element $v_{ji}$ of $V_j$ is randomly generated between $(-1, 1)$ at initialization. Then Each particle is updated in an iteration until the maximum number of iterations $M$ is reached. Each time the particle completes the update in the iteration, the update of the optimal position must be considered. The updates of $P_{best_j}$ and $G_{best}$ satisfy the following formulae (19) and (20):

$$P_{best_j} = \begin{cases} P_{best}, & \text{if } f(P_{best}) \leq f(X_j) \\ X_j, & \text{if } f(P_{best}) > f(X_j) \end{cases}$$ (19)

$$G_{best} = \arg\min f(P_{best_i})$$ (20)

Where $f$ is a fitness function, which conforms to the above objective function (11) and the constrains (12), (13), (14), (15), and (16).

4.1. Algorithm improvement

In the PSO algorithm, particles are affected by the individual optimal position and the global optimal position during the iterative search process, and they will gather to the same position to different degrees. After reaching a certain number of iterations, a majority of particles will surround a certain space, this phenomenon is called falling into local optimum. Particles trapped in a local optimum will bring a lot of invalid searches. To prevent this, we proposed the CD-PSO algorithm to determine the local optimum in each iteration. If the particles are found to be trapped in the local optimum, an optimal guiding position, called OGP, will be selected to help the particles jump out of the local optimum.

4.1.1. determine the local optimum.

To get rid of the particles from the local optimality is basically to guide the particles to diverge from the vicinity of the aggregation position. In the process of divergence, the particles search for the spatial position that they passed. If the particles do not fall into a local optimum, and thus no guidance necessary, otherwise the particle search will be very uncertain, which will reduce the efficiency of algorithm optimization. Therefore, it is necessary to determine whether the particles are in a local optimum or not in time.

a. Hamming distance: In the information encoding, the alignment elements on the two strings are different, which is called Hamming distance. For example, the strings $s_1 = [01538]$, $s_2 = [04237]$, the elements in the first and fourth positions of $s_1$ and $s_2$ are the same, and the second, third, and fifth positions are different. The Hamming distance between the two strings is 3. For the same reason, it is stipulated that the absolute value of the difference between the alignment elements of the vectors $X_j$ and $G_{best}$ does not exceed 0.5, which is called the same alignment element, and the Hamming distance $HD_j$ of each particle position is calculated.

b. Calculate the average Hamming distance $HD_{ave}$ of the particle population in the current number of iterations. If formula (21) holds, it means that the current particle population has fallen into a local optimum, where $\delta$ is the coefficient and its value is set to 1.

$$HD_{ave} \leq \delta \cdot ln^{\text{nodeave}}$$ (21)

4.1.2. Select OGP. Particles have been updated from the initial initialization to the local optimal position. The global optimal position has undergone a series of updates. We put the updated global optimal position into the set $G$, $G = (G_1, G_2, ..., G_k)$, the $k$ represents the number of $G_{best}$ updates from
initialization to a local optimum. For example, $G_k$ represents the $G_{best}$ of the $k$-th update. The process is simulated, as shown in Figure 3.

\[ l_{n,n+1} = f(G_n) - f(G_{n+1}) \]  
\[ (22) \]

For any $G_n$ (except $G_i$, $G_k$) selected as the new $OGP$, the probability $f_{e_n}$ satisfies the following formula (23):

\[ f_{e_n} = l_{n-1,n} + l_{n,n+1} \]  
\[ (23) \]

We think that in the nearby space where the global optimal position appears multiple times, the spatial position that has not been searched by the particle is limited, while in the space where the distance between adjacent global optimal positions is relatively large, the particle has more spatial positions to search for. We use $f_{e_n}$ to indicate how much space has not been searched by particles near $G_n$. It is believed that the larger the $f_{e_n}$, the more positions in the space near $G_n$ the particles have not been searched for, and the greater the probability that $G_n$ will be selected.

4.2. Restrictions

The following restrictions are made in this algorithm.

4.2.1. Vector element update. When initialized, $x_{ji}$ is a random value within the range 0 to $E_{node(i)}\_capacities$. During the iteration process, it is necessary to frequently update $x_{ji}$. If $x_{ji}$ does not meet the formula (12) after updating, determine whether the value that brought the update to $x_{ji}$ is positive or negative. If the value is negative, subtract a random number in 0 to $x_{ji}$ from $x_{ji}$. When the value is positive, add a random number between 0 and $E_{node(i)}\_capacities - x_{ji}$ to $x_{ji}$.

4.2.2. $T$ keeps fixed. At initialization, T is assigned according to a random algorithm. As the algorithm iterates, the vector elements will change. To ensure that the load T remains unchanged after the vector is updated, we use the following steps:
a. Record the change value of the vector, defines as $\Delta$, which represents the difference between the sum of the vector element values before and after the update, and the number of non-zero elements, defined as $number$.

b. If $\Delta$ positive, subtract $\frac{\Delta}{number}$ from each non-zero element; if $\Delta$ negative, add the absolute value of $\frac{\Delta}{node number}$ to each element.

c. If the condition that the formula (12) is not satisfied after the element update occurs in step b, the update of the element is abandoned, and the $\frac{\Delta}{number}$ or $\frac{\Delta}{node number}$ of the part is retained.

d. Count the sum of all remaining $\frac{\Delta}{number}$ or $\frac{\Delta}{node number}$, defined as $\Delta'$, and determine whether the largest vector element minus $\Delta'$ or the smallest vector element plus the absolute value of $\Delta'$ satisfies formula (12).

e. If the formula (12) is satisfied, perform the corresponding operation; if it is not satisfied, it will be substituted into step b until it is satisfied.

4.3. Algorithm

**Input**: $N, M$, Task queue($tq$) 

**Output**: Task allocation scheme (Gnode)

1. Initialization $X = (X_1, X_2, ..., X_j, ..., X_N)$
2. While Number of iterations(iter) < $M$
3. For $j <= N$
4. $P_j = f(X_j)$
5. If $f(P_{best_j}) > P_j$
6. $P_{best_j} = X_j$
7. End
8. If $f(G_{best}) > f(P_{best_j})$
9. $G_{best} = P_{best_j}$
10. End
11. End
12. If $f(G_{node}) > f(G_{best})$
13. $G_{node} = G_{best}$
14. End
15. If $iter >= 3/4*M$
16. $HD_{sum} = 0$
17. For $j <= N$
18. Calculation $HD_j$
19. $HD_{sum} = HD_j + HD_{sum}$
20. End
21. $HD_{ave} = HD_{sum} / node number$
22. If $HD_{ave} <= \delta * ln(node number)$
23. Select OGP
24. $G_{best} = OGP$
25. End
26. End
27. Update $x_{ji}$
28. End
29. Return $G_{node}$
We found in experiments that the determine local optimum of operating was started when the number of iterations reached 4/3 of the maximum iteration, the execution time of the program was within a reasonable range, and the cost optimization performance was improved. Detailed experimental results are shown in section V.

5. Experiments
We conduct experiments to evaluate the performance of our proposed algorithm by comparing it with Binpack algorithm employed in the Docker Swarm [26] orchestration framework, Linear optimization algorithm (LP) proposed in reference [22], standard PSO, and TMPSO algorithm proposed by Shi et al [19]. We compare the cost of each algorithm during container deployment.

5.1. Experiment setup
For the sake of simplification and to expel other factors, the five scheduling algorithms (CD-PSO, Binpack, LP, PSO, and TMPSO) are implemented via MATLAB in a controlled experimental environment. The data of the test experiment comes from the CPU utilization of a set of tasks monitored by PlanetLab [27].

The experiment is performed in two ways: the first method selects a fixed number of tasks (10, 20, 30) and performs multiple experiments; The second method is to select 40, 60, 80, 100, 120, 140, 160, 180, 200, and 220 tasks for experiments. Each experiment is repeated multiple times, and the experimental result is the average of multiple experimental data. The computing resource capacity of each node is 100, node

\[
\text{node}_{\text{number}} = \text{ceil} \left( \frac{T}{100} \right) + c
\]

, ceil is an upward rounding function, c is the number of additional hosts.

The purpose is to ensure that there are sufficient computing resources in the cluster for allocation.

Without loss of generality, the weighting factors $\alpha$, $\beta$, $\gamma$ in the function (11) were set to 1. Following Morin et al. practice in 2011, the idle and fully utilized power values were fixed to 171 and 218 Watt, on a Dell PowerEdge 1950 server with 4 GB of RAM and two Intel Xeon 5148 2.33 GHz CPUs [25]. The powercost value is set to 1, the trafficcost is 1.2, the distance ($i$) that between the execution host $i$ and the management host is randomly generated between 2 and 6, and the host file pull cost pullcost is randomly generated within 1 to 3. The number of particles in the CD-PSO algorithm is $N = 60$, and the maximum number of iterations is $M = 300$.

5.2. Result
In the first method, 6 experiments were performed with 10, 20, and 30 tasks selected and 1 additional hosts. The experimental results are shown in Tables 1, 2, 3, 4, 5, and 6. The data in Tables 1, 3, and 5 show the deployment cost values of the allocation strategies implemented by several scheduling algorithms. The data in Tables 2, 4, and 6 represent the CD-PSO Optimize performance, which is the reduction percentage of the cost value of the CD-PSO algorithm compared with the cost value of other scheduling algorithms. From the experimental data, the cost of the CD-PSO algorithm is lower than other scheduling algorithms. The best experimental results can reach about 26% lower than the cost value of the Binpack algorithm, about 15% lower than the cost value of the LP algorithm, reducing the cost value of the PSO algorithm and the TMPSO algorithm more than 8%.

|        | Binpack | LP     | PSO    | TMPSO  | CD-PSO  |
|--------|---------|--------|--------|--------|---------|
| Round 1| 1763.75 | 1698.03| 1550.48| 1506.09| 1394.54 |
| Round 2| 1809.03 | 1769.38| 1458.05| 1430.02| 1353.05 |
| Round 3| 1910.05 | 1662.01| 1660.37| 1532.36| 1408.82 |
Table 2: CD-PSO reduces the percentage of other algorithm costs (10 tasks).

| %     | Binpack | LP   | PSO  | TMPSO |
|-------|---------|------|------|-------|
| Round 1 | 20.93   | 17.87| 10.05| 7.40  |
| Round 2 | 25.20   | 13.78| 7.20 | 5.38  |
| Round 3 | 26.24   | 15.23| 9.71 | 8.06  |

Table 3: Cost of 4 kinds of scheduling algorithms (20 tasks).

| Binpack | LP   | PSO  | TMPSO | CD-PSO |
|---------|------|------|-------|--------|
| Round 1 | 2953.46 | 2709.62 | 2596.79 | 2526.09 | 2383.06 |
| Round 2 | 3161.31 | 2914.37 | 2680.92 | 2620.02 | 2465.71 |
| Round 3 | 2756.73 | 2590.33 | 2560.37 | 2512.36 | 2269.66 |

Table 4: CD-PSO reduces the percentage of other algorithm costs (20 tasks).

| %     | Binpack | LP   | PSO  | TMPSO |
|-------|---------|------|------|-------|
| Round1 | 19.31   | 12.05| 8.23 | 5.66  |
| Round2 | 22.00   | 15.39| 8.03 | 5.89  |
| Round3 | 20.55   | 12.37| 11.31| 9.66  |

Table 5: Cost of 4 kinds of scheduling algorithms (30 tasks).

| Binpack | LP   | PSO  | TMPSO | CD-PSO |
|---------|------|------|-------|--------|
| Round 1 | 4659.84 | 4301.33 | 3993.50 | 3878.57 | 3658.97 |
| Round 2 | 4754.23 | 4434.46 | 4308.62 | 4257.14 | 3813.23 |
| Round 3 | 4816.98 | 4202.77 | 4121.86 | 4043.66 | 3712.94 |

Table 6: CD-PSO reduces the percentage of other algorithm costs (30 tasks).

| %     | Binpack | LP   | PSO  | TMPSO |
|-------|---------|------|------|-------|
| Round 1 | 21.48   | 14.93| 8.37 | 5.67  |
| Round 2 | 19.79   | 14.00| 11.49| 10.45 |
| Round 3 | 22.92   | 11.65| 9.92 | 8.18  |

In the second method, the number of tasks is between 40 and 120, the number of additional hosts is 2, the number of tasks is between 120 and 220, and the number of additional hosts is 3. The experimental results are shown in Figures 4 and 5. In different experimental environments, the optimization performance of the CD-PSO algorithm is basically stable. Among them, the cost value of the CD-PSO algorithm is more than 20% lower than the cost value of the Binpack algorithm, about 14% lower than the cost value of the LP algorithm, about 9% lower than the cost value of the PSO algorithm, and about 7% lower than the cost value of the TMPSO algorithm on average.
Figure 4. Cost of 4 scheduling algorithms.

Figure 5. CD-PSO reduces the percentage of Binpack, Random, LP cost values.

The three polylines in Figure 6 indicate the execution time of the PSO algorithm, TMPSO algorithm, and CD-PSO algorithm, which is from task submission to the end of optimization. It can be seen from the figure that the execution time of the TMPSO algorithm is the shortest, because the algorithm reduces the number of calculations and the generation of random numbers when the particle speed is updated. CD-PSO algorithm and PSO algorithm have a small difference in execution time when the number of tasks is 20 to 100. When the number of tasks reaches 100 or more, the difference in execution time is obvious. The reason is that the CD-PSO algorithm adds operations to determine the local optimum and select OGP on the basis of the PSO algorithm, which leads to an increase in the calculation amount of the algorithm. We set these operations to occur when the algorithm iteration reaches 4/3 of the maximum iteration. When the number of tasks is small, the amount of computation of the CD-PSO algorithm is only a little more than that of the PSO algorithm. Therefore, execution time of the CD-PSO algorithm is almost equal to the execution time of the PSO algorithm. However, As the number of tasks increases, the difference in execution time will become larger and larger. It can be seen that in small-scale task scheduling scenarios, the CD-PSO algorithm achieves the purpose of optimizing performance improvement with less time cost.
5.3. Analysis
We study the experimental results in different ways for a brief analysis. Binpack algorithm tends to load tasks onto several main execution hosts, the execution host is easily fully loaded, so the energy cost of the execution host is the highest, as the energy cost is container deployment costs accounting for the largest proportion. The LP algorithm, the PSO algorithm, the TMPSO algorithm, and the CD-PSO algorithm are all optimization algorithms. When scheduling, these algorithms will consider the various constraints of the hosts in the cluster to find the most appropriate task allocation scheme. However, the linear optimization algorithm is not as good as swarm intelligence optimization algorithm in swarm optimization applications. Due to the PSO algorithm often falls into a local optimum and cannot complete a full and efficient search. The CD-PSO algorithm has been improved to address this shortcoming to improve the ability of particles search in the overall solution space, the TMPSO algorithm is designed to focus on finding the optimization results in a short time, so it is difficult to guarantee stability and optimization performance. Therefore, the experimental results show that the cost of the CD-PSO algorithm is lower than other scheduling algorithms, and in a small-scale experimental environment, the time cost of the CD-PSO is also acceptable.

6. Conclusion
In this paper, we propose a particle swarm optimization algorithm for container deployment (CD-PSO) and implement it on the Matlab simulation platform. Through the data obtained from comparing experiments of CD-PSO algorithm with Binpack algorithm, LP algorithm, PSO algorithm and TMPSO algorithm, it is demonstrated that the CD-PSO algorithm can find the optimal loading solution during the container deployment process to reduce the deployment cost.

In the future, we will consider the diversity of required resources within the scope of our research and apply the CD-PSO algorithm to the Swarm or Apache mesos scheduling module.

Acknowledgment
We are grateful to Zhang D for providing the source code, and Anton Beloglazov for providing the dataset. This work was supported by National Science Support Program of China (No.2015BAH54F01).

References
[1] Liu, S. Y., Li, Q., Li, B. (2015) Research on isolation of container based on Docker technology. Computer Engineering & Software, 36: 110-113.
[2] Pahl, C., Brogi, A., Soldani, J. and Jamshidi, P. (2019) Cloud container technologies: a state-of-the-art review. IEEE Transactions on Cloud Computing, 7: 677-692.
[3] Singh, S., Singh, N., (2016) Containers & Docker: emerging roles & future of cloud technology. In 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATcT). Bangalore. pp. 804-807.

[4] Tao, Y., Wang, X., Xu, X. and Chen, Y. (2017) Dynamic resource allocation algorithm for container-based service computing. In 2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS). Bangkok. pp. 61-67.

[5] Kaewkasi, C., Chuenmuneewong, K. (2017) Improvement of container scheduling for Docker using ant colony optimization. In 2017 9th International Conference on Knowledge and Smart Technology (KST). Chonburi. pp. 254-259.

[6] Bhimani, J., Yang, Z. Y., Mi, N. F., et al. (2018) Docker container scheduler for I/O intensive applications running on NVMe SSDs. IEEE Transactions on Multi-Scale Computing Systems, 4: 313-326.

[7] Wu, J., Yang, T. (2018) Dynamic CPU allocation for Docker containerized mixed-criticality real-time systems. In 2018 IEEE International Conference on Applied System Invention (ICASI). Chiba. pp. 279-282.

[8] Zhang, J., Ren, R., Huang, C., Fei, X., Qun, W. and Cai, H. (2018) Service dependency based dynamic load balancing algorithm for container clusters. In 2018 IEEE 15th International Conference on e-Business Engineering (ICEBE). Xi’an. pp. 70-77.

[9] Cérin, C., Menouer, T., Saad, W. and Abdallah, W. B. (2017) A new Docker Swarm scheduling strategy. In 2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2). Kanazawa. pp. 112-117.

[10] Canosa, R. et al., (2018) Energy consumption and quality of service optimization in containerized cloud computing. In 2018 Ivannikov Ispras Open Conference (ISPRAS). Moscow. pp. 47-55.

[11] Cheng, Y., Lin, C., Liu, P. and Wu, J. (2017) High resource utilization auto-scaling algorithms for heterogeneous container configurations. In 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS). Shenzhen. pp. 143-150.

[12] Wu, S., Niu, C., Rao, J., Jin, H. and Dai, X. (2017) Container-Based cloud platform for mobile computation offloading. In 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Orlando. pp. 123-132.

[13] Greenberg, A., Hamilton, J., Maltz, D. A. and Patel, P. (2008) The cost of a cloud: research problems in data center networks. ACM SIGCOMM Computer Communication Review, 39: 68-73.

[14] Kurpicz, M., Orgerie, A. and Sobe, A. (2016) How much does a VM cost? Energy-proportional accounting in VM-based environments. In 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP). Heraklion. pp. 651-658.

[15] Dabbagh, M., Hamdaoui, B., Guizani, M. and Rayes, A. (2018) An energy-efficient VM prediction and migration framework for overcommitted clouds. IEEE Transactions on Cloud Computing, 6: 955-966.

[16] Sonkar, S. K. and Kharat, M. U. (2016) A review on resource allocation and VM scheduling techniques and a model for efficient resource management in cloud computing environment. In 2016 International Conference on ICT in Business Industry & Government (ICTBIG). Indore. pp. 1-7.

[17] Tian, W., He, M., Guo, W., et al. (2018) On minimizing total energy consumption in the scheduling of virtual machine reservations. Journal of Network and Computer Applications, 113: 64-74.

[18] Alrakhami, M., Gumaei, A., Alsahtli, M., Hassan, M. M. and Fortino, G. (2019) A lightweight and cost effective edge intelligence architecture based on containerization technology. World Wide Web,1:76-84.

[19] Shi, T., Ma, H. and Chen, G. (2018) Energy-aware container consolidation based on PSO in cloud data centers. In 2018 IEEE Congress on Evolutionary Computation (CEC). Rio de Janeiro. pp. 1-8.
[20] Hanafy, W. A., Mohamed, A. E. and Salem, S. A. (2017) Novel selection policies for container-based cloud deployment models. In 2017 13th International Computer Engineering Conference (ICENCO). Cairo. pp. 237-242.

[21] Guan, X., Wan, X., Choi, B., Song, S. and Zhu, J. (2017) Application oriented dynamic resource allocation for data centers using Docker containers. IEEE Communications Letters, 21: 504-507.

[22] Zhang, D., Yan, B., Feng, Z., Zhang, C. and Wang, Y. (2017) Container oriented job scheduling using linear programming model. In 2017 3rd International Conference on Information Management (ICIM). Chengdu. pp. 174-180.

[23] Kennedy, J. and Eberhart, R. (1995) Particle swarm optimization. In Proceedings of ICNN95 - International Conference on Neural Networks. Perth. pp. 1942-1948.

[24] Brogi, A., Neri, D. and Soldani, J. (2017) DockerFinder: multi-attribute search of Docker Images. In 2017 IEEE International Conference on Cloud Engineering (IC2E). Vancouver. pp. 273-278.

[25] Feller, E., Rilling, L., and Morin, C. (2011) Energy-aware ant colony based workload placement in clouds. In 2011 IEEE/ACM 12th International Conference on Grid Computing. Lyon. pp. 26-33.

[26] Ismail, B. I., Goortani, E. M., Karim, M. B. A., Tat, W. M. and Hoe, O. H. (2015) Evaluation of Docker as edge computing platform. In 2015 IEEE Conference on Open Systems (ICOS). Bandar Melaka. pp. 130-135.

[27] Park, K.S., and Vivek, S. Pai. (2006) CoMon: a mostly-scalable monitoring system for PlanetLab. Acm Sigops Operating Systems Review, 40: 65-74.

[28] Beloglazov, A., Buyya, R. (2012) Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency and Computation: Practice and Experience (CCPE), 24: 1397-1420.