IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received August 9, 2019, accepted August 20, 2019, date of publication August 26, 2019, date of current version September 10, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2937553

A Genetic Algorithm-Based Energy-Efficient Container Placement Strategy in CaaS

RONG ZHANG[1,2], YAXING CHEN[1,2], BO DONG[3,4], FENG TIAN[1,2,4], AND QINGHUA ZHENG[1,2]

[1]Shaanxi Province Key Laboratory of Satellite and Terrestrial Network Technology Research and Development, Xi'an Jiaotong University, Xi'an 710049, China
[2]School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China
[3]School of Continuing Education, Xi'an Jiaotong University, Xi'an 710049, China
[4]National Engineering Laboratory for Big Data Analytics, Xi'an Jiaotong University, Xi'an 710049, China

Corresponding author: Feng Tian (fengtian@mail.xjtu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB1000303, in part by the Innovative Research Group of the National Natural Science Foundation of China under Grant 61721002, in part by the Innovation Research Team of Ministry of Education under Grant IRT_17R86, in part by the National Natural Science Foundation of China under Grant 61877048 and Grant 61532015, in part by the Project of China Knowledge Centre for Engineering Science and Technology, in part by the Project of Chinese Academy of Engineering "The Online and Offline Mixed Educational Service System for 'The Belt and Road' Training in MOOC China," and in part by the Natural Science Basic Research Plan in Shaanxi Province of China under Grant 2019JM-458.

ABSTRACT Container placement (CP) is a nontrivial problem in Container as a Service (CaaS). Many works in the literature solve it by using linear server energy-consumption models. However, the solutions of using a linear model makes different CPs indistinguishable with regard to energy consumption in a homogeneous host environment that has a same amount of active hosts. As such, these solutions are energy inefficient. In this paper, we demonstrate that an energy-saving gain can be achieved by optimizing the placement of containers under a nonlinear energy consumption model. Specifically, we leverage a strategy based on genetic algorithm (GA) to search the optimal solution. Unfortunately, the conventional GA incurs performance degradation when the virtual machine (VM) resource utilization is high. In order to solve this problem, we propose an improved genetic algorithm called IGA for efficiently searching the optimal CP solution by introducing two different exchange mutation operations and constructing a function as the control parameter to selectively control the usage of the two operations. Extensive experiments are carried out under different settings, and their results show that our strategy is better than the existing CP strategies, i.e., spread and binpack, on energy efficiency target. In addition, the introduced IGA is experimentally proved to be more effective compared with the First Fit, Particle Swarm Optimization (PSO) algorithm and conventional GA. Moreover, the results validate that our proposed strategy can search new CP solutions with better fitness and alleviate the performance degradation caused by the conventional GA when the VM resource utilization is high.

INDEX TERMS CaaS, container placement, genetic algorithm, exchange mutation operation.

I. INTRODUCTION

Container, e.g., Docker [1]–[3], is an operating system-level virtualization technology, which can be deployed either on virtual machines (VMs) or on physical machines (PMs). It is primarily used for providing an isolated environment for application execution. Compared with a VM that needs to occupy the entire operating system (OS) resources, a container can share the same OS kernel with others. Thus, a container is considered to be lightweight with less resource consumption and low bootstrap time. Nowadays, Container as a Service (CaaS) [4], [5], e.g., Fargate [6] by Amazon and Kubunetes [7] by Google, is widely adopted in the cloud to provide more service options for end users. It is still a challenge to find a feasible placement with an optimum energy consumption under the widely adopted CaaS architecture [4] where containers are restricted to be only placed on VMs. Basically, the CP refers to assigning containers to suitable computational nodes to achieve an expected goal under specific resource constraints.

The associate editor coordinating the review of this article and approving it for publication was Jiankang Zhang.

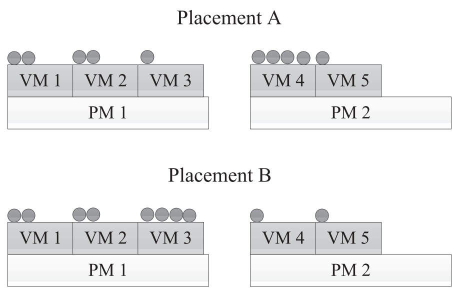


FIGURE 1. **Two different container placements.**

As described in the Docker Docs [8], three rapid CP strategies [9] (spread, binpack and random) are currently supported by the Docker Swarm scheduler, but they do not take into account the energy efficiency. References [10] and [11] achieve the energy-savings goal by introducing a mechanism of container consolidation. However, they exploit a linear server energy-consumption model in their work, which is not accurate enough for new servers that have new hardware and software upgrades in some cases [12]–[15]. In fact, there are various server energy consumption models [16], and different models provide different results with respect to total energy costs. In addition, when adopting a linear server energy consumption model in a homogeneous host environment, the energy consumption of each server is a linear function of its resource utilization. Therefore, the total server energy consumption, which is the sum of all servers' energy consumption, would be a linear function of the total resource utilization. As a result, different CPs do not affect the total server energy consumption since the total resource utilization is always the same in homogeneous host environment with the same amount of actives hosts. Next, we craft a concrete example to illustrate the difference between linear and nonlinear models. As shown in Fig. 1, where the circles, small grey rectangles and big white rectangles represent containers, VMs and PMs, respectively, and the normalized CPU specification of each container is 0.0714. We have two CPs A and B. The total workload for the servers (PMs) in both CPs is 10 containers. In A, 5 containers are placed in each PM; while, in B, 8 containers are placed in PM 1 and two containers are placed in PM 2. Suppose that we adopt a linear energy consumption model in [17], i.e., $LP_i(u_i) = 175+75u_i$, where LP_i and u_i represent the energy consumption and CPU utilization of PM i, respectively. The total energy consumption for CP A and CP B are both 403.55. When we consider a nonlinear energy-consumption model as in [18], i.e., $NLP_i(u_i) = 155 + 345u_i - 359u_i^2 + 144u_i^3$, the energy consumption for CP A is 477.9254 and that for CP B is 459.1345. Obviously, CP B is better than CP A (approximately 3.93% power saving). The two examples verify that different CPs do not affect the total server energy consumption under the linear model, but that is not true when considering the nonlinear model. We conclude that utilizing different energy consumption models can affect the energy efficiency of a CP and further energy savings can be achieved by optimizing the CP under a nonlinear model.

However, since the CP problem is viewed as a combinatorial optimization problem, it is impractical to make a complete enumeration of all possible solutions to find the best one. Heuristic algorithms are well suited for solving such problems. In this paper, we opt for a heuristic algorithm, e.g., the GA, to search a good quality solution. GA [19]–[24] is a population-based searching method, which encodes the possible solutions as chromosomes, also called individuals, to form a population. The algorithm simulates the evolutionary mechanism of organisms by means of selection, crossover and mutation, and updates a population for evolution in each iteration until the termination condition is reached. We notice that the mutation operation in a conventional GA randomly changes an exist chromosome on one or more genes' values to get a new chromosome. In other words, one or more containers in the CP solution will be relocated to other VMs. Unfortunately, this can easily leads to a situation where the number of containers in a VM increases while the number decreases in another VM. Such kind of resource deviation often makes the mutation operation too "intense" when the VM resource utilization is high. Thus, it easily triggers the elimination of new individuals due to the worse fitness or exceeding the resource constraints, causing performance degradation.

The following example illustrates the performance degradation problem under the condition of high VM resource utilization. Provided that there exist two PMs and each PM only hosts one VM. We assume that a placement with the maximum VM resource utilization can get the best fitness under a certain energy consumption model. As shown in Fig. 2, provided that (a) is the best CP solution and (b) is the current CP, (c) is an example of transferring one container from a VM to the other, which simulates a conventional GA mutation at the current CP with a one-gene mutation operation, (d) is an example of the gene exchange plan based on an exchange mutation operation at the current CP. (e) is the chromosome code of (c). (f) is the chromosome code of (d). The details of the chromosomes encoding can be seen in Section III.A. It can be easy to find that the resource utilization of VM 1 is high in the current CP but is the maximum one in the best CP. When a conventional GA performs one-gene mutation operation at the current CP, each container in VM 2 cannot be relocated to VM 1 due to exceeding the resource constraints and each container in VM 1 also cannot be relocated to VM 2 due to its worse fitness. Therefore, such a mutation operation incurs performance degradation and makes it difficult to obtain a new chromosome with better fitness. In order to solve this problem, we introduce a special mutation strategy called exchange mutation operation, the core idea of which is changing the values on different gene positions but keeping all the gene values and their numbers the same before and after the change. Take Fig. 2(f) as an example, the value on the gene position 5 is changed from 2 to 1, and the value on the gene position 8 is changed from 1 to 2, all the gene values and their numbers are keeping the same before and after the change. This change is like a value exchange between two different gene positions. Furthermore, in a view

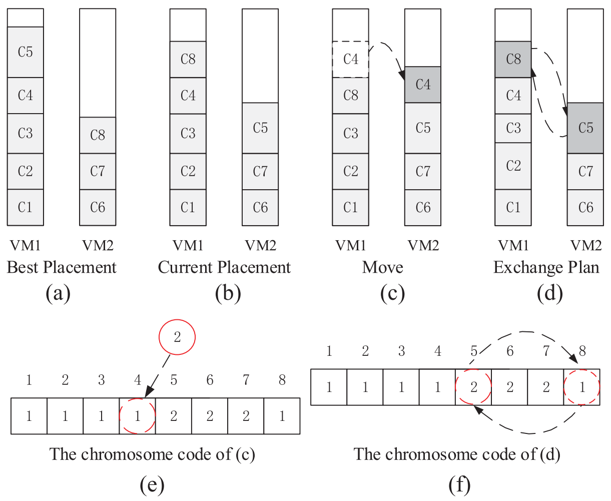


FIGURE 2. An example of the best placement, current placement, two different mutation methods and their chromosome codes.

of container placement, the exchange mutation operation means exchanging the containers among some VMs but does not change the amount of containers on each VM before and after the exchange. Take Fig. 2(d) as an example. The two VM nodes exchange one container. In this exchange, container C8 is transferred from VM 1 to VM 2 and C5 is transferred from VM 2 to VM 1. However, the amount of containers on each VM does no changed before and after the exchange. Notably, after this exchange mutation operation (i.e. (d)), the current CP (i.e. (b)) becomes the best CP (i.e. (a)) and achieves the optimal fitness. Therefore, compared with the "intense" adjustment, this exchange mutation operation, which plays the role of the "lightweight" adjustment, can find more CPs that are superior to the current CP. It is more conducive to the retention of the newly generated placement.

In this paper, we study the energy efficient CP optimization problem under a nonlinear energy consumption model. Specifically, we establish an energy-efficient optimization target model and propose a strategy based on an improved GA, called IGA to search an optimal CP solution. Two kinds of exchange mutation operations along with a control parameter φ are proposed in IGA to improve the mutation operation in the conventional GA. More concretely, both of the two exchange mutation operations keep the numbers of containers in the VMs constant, but one helps local search optimization while the others helps jumping out of local optimal. The control parameter, defined as a function of the number of searching iterations, chooses which exchange mutation operation should be performed during the whole solution search process. Experiments simulate CPs in small, medium and large scales under scenarios where VMs are non-uniformly and uniformly distributed across PMs. The experimental results show that our method can effectively reduce the total server energy consumption under a nonlinear model compared with the existing strategies. The main contributions are summarized as follows.

TABLE 1. Notations.

Symbol	Description
$cpu_{PM_i}^{total}$	The total CPU resource of the PM_i
$mem_{PM_i}^{total}$	The total memory resource of the PM_i
$cpu_{PM_i}^{used}$	The CPU utilization of PM_i
$mem_{PM_i}^{used}$	The memory utilization of PM_i
$cpu_{VM_j}^{total}$	The total CPU resource of VM_j
$mem_{VM_j}^{total}$	The total memory resource of VM_j
$cpu_{VM_j}^{ini}$	The CPU utilization of VM_j in the initial state
$mem_{VM_j}^{ini}$	The memory utilization of VM_j in the initial state
$cpu_{VM_j}^{used}$	The CPU utilization of VM_j
$mem_{VM_j}^{used}$	The memory utilization of VM_j
cpu_{con}^{k}	The CPU resource requirement for the new Con_k
mem_{con}^{k}	The memory resource requirement for the new Con_k
$\delta_j^k, \tau_i^j, x(i)$	$0-1$ variables

- We illustrate that energy-saving gain can be achieved by optimizing the placement of containers under a nonlinear energy consumption model.
- We formalize the energy-efficient CP problem in CaaS and establish an optimization model for it.
- We present an improved genetic algorithm called IGA to solve the performance degradation caused by the traditional GA, which makes use of two exchange mutation operations and a control parameter.
- We perform extensive experiments and the results show that our strategy achieves a better energy-saving goal, compared with the two existing Docker Swarm strategies. Besides, the introduced IGA is experimentally proved to be more effective than the First Fit, PSO and conventional GA.

The rest of this paper is organized as follows. Section 2 introduces the objective model presented in this study. Our method based on an improved algorithm (IGA) is described and evaluated in Sections 3 and 4, respectively. Section 5 discusses the related works. At last, we summarize the whole paper and our future work.

II. PROBLEM FORMULATION

In this section, we introduce the objective model and corresponding constraints for the energy-efficient CP problem that is to be solved. For simplicity, we make the following assumptions.

We adopt the CaaS architecture in which containers are placed on VMs while VMs are hosted by PMs. We only consider the situation where the existing VMs can meet the resource requirements for all new containers. We only focus on two kinds of resources, i.e., CPU and memory. All PMs and VMs have the same configurations and/or specifications.

Table 1 shows the main notations used in this paper.

A. PROBLEM DESCRIPTION

The CP issue in this paper refers to assigning new containers to suitable VM nodes that have limits on multidimensional resources, and it can be described as a packing problem. Assume that we have $Cnum$ new containers with different resource requirements to be placed, $VMnum$ VMs with the same specification and $PMnum$ PMs with a same configuration. Initially, a random number of containers are placed on each VM and the VMs are randomly distributed across PMs. The question is how to solve this packing problem and achieve the goal of minimizing the total server energy consumption. To be specific, two sub-questions need to be solved, i.e., whether the potential target VMs have enough available resources for the current placement and whether such a placement will contribute to the energy-savings target.

B. RESOURCE NOTATIONS

The set of PMs is denoted as $PM_{set} = \{PM_i | i = 1, \cdots, PMnum\}$. We use a tuple $(cpu_{PM_i}^{total}, mem_{PM_i}^{total})$ to represent the resource specification of each PM_i.

The set of VMs is denoted as $VM_{set} = \{VM_j | j = 1, \cdots, VMnum\}$. We use a tuple $(cpu_{VM_j}^{total}, mem_{VM_j}^{total})$ to represent the resource specification of each VM_j.

The set of containers is denoted as $Con_{set} = \{Con_k | k = 1, \cdots, Cnum\}$. We use a tuple $(cpu_{Con}^{k}, mem_{Con}^{k})$ to represent the resource specification of each Con_k.

C. PLACEMENT RELATION

We formally define the placement relationship among the containers, VMs and PMs. The relationship between Con_k and VM_j is formally expressed as follows:

$$\delta_j^k = \begin{cases} 1, & \textit{if } Con_k \textit{ is placed on } VM_j \\ 0, & \textit{others.} \end{cases} \tag{1}$$

The relationship between VM_j and PM_i is formally expressed as follows:

$$\tau_i^j = \begin{cases} 1, & \textit{if } VM_j \textit{ is placed on } PM_i \\ 0, & \textit{others.} \end{cases} \tag{2}$$

Therefore, the relationship between Con_k and PM_i can be expressed as $\delta_j^k \tau_i^j$, where, $k = 1, \cdots, Cnum$; $j = 1, \cdots, VMnum$; and $i = 1, \cdots, PMnum$.

D. RESOURCE UTILIZATION

For each VM, its workload consists of all containers running on it, which include those that are already placed at the initial state and the new ones. Thus, the respective CPU and memory utilization $cpu_{VM_j}^{used}$ and $mem_{VM_j}^{used}$ of a VM can be formalized as

$$cpu_{VM_j}^{used} = \sum_{k=1}^{Cnum} \delta_j^k cpu_{con}^k + cpu_{VM_j}^{ini}; \tag{3}$$

$$mem_{VM_j}^{used} = \sum_{k=1}^{Cnum} \delta_j^k mem_{con}^k + mem_{VM_j}^{ini}. \tag{4}$$

While VMs run on PMs, the real workload of a PM includes all the containers of the VMs that are hosted by the PM. Similarly, it has two parts, i.e., the containers that are initially hosted and the newly assigned ones. Formally,

$$cpu_{PM_i}^{used} = \sum\nolimits_{k=1}^{Cnum} \sum\nolimits_{j=1}^{VMnum} \delta_j^k \tau_i^j cpu_{con}^k + cpu_{PM_i}^{ini}, \tag{5}$$

$$cpu_{PM_i}^{ini} = \sum\nolimits_{j=1}^{VMnum} \tau_i^j cpu_{VM_j}^{ini}; \tag{6}$$

$$\begin{aligned} mem_{PM_i}^{used} = {} & \sum\nolimits_{k=1}^{Cnum} \sum\nolimits_{j=1}^{VMnum} \delta_j^k \tau_i^j mem_{con}^k \\ & + mem_{PM_i}^{ini}, \end{aligned} \tag{7}$$

$$mem_{PM_i}^{ini} = \sum\nolimits_{j=1}^{VMnum} \tau_i^j mem_{VM_j}^{ini}. \tag{8}$$

The first parts of formulas (5) and (7) represent the resources that are required for the new containers, and the second part represents the resources that are occupied by the containers that are initially hosted.

E. SERVER ENERGY CONSUMPTION MODEL

We adopt a nonlinear server energy consumption model in our work. Specifically, we use a polynomial model that was proposed by [25], which precisely reflects the influence of both CPU and memory resource utilization on the server's energy consumption. Moreover, when running the computer performance Benchmarks on a real server, the energy-consumption function is fitted and achieves a prediction accuracy more than 95%. The energy consumption of a single server $power(i)$ in this work is calculated as

$$\begin{aligned} power(i) = {} & (375.8550 \quad -401.0088 \quad 164.4327) \\ & \times \begin{pmatrix} cpu_{PM_i}^{used} \\ (cpu_{PM_i}^{used})^2 \\ (cpu_{PM_i}^{used})^3 \end{pmatrix} \\ & +(-30.6192 \quad 41.8946 \quad -19.8122) \\ & \times \begin{pmatrix} mem_{PM_i}^{used} \\ (mem_{PM_i}^{used})^2 \\ (mem_{PM_i}^{used})^3 \end{pmatrix} \\ & +155.0537. \end{aligned} \tag{9}$$

F. ENERGY-EFFICIENT CP MODEL

We use the following optimization model to formalize the energy-efficient CP problem.

$$Minimize\ Power = \sum\nolimits_{i=1}^{PMnum} x(i) power(i), \tag{10}$$

which is subject to

$$cpu_{VM_j}^{used} \leq cpu_{VM_j}^{total}, \quad j = 1, \ldots, VMnum, \tag{11}$$

$$mem_{VM_j}^{used} \leq mem_{VM_j}^{total}, \quad j = 1, \ldots, VMnum, \tag{12}$$

$$x(i) = \begin{cases} 0, & \textit{if } cpu_{PM_i}^{used} = mem_{PM_i}^{used} = 0 \\ 1, & \textit{others.} \end{cases} \tag{13}$$

When $x(i) = 1$, it means the PM_i is active, otherwise inactive. Notably, the two type of constraints (11) and (12) guarantee that the resource requirement of the new containers assigned to a VM do not exceed its resource limitation.

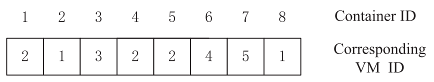


FIGURE 3. A chromosome encoding example.

III. OUR METHOD

As mentioned above, the mutation operation in the conventional GA is hard to get a new chromosome that has better fitness when the VM resource utilization is high, and it will result in a performance degradation. Therefore, this paper proposes an improved GA called the IGA. The improvements primarily have two aspects. First, it optimizes the mutation operation help to get a new chromosome that has better fitness by introducing two exchange mutation operations and offering a control parameter φ to affect the selection of the two operations in terms of a random probability. Second, it adds a check and correction operation to make sure that the population is safe. It examines each generated placement solution. If it cannot satisfy the resource constraints, i.e., it is not a feasible solution, the algorithm derives a feasible one by correcting the solution, and then IGA is applied to the container scheduling scenario. The main steps of the IGA are shown below.

A. CHROMOSOME ENCODING AND POPULATION INITIALIZATION

1) CHROMOSOME ENCODING

Each possible CP solution in the CP problem represents a relationship between containers and VMs. Therefore, we first encode this placement and call it a chromosome. Because the numbers of containers and VMs are *Cnum* and *VMnum*, respectively, a chromosome is essentially a vector with *Cnum* elements and each element is a number from 1 to *VMnum*. The order of elements in the vector represents the corresponding IDs of the containers, and the value of each element represents the ID of a VM. As shown in Fig. 3, the chromosome encoding means that Con_1 is placed in VM_2, Con_2 is placed in VM_1, and container Con_3 is placed in VM_3.

2) POPULATION INITIALIZATION

A population is made up of *popsize* individuals, and each individual represents a possible CP solution. To ensure that the generated CP solutions are feasible, i.e., the resource requirements of each container to be placed can be satisfied, We opt to use the First Fit algorithm [26], [27] to randomly generate an initial population. The process of population initialization is as follows. It successively selects an available VM for each new container until all containers are placed. The pseudocode of the population initialization is shown in Algorithm 1. Notably, the pseudocode of the First Fit algorithm is shown in Algorithm 1 from line 4 to line 11.

B. SELECTION AND CROSSOVER OPERATIONS

Simulating the crossover and mutation in biological evolution, the IGA relies on a population evolutionary strategy to search the optimal solution. Notably, new individuals

Algorithm 1 Population Initialization

```
Input:CList,VMList,popsize; Resources.
Output:Population
1  for index=1:popsize
2    Resources0 ← Resources
3    newVMList ← Prearrange the sequence of VMList
4    for =1 to length(Clist) do
5      for $j$ = 1:length(newVMList) do
6        if VM has enough resource for the placement
         of Con do
7          Population(index, $i$)= $j$;
8          Resources0 update;
9          break
10       end if
11     end for
12   end for
13 end for
14 return Population
```

(children) are created by mating individuals (parents) from the last generation.

1) SELECTION OPERATION

In order to generate each new individual, we first need to select its parents for mating. Here we adopt a random selection strategy, which selects two different individuals (denoted by Chromosomes A and B) at random. The random selection strategy fairly selects two parents at random, and thus can obtain various combinations of parents. In this way, it will reserve some potential good parents that have good genetic segments but bad fitness, and can effectively avoid the premature convergence problem caused by over-copying a high fitness parent.

2) CROSSOVER OPERATION

The crossover operation is a mating mechanism, which generates a new individual by combining two chromosome pieces from their parents. We opt for one-point crossover operation in this paper. More concretely, a random number r is generated as a crossover site, the value of which is set between 1 and the element number of a chromosome subtracting 1. Then, a new chromosome is created by gluing the genome before that crossover site of a chromosome with the genome after that crossover site of another chromosome, as shown in Fig. 4. Such an operation simulates the evolutionary behavior of preserving the excellent genomes from different individual and reserves some mapping relationship between containers and virtual machines in different CPs. On the other hand, the conventional GA provides a crossover probability threshold *pcorssover* to determine whether to carry out the crossover operation or just directly copy the ith individual in the previous generation population to the new one as its ith individual. In a nutshell, it compares a randomly generated value ranging from 0 to 1 with the *pcorssover*.

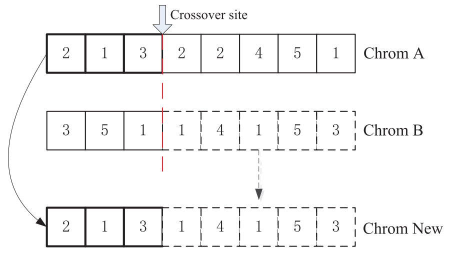


FIGURE 4. An example of the crossover operation.

If and only if the random value is less than the *pcorssover*, the crossover operation will be performed. We remark that a bigger crossover probability threshold can result in a bigger probability to perform the crossover operation, more new individuals, as well as faster searching speed of the algorithm. To obtain as many new individuals as possible, we set the crossover probability threshold *pcorssover* as 1 in the IGA. Another reason to do so is that the CP problem is a container-based combinatorial optimization problem, and an optimal solution may have many different matches between containers and VMs (i.e., chromosome code). Therefore, the chromosome code of the optimal CP solution may not be unique, and the more new individuals are generated, the better the optimal CP solution will be searched.

C. EXCHANGE MUTATION OPERATION

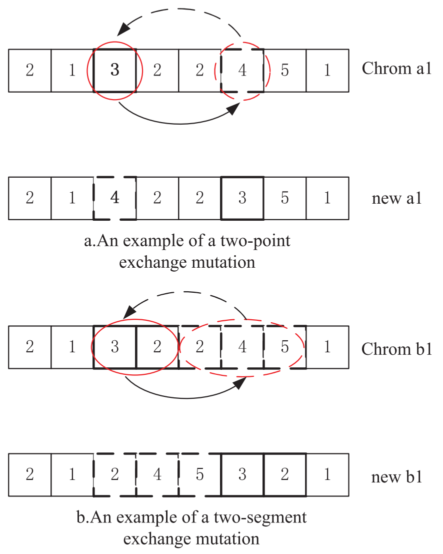
The mutation operator operates on a single chromosome; it changes the value(s) in one or more gene positions of the chromosome, which is an effective way to enhance the diversity of the population and helps the algorithm itself to jump out of the local optimum. However, as discussed in Section I, the mutation operation in the conventional GA often changes the number of containers in a VM. When the resource utilization of VMs is high, such changes will make it easy to eliminate a newly generated individual and hard to enhance the diversity. A potential solution is to just exchange containers between VMs, such that the number of containers in each VM keeps constant. To achieve such a goal, we introduce two exchange mutation operations, i.e., the two-point exchange mutation operation (TPEMO) and the two-segment exchange mutation operation (TSEMO). Our experimental tests in Section IV also proves that this is a much better way.

1) TPEMO

As Fig. 5(a) shows, the TPEMO randomly selects two different positions from a chromosome for exchange, which simulates a lightweight adjustment (mutation) for two containers in a placement. The newly derived individual has little difference from its origin. This helps to find the optimal solution near the native individual.

2) TSEMO

By comparison, TSEMO randomly selects two adjacent genome segments for the position exchange (see Fig. 5(b)). It simulates an adjustment to a portion of containers, which is a stronger level adjustment than TPEMO. Generally,



2 1 3 2 2 4 5 1 Chrom a1
2 1 4 2 2 3 5 1 new a1
a.An example of a two-point exchange mutation
2 1 3 2 2 4 5 1 Chrom b1
2 1 2 4 5 3 2 1 new b1
b.An example of a two-segment exchange mutation

FIGURE 5. Two kinds of mutation operation.

the newly derived individual has a significant distinction from the original one. This makes a key contribution to jumping out of a local optimum.

The two types of mutation exchange operations have their own advantages, and we need a strategy to control the number of individuals performing the two mutation exchange operations to guide the evolution of the population.

3) CONTROL PARAMETER

The conventional GA exploits a mutation probability threshold *pmutation* to determine whether or not to perform the mutation operation. In a nutshell, it compares a randomly generated value, ranging from 0 to 1, with the *pmutation*. If the random value is less than *pmutation*, the mutation operation will be performed; otherwise, it will not. If the probability is too small, the number of individuals performing the mutation operation will be small, which makes it hard to generate new individuals. However, as analyzed in Section III.*B*, the more new individuals that we have, the better the search performance for the optimal solution will be. Thus, we perform the mutation operation on each individual using a different method than the conventional GA. With the increase in the iteration number t, the population gradually evolves towards an optimal solution. To maintain the diversity of the population and prevent falling into the local optimum, the proportion of individuals that execute the two kinds of variations in the population should also vary as the iteration number t changes. We define a control parameter φ to act as the threshold of the mutation probability to regulate the proportion of individuals executing the two types of mutation operations in the population. Notably, when the control parameter φ is small, the proportion of individuals performing a specific type of mutation operations is small, and vice versa.

More concretely, at the early stage of search, i.e., when the number of iteration meets $t < \frac{MaxI}{2}$, in order not to destroy the good genomes that are preserved in the crossover operation, that is, the good mapping relationship between the containers and VMs, the mutation should not be too "intense" so that the individuals performing "lightweight" mutations should occupy a relatively high proportion in the population. When the search arrives at the middle period, i.e., $t \rightarrow \frac{MaxI}{2}$, individuals with sufficient crossover and mutation operations tend to gather around the local optimum. However, lightweight mutation has low diversity at this moment due to its weak ability to jump out of the local optimum. Therefore, it is necessary to gradually increase the number of individuals carrying out the TSEMO to make them jump out of the local optimum. At the final stage of search, i.e., $t \rightarrow MaxI$, most individuals have approached the global optimum and the TSEMO mutation will become too intense, which does not contribute to the local search. Therefore, we adopt the TPEMO once again for most individuals. To sum up, in the whole iterative search process, the proportion of individuals carrying out TPEMO should decrease slowly from the highest point and then increase sharply after reaching the lowest point. Since the control parameter φ and the proportion of individuals performing a given type of mutation operation are synchronous, the control parameter φ also has the above characteristics. Thus, we should find a function of t that corresponds to these characteristics as the control parameter φ.

There exist many off-the-shelf functions that satisfy the requirement, i.e., the value of the function slowly decreases from the highest point and then rises sharply after reaching the lowest point. An alternative way is function construction. We can easily construct a desired piecewise function with existing straightforward functional primitives. However, it is very difficult to compare and find the optimal function since it has no general formula. By experimental analysis, we found that the lower half of the star line fits the above characteristics, and so we choose one star line from a class of star lines as our control parameter φ. The function of the lower half of the star line is shown as follows:

$$param(t) = 1 - \frac{\sqrt[3]{\left(1-\sqrt[3]{\left(\frac{2t}{MaxI}-1\right)^2}\right)^2}}{\theta}, \quad t \in [1, MaxI].$$

where $MaxI$ is the maximum number of iterations and the value range of $param(t)$ is $[1 - 1/\theta, 1]$. θ can be used to control the lower bound of the $param(t)$. We have conducted many experiments on the value of θ and found that the search effect is the best when θ is 1.55. Therefore, we set $\theta = 1.55$. The graph of $param(t)$ is shown in Fig. 6. When $t \in [0, \lfloor \frac{MaxI}{2} \rfloor]$, the value of $param(t)$ decreases first at a slow pace and then fast. When $t \in [\lfloor \frac{MaxI}{2} \rfloor, MaxI]$, it increases first at a fast pace and then slow.

control parameter usage. While we are using $param(t)$ as the control parameter φ, we first generate a number range from 0 to 1 at random as the value of the random probability. If this probability is less than $param(t)$, then we perform TPEMO; otherwise, we perform TSEMO.

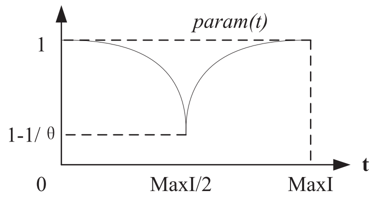


FIGURE 6. The graph of the control parameter.

D. CHECK AND CORRECTION OPERATION

When a new individual is generated, it must to be checked whether it meets the constraints in the model. If the placement exceeds the resource constraints, the placement is not feasible and thus needs to be corrected. We put each failed container into a VM that can hold it according to the First Fit algorithm until all containers successfully complete placement. Otherwise, a new individual is generated as initialized.

E. OTHER RELATED OPERATIONS

1) FITNESS FUNCTION

After the feasibility check and correction operation, the fitness of a new individual needs to be calculated for comparison purposes. Here, we use the target function Power in Section II as the fitness function.

2) NEW POPULATION GENERATION AND OPTIMAL INDIVIDUAL UPDATE

To retain the high-quality individuals, we mix parent and children individuals together, arrange them in ascending order of their fitness values, and assemble a new population by choosing the top $Cnum$ individuals. If the energy consumption of the optimal individual in the new population is less than that of the global optimal individual, the individual will be recorded as the global optimal individual. Otherwise, the global optimal individual stays the same.

IGA FLOW CHART

The flowchart of the improved IGA is shown in Fig. 7. It first performs the crossover operation. After performing the test and correction operation for each new individual, IGA updates the optimal location for each individual. Next, it uses the control parameter φ to selectively execute the mutation operations and once again performs the test and correction operation for the newly generated individuals. At last, IGA generates a new population and iteratively updates the global optimal individual.

IV. PERFORMANCE EVALUATION

In order to comprehensively evaluate the performance of our method, we introduce several state-of-the-art strategies or algorithms for comparison. The following subsection briefly describes the selected strategies or algorithms. What follows is the experimental scenarios and relevant parameter settings. Finally, we perform an analysis of the results obtained under each scenario.

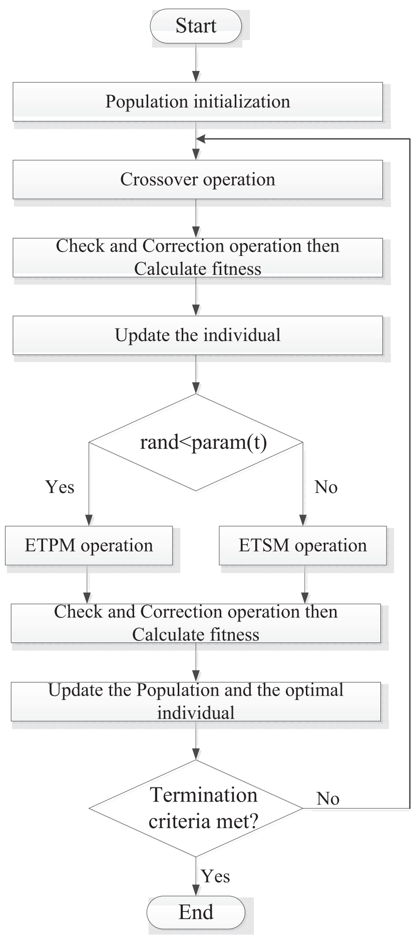

Start
Population initialization
Crossover operation
Check and Correction operation then Calculate fitness
Update the individual
rand<param(t)
Yes
No
ETPM operation
ETSM operation
Check and Correction operation then Calculate fitness
Update the Population and the optimal individual
Termination criteria met?
No
Yes
End

FIGURE 7. **The flow chart of IGA.**

A. ALGORITHMS FOR COMPARISON

In order to measure the performance of our strategy against the strategies commonly used in industrial, the traditional rapid placement algorithm, the goal-driven greedy algorithm and other heuristics algorithms, we choose the spread and binpack strategies, First Fit, Best Fit [28], GA and PSO [29]–[31] for comparison.

The spread and binpack strategies are two common strategies for container scheduling in industrial. They compute ranking based on a node's available CPU and memory, and the number of its hosted containers. In this paper, we rank the nodes using the sum of their available CPU and memory. The spread strategy places a new container in the node that not only has the biggest ranking but also meets the container's resource requirement. If two nodes have the same amount of available CPU and memory, the spread strategy prefers the node that has minimum number of containers. The advantage of this strategy is that you will lose only a few containers when a node goes down. In contrast, the binpack strategy places the new container in the node that meets the container's resource requirement but has the smallest ranking. If two nodes have the same amount of available CPU and memory, the binpack strategy prefers the node that has the maximum number of containers. The binpack strategy is able to avoid fragmentation since it leaves room for bigger containers.

The First Fit algorithm is a classical rapid placement algorithm with advantage of short response time. It processes the containers in a container queue and places each container in the first node that can meet its resource requirement.

The Best Fit algorithm is a greedy algorithm. It processes the containers in a container queue and places each container in the node that meets its resource requirement with the minimum fitness. In this paper, we use the energy consumption function in Section II as its fitness function.

The PSO is a heuristic algorithm that has fast convergence. It uses the global optimal position and individual historical optimal position to adjust the movement of current particle. The PSO algorithm typically involves two core formulas, i.e., velocity update formula and position update formula, which are shown as follows:

$$v_{i+1} = w{*}v_i + c1 * \zeta * (P_i - X_i) + c2 * \eta * (Pg_i - X_i), \quad (14)$$
$$X_{i+1} = X_i + v_{i+1}. \quad (15)$$

where i denotes the ith iteration; X_i and v_i denote the particle position and velocity at the ith iteration, respectively; w is the inertia weight for keeping the primary velocity; $c1$ and $c2$ are the learning factor; ζ and η are random numbers ranging from 0 to 1, and $P_i(Pg_i)$ is the best position in history of the ith(all) particle. The more details can be seen in the literature [29].

GA, as described in chapter III, is a heuristic algorithm with a completely different searching mechanism, compared to the PSO. It simulates an evolutionary mechanism of species in nature for searching. Our proposed IGA improves the mutation operation in GA.

The spread and binpack strategies, the First Fit and the Best Fit algorithms have similarities, in that they both process containers in a container queue and deal with only one container at each time. Moreover, they only obtain a single one CP solution at last. Furthermore, these algorithms inherently do not support parallelism and the quality of CP solution primarily depends on the container queue. The PSO, GA and IGA, however, are population-based searching techniques. They select an optimal CP solution from multiple feasible CP solutions, which make the algorithm have a global view to select the optimal one after all new containers being placed. Besides, the three heuristic algorithms have a relatively high complexity, but they can be parallelized and their execution time then can be greatly reduced. Table 2 summarizes the difference between these strategies or algorithms.

B. EXPERIMENTAL SCENARIO DESIGN AND PARAMETERS SETTING

Generally, due to the continuous creation and logout of VMs on the PM, the number of VMs hosted by each PM is not always the same. We call this scenario as nonuniform distribution. Instead, we call the scenario where there are same number of VMs on each PM as uniform distribution. In addition, in order to compare the performance of different

TABLE 2. The characteristics comparison of the seven algorithms.

Algorithm	Classify	characteristic
spread	common industrial strategie	containers are scattered across the nodes
binpack	common industrial strategie	tend to avoid fragmentation
First Fit	rapid placement algorithm	short response time
Best Fit	greedy strategie	choose the optimal state in each step
PSO	heuristic algorithm	velocity and position update
GA	heuristic algorithm	chromosome update
IGA	heuristic algorithm	improved the mutation operation of GA

algorithms at different scales of new containers, we define three scales, namely small-scale, med-scale and large-scale. Thus, we compare the performance of each algorithm on the placement for new containers with different scales in the two different VM distribution scenarios. Next, we set the required parameters in the experiment.

We set 50 heterogeneous PMs in the simulation. Through normalization, the total CPU and memory resources of each PM are both set as 1, i.e., $cpu_{PM_i}^{total} = 1$ and $mem_{PM_i}^{total} = 1$, and the resource specification for each VM is set as $cpu_{VM_j}^{total} = 0.0714$ and $mem_{VM_j}^{total} = 0.0832$; thus, each PM can hold up to 12 VMs. In addition, we use the method introduced in [32], [33] to generate random containers with different CPU and memory requirements. The method is shown in algorithm 2, where $\overline{U^{cpu}}$ and $\overline{U^{mem}}$ references for the CPU and memory utilization, respectively; $rand(1.0)$ returns a number in the range from 0 to 1.0 (not included) at random; and T is a probabilistic reference value, which is used to control the correlation of the CPU and memory utilization. To be more specific, $\overline{U^{cpu}} = 0.0071, \overline{U^{mem}} = 0.0082, T = 1$. The numbers of small, medium and large scale containers are set as 360, 720 and 1100, respectively. In addition, all experiments are repeated for 15 times.

Algorithm 2 Generation of container Synthetic Instances

```
  Input: n (the number of container), $\overline{Ucpu}$, $\overline{U^{mem}}$ and T
  Output: $< U_i^{cpu}, U_i^{mem} >$ Set
1 for $i = 1$ to $n$ do
2     $U_i^{cpu} = 2 * rand(\overline{U^{cpu}})$;
3     $U_i^{mem} = rand(\overline{U^{mem}})$
4     $r = rand(1.0)$
5     if $\left(\left(r < T\&\&U_i^{cpu} \geq \overline{U^{cpu}}\right) \| \left(r \geq T\&\&U_i^{cpu} < \overline{U^{cpu}}\right)\right)$
      then
6         $U_i^{mem} = U_i^{mem} + \overline{U^{mem}}$
7     end if
8 end for
9 return $< U_i^{cpu}, U_i^{mem} >$ Set
```

All the parameter settings of the three heuristic algorithms are shown in Table 3.

C. NONUNIFORM DISTRIBUTION SCENARIO

The nonuniform distribution scenario simulates the placement after the PMs randomly accept a batch of VMs for the continuous creation and logout, and the VM accept a batch of containers for the continuous copy and delete. The total number of VMs hosted by PMs is set as 340, and each VM runs some containers. The average VM resource utilization before CP and that after CP are shown in Table 4.

TABLE 3. Parameter setting.

Algorithm	Parameter setting
PSO	$c1 = c2 = 2, w = 0.73, Popsize = 25$
GA	$Popsize = 25, pcrossover = 0.75, pmutation = 0.035$
IGA	$Popsize = 25, pcrossover = 1, \varphi = param(t)$

TABLE 4. Average utilization of VM resources before and after CP in the nonuniform distribution Scenario.

	Small-scale	Med-scale	Large-scale
Before placement	Cpu:63.87%; Mem:63.32%	Cpu:63.87%; Mem:63.32%	Cpu:63.87%; Mem:63.32%
After placement	Cpu:74.26%; Mem:73.89%	Cpu:74.74%; Mem:83.88%	Cpu:95.64%; Mem:94.74%

TABLE 5. The relative gains in energy-saving of the IGA compared to other six algorithms in the nonuniform distribution scenario.

	Small-scale	Med-scale	Large-scale
Spread	0.94%	1.14%	0.13%
binpack	0.87%	1.07%	0.85%
First Fit	0.45%	0.58%	0.66%
Best Fit	-0.45%	-0.75%	0.17%
PSO	0.55%	0.67%	0.36%
GA	-0.0707%	0.14%	0.57%

TABLE 6. The computation time (s) comparisons of the four low computational complexity algorithms for one run in the nonuniform distribution scenario.

	Small-scale	Med-scale	Large-scale
Spread	0.0230186079	0.0388297173	0.0477303978
binpack	0.0291823719	0.0373321108	0.0651015496
First Fit	0.0557253299	0.1633873561	0.3491435875
Best Fit	0.9147878163	1.6545005028	2.4553231172

TABLE 7. The average computation time (s) comparison of the three heuristic algorithms at 5000 iterations in the nonuniform distribution scenario.

	Small-scale	Med-scale	Large-scale
PSO	2061.5889311	7385.1640248	156874.51320
GA	22215.032024	48948.756198	74844.075756
IGA	3919.0178061	7828.3049242	37791.866990

We compare IGA with other strategies and algorithms. The relative energy-saving gains are shown in Table 5. The comparison of the energy consumption and the average computation time for one iteration are summarized in Fig. 8, Table 6 and Table 7, respectively.

The graphs of the spread, binpack, First Fit and Best Fit in our experiment are a horizontal line, as shown in Fig. 8. The reason is that the four algorithms do not have a random mechanism, such that they can produce only one CP solution no matter how many times the experiment is repeated. The performance of GA and IGA are substantially better than the other methods except Best Fit on all three container scales.

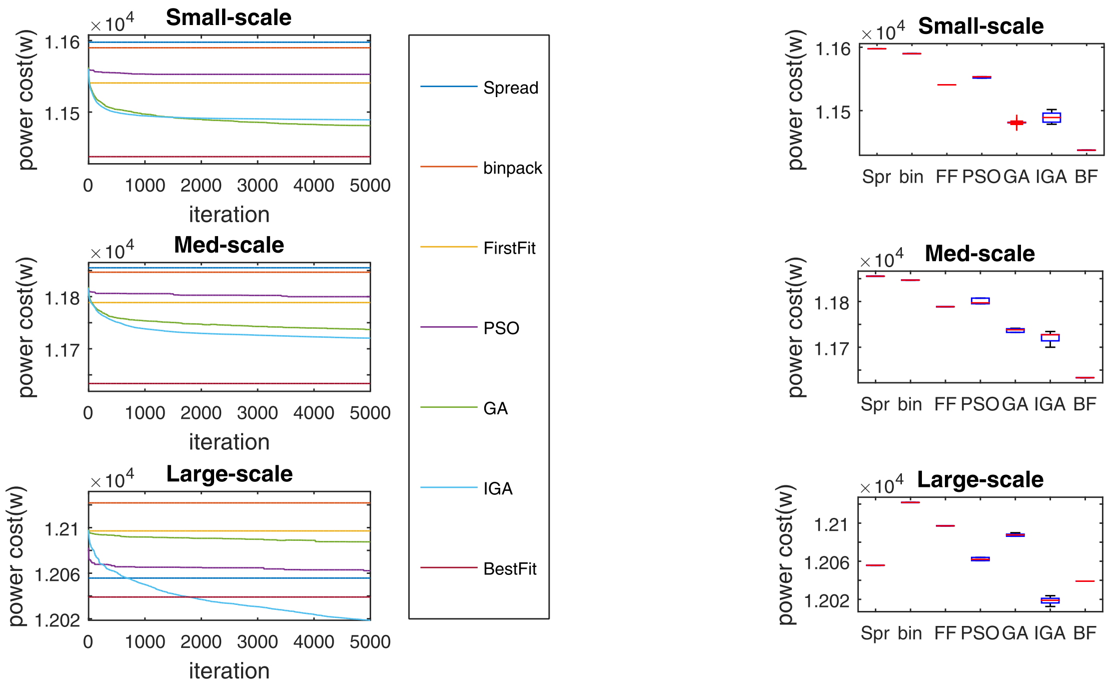


FIGURE 8. The energy consumption comparison of the seven algorithms in the nonuniform Distribution Scenario.

They perform the best when the CPU resource utilization is 74.74% and memory resource utilization is at 83.88%, and the largest improvement of the IGA compared to other algorithms can even reach 1.14%. However, when the resource utilization exceeds 84.74% for CPU and 83.88% for the memory, the improvements of GA and IGA slightly decrease. In general, the performance of IGA is better than that of GA, especially when the resource utilization of the VM is relatively high (CPU resource utilization exceeds 74.26% and memory resource utilization exceeds 73.89%). The improvement of IGA over GA increases as the resource utilization increases, and the largest energy savings improvement can even reach 0.57%. This also verifies that our improved mutation operation performs better than GA when VMs have high resource utilization. In addition, the experimental result shows that IGA can save more than 0.45% of energy consumption comparing to the best solution in the initial population generated by the First Fit algorithm, which indicates that IGA has a good searching performance than First Fit. It is also observed that PSO performs better than GA in the large-scale scenario but still worse than IGA, and that the Best Fit algorithm is significantly better than GA and IGA in the small-scale and mid-scale container placement scenarios while IGA performs better than Best Fit and GA (worst) in the large-scale container placement scenario. Therefore, our improved mutation method is experimentally proved to have a good effect on searching new CP solutions with better fitness when the VM resource utilization is high.

On one hand, the four algorithms, i.e., spread, binpack, First Fit and Best fit, have far lower computational complexity than the three heuristic algorithms, i.e., PSO, GA and IGA. On the other hand, the former only needs one run to complete the placement while the latter requires huge numbers of iterations to search a good solution. Therefore, we compare these two kinds of algorithms separately. The computation time comparisons over the four algorithms are shown in Table 6 and the computation time comparisons over the three heuristic algorithms are shown in Table 7.

As shown in Table 6, the four algorithms as a whole take a very small computational time cost because of their low computational complexity. However, the time cost of Best Fit is larger than the other three ones' since it uses a traversal searching strategy, which has a higher computational complexity. By comparison, the three heuristic algorithms have a very poor computational time cost. We remark that the efficiency of these algorithms can be improved by parallelization. In this paper, we only focus on the improvement of the algorithm's search mechanism and do not perform research on algorithms' parallel optimization. In the following, we compare the computational time cost of the three heuristic algorithms. As shown in Table 7, GA has more time cost in the three different scenarios. This may be caused by the worse performance of its mutation operation in high VM resource usage that leads to a large number of infeasible solutions to be corrected. The time costs of the three heuristic algorithms increase sharply in Large-scale scenario. Notably, PSO has the maximum time cost and may also encounter the problem that too many infeasible solutions need to be corrected. Thanks to the exploitation of our improved mutation method, IGA has a low time cost in the three heuristic algorithms, especially in the large-scale scenario.

Best Fit and IGA are completely different in searching mechanisms and this experiment shows their own advantages. Specifically, Best Fit is a greedy algorithm, which

TABLE 8. Average utilization of the VM resources before and after CP in the uniform distribution scenario.

	Small-scale	Med-scale	Large-scale
Before placement	Cpu:60.5%; Mem:60.46%	Cpu:60.5%; Mem:60.46%	Cpu:60.5%; Mem:60.46%
After placement	Cpu:70.92%; Mem:70.58%	Cpu:81.26%; Mem:80.89%	Cpu:92.62%; Mem:91.93%

TABLE 9. The relative gains in energy-saving of the IGA compared to other six algorithms in the uniform distribution scenario.

	Small-scale	Med-scale	Large-scale
Spread	0.24%	0.29%	0.19%
binpack	0.16%	0.24%	0.16%
First Fit	0.14%	0.16%	0.14%
Best Fit	-0.32%	-0.35%	-0.12%
PSO	0.13%	0.168%	0.14%
GA	-0.0726%	0.0854%	0.13%

TABLE 10. The computation time (s) comparisons of the four low computational complexity algorithms for one run in the uniform distribution scenario.

	Small-scale	Med-scale	Large-scale
Spread	0.0264035464	0.0462832532	0.0451671044
binpack	0.0237872253	0.0326790677	0.0562550373
First Fit	0.0534573955	0.1480580580	0.3267901073
Best Fit	1.0719190374	1.9153617134	2.7334571218

TABLE 11. The average computation time (s) comparison of the three heuristic algorithms at 5000 iterations in the uniform distribution scenario.

	Small-scale	Med-scale	Large-scale
PSO	1394.4937099	4104.5347651	14860.896511
GA	23735.186635	45377.754627	70008.606312
IGA	3754.0942869	7177.3628598	12230.74869

places each container one by one to choose the current best placement. It can get a good solution, but the effect mainly depends on the container queue. Albeit such a limitation, the algorithm does not support random search mechanism and thus results in poor target searching performance. IGA is a heuristic algorithm based on a population update strategy, which shows a good target searching performance but needs lots of iterations to search a good result. The combination of the two algorithms may have a very good performance in both solution quality and convergence rate. For example, IGA is used to select the optimal container queue for Best Fit or Best Fit is used to generate the initial population for IGA. We will try these combinations in the future work.

D. UNIFORM DISTRIBUTION SCENARIO

To fairly compare the performance of these algorithms, we run them in exactly the same initial scenario. Each PM hosts 7 VMs, and each VM runs the same batch of containers. The average VM resource utilization before CP and that after CP are shown in Table 8.

We compare IGA with other strategies and algorithms, and the relative energy-saving gains are shown in Table 9. Table 10, Table 11 and Fig. 9 compare the average execution time and the energy consumption, respectively.

The results are similar to the nonuniform scenario, but the difference is that the energy-saving improvement of IGA is lower compared with the nonuniform scenario. As a result, our improved IGA is more effective in solving the CP problem in the nonuniform distribution scenario, especially when the VM resource utilization is high. It performs the best when both the CPU and memory resource utilization are at approximately 84% based on our experiment.

V. RELATED WORK

The optimal scheduling of VMs in data centers has been widely investigated in the past decade. As a finer-grained virtualization technique, containers have recently gained increasingly more popularity in both academia and industry, and such a problem in the CaaS paradigm is revisited in the literature.

Tchana *et al.* [34] first proposed a resource management strategy under CaaS to reduce the resource usage of VMs while keeping a high Quality of Service (QoS) for applications. It adequately utilizes the application-level information in the optimal scheduling for the underlying VM consolidation. They concluded that the resource management that combines the two-layer information (application level and virtualization level) is better than the separate resource management for each layer. Zhang *et al.* [28] also obtained a similar conclusion in their work. By comparison, they analyzed the differences in the resource utilization computations under the two methodologies and then introduced a container placement strategy based on the bestfit algorithm for the two situations where there are sufficient and/or insufficient virtual machine resources. More concretely, they optimized the initial placement for new containers to minimize the number of PMs and resource waste. The work of Hussein *et al.* [35] is mainly based on [28]. They focused on improving the utilization of both VM and PM resources simultaneously. Different from the work of [28], they proposed an improved ant colony optimization (ACO) algorithm based on Best Fit algorithm and evaluated their algorithms based on Google cloud workload. However, the above works do not consider the objective of energy savings. Tchana *et al.* [11] took into account the energy consumption in their following work. They introduced an improved strategy for VM consolidation, which aggregates container payloads to reduce the number of VMs and PMs and thus decreases the power consumption.

In [4], Piraghaj *et al.* optimized the initial placement of containers to minimize VM resource waste in CaaS. Specifically, they first analyzed the use pattern of application tasks, investigated the impact of workload characters on VMs, and then proposed an efficient strategy to choose suitable host VMs for containers by considering their resource usage instead of the amount that is estimated by users. Afterwards, they extended the work to support energy-efficient VM consolidation in [10] by aggregating container payloads and reducing the number of active PMs. Notably, they leverage Pearson's correlation coefficient for container and PM payloads to select suitable PMs and propose a container

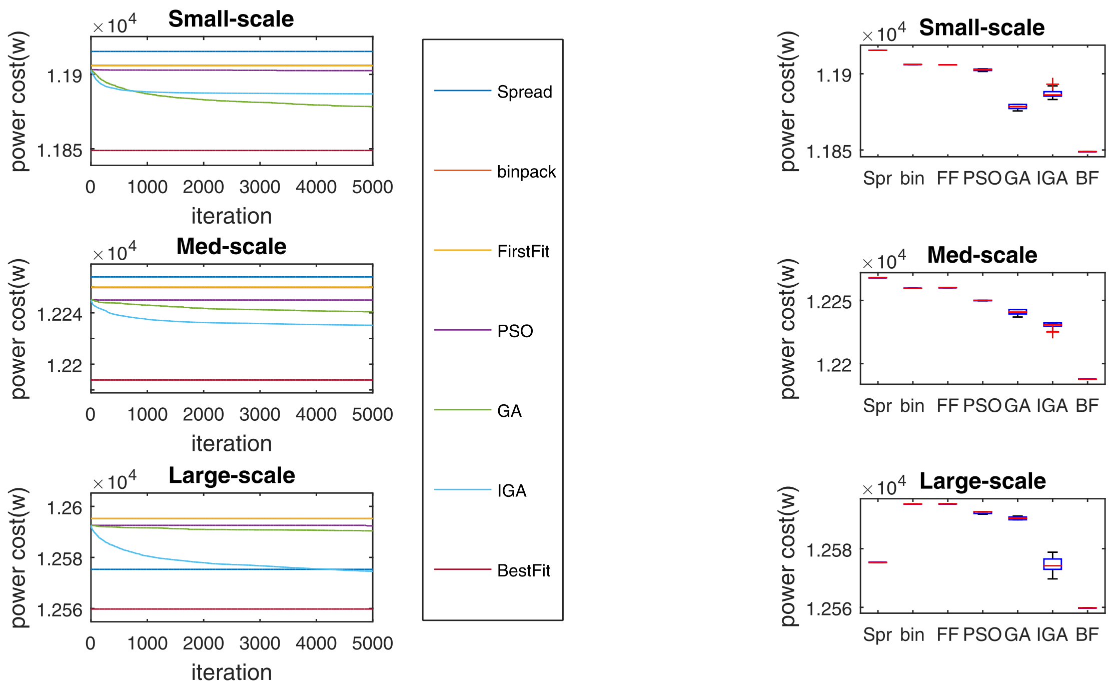


FIGURE 9. **The energy consumption comparison of the seven algorithms in the uniform Distribution Scenario.**

migration strategy for both underloaded and overloaded PM scenarios to reduce the number of active PMs and the server's energy consumption. However, the above works [10], [11] only consider a linear energy consumption model.

Docker, a widely adopted off-the-shelf container technique in the industry, provides three placement strategies, i.e., spread, binpack and random [8]. Their main purpose is to quickly deploy containers for customers, but they are not ideal solutions with regard to energy efficiency.

Furthermore, many heuristic algorithms are used in container management. Guerrero *et al.* [36] used NSGA-II to optimize container allocation and elasticity management. Kaewkasi *et al.* [3] exploited an ACO-based algorithm in container scheduling for resource balance. Fan *et al.* [29] leveraged an improved PSO to optimize a VM-Container hybrid hierarchical resource scheduling problem.

VI. CONCLUSION AND FUTURE WORK

This paper clarified the energy-efficient CP optimization problem under a nonlinear server energy-consumption model in CaaS and proposed an energy-efficient strategy to solve it. To be specific, we treat the CP issue as a packing problem and establish an energy-efficient CP model for optimization. When adopting the conventional GA to find the optimal solution for the problem, a flaw occurs in which the diversity of the population cannot be guaranteed as the resource utilization of VMs becomes higher. This can result in significant performance degradation. To solve this problem, we introduced an improved GA called the IGA that replaces the original mutation operation with two proposed exchange mutation operations. We also creatively use a control parameter φ to control their usage. At last, through extensive experimental tests, we show that our strategy can achieve better energy savings compared with the two existing Docker Swarm strategies, i.e., spread and binpack. In addition, the proposed IGA is more effective at solving the energy-efficiency CP problem in the nonuniform distribution scenario when the resource utilization of VMs is high, compared with the First Fit, PSO algorithm and conventional GA. Both of the Best Fit algorithm and IGA have their advantages and disadvantages, the combination of these two algorithms may has a very good performance in both solution quality and convergence rate.

However, the current work still needs to be improved in terms of the following aspects. First, it only considers a single optimization objective, i.e., energy efficiency, and thus, more objectives such as low resource waste and high availability should be taken into account. Second, we only evaluate the IGA using MATLAB simulations with homogeneous PMs and VMs and randomly generated containers. Therefore, it should be tested in a real system. Third, the control parameter φ in our algorithm is a function of the number of iterations. It might be better to add more different mutation operations and change the control parameter φ as a function of the number of iterations and VM resource utilization. Last, our strategy only applies to a static placement of containers, and it is unable to support real-time relocation and scheduling; thus, we hope to extend the work to dynamic container consolidation in the future.

REFERENCES

[1] T. Vasily, L. Rupprecht, D. Skourtis, W. Li, R. Rangaswami, and M. Zhao, "Evaluating Docker storage performance: From workloads to graph drivers," *Cluster Comput.*, vol. 3, pp. 1–14, Jan. 2019.

[2] C. Diekmann, J. Naab, A. Korsten, and G. Carle, "Agile network access control in the container age," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 1, pp. 41–55, Mar. 2019.

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

[4] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Efficient virtual machine sizing for hosting containers as a service (SERVICES 2015)," in *Proc. IEEE World Congr. Services*, Jun./Jul. 2015, pp. 31–38.

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

[6] AWS. *AWS Fargate*. Accessed: Jun. 20, 2019. [Online]. Available: https://aws.amazon.com/fargate/?nc1=h_ls

[7] Kubunetes. *Production-Grade Container Orchestration*. Accessed: Jun. 20, 2019. [Online]. Available: https://kubernetes.io/

[8] Docker Docs. *Docker Swarm Strategies*. Accessed: Jun. 20, 2019. [Online]. Available: https://docs.docker.com/swarm/scheduler/strategy/

[9] Dperny. *Docker/Swarm*. Accessed: Jun. 20, 2019. [Online]. Available: https://github.com/docker/swarm/tree/master/scheduler/strategy

[10] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "A framework and algorithm for energy efficient container consolidation in cloud data centers," in *Proc. IEEE Int. Conf. Data Sci. Data Intensive Syst.*, Dec. 2016, pp. 368–375.

[11] A. Tchana, N. De Palma, I. Safieddine, and D. Hagimont, *Software Consolidation as an Efficient Energy and Cost Saving Solution*. Amsterdam, The Netherlands: Elsevier, 2016.

[12] H. Zhao, J. Wang, F. Liu, Q. Wang, W. Zhang, and Q. Zheng, "Power-aware and performance-guaranteed virtual machine placement in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1385–1400, Jun. 2018.

[13] A.-C. Orgerie, L. Lefèvre, and J. P. Gelas, "Demystifying energy consumption in grids and clouds," in *Proc. Int. Conf. Green Comput.*, Aug. 2010, pp. 335–342.

[14] G. Varsamopoulos, Z. Abbasi, and S. K. S. Gupta, "Trends and effects of energy proportionality on server provisioning in data centers," in *Proc. Int. Conf. High Perform. Comput.*, Dec. 2011, pp. 1–11.

[15] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah, "Analyzing the energy efficiency of a database server," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 231–242.

[16] D. Miyuru, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Commun. Surv. Tuts.*, vol. 18, no. 1, pp. 732–794, 1st Quart., 2016.

[17] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[18] K. Xie, X. Huang, S. Hao, M. Ma, P. Zhang, and D. Hu, "E^3 MC: Improving energy efficiency via elastic multi-controller SDN in data center networks," *IEEE Access*, vol. 4, pp. 6780–6791, 2017.

[19] G. Dunwei, J. Sun, and Z. Miao, "A set-based genetic algorithm for interval many-objective optimization problems," *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 47–60, Feb. 2018.

[20] K. Sahil, H. Kumar, S. Kaushal, A. K. Sangaiah, "Genetic algorithm-based cost minimization pricing model for on-demand IaaS cloud service," *J. Supercomput.*, vol. 2, pp. 1–26, Mar. 2018.

[21] C.-H. Chen, Y.-H. Chen, J. C.-W. Lin, and M.-E. Wu, "An effective approach for obtaining a group trading strategy portfolio using grouping genetic algorithm," *IEEE Access*, vol. 7, pp. 7313–7325, 2019.

[22] S. Yin, P. Ke, and L. Tao, "An improved genetic algorithm for task scheduling in cloud computing," in *Proc. 13th IEEE Conf. Ind. Electron. Appl. (ICIEA)*, Wuhan, China, May/Jun. 2018, pp. 526–530.

[23] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1996.

[24] S. M. Thede, "An introduction to genetic algorithms," *J. Comput. Sci. Colleges*, vol. 20, no. 1, pp. 325–336, 2004.

[25] L. Luo, W.-J. Wu, and F. Zhang, "Energy modeling based on cloud data center," *J. Softw.*, vol. 25, no. 7, pp. 1371–1387, 2014.

[26] W. T. Rhee and M. Talagrand, "On line bin packing with items of random size," *Math. Oper. Res.*, vol. 18, no. 2, pp. 438–445, 1993.

[27] S. Polyakovskiy and R. M'Hallah, "A hybrid feasibility constraints-guided search to the two-dimensional bin packing problem with due dates," *Eur. J. Oper. Res.*, vol. 266, pp. 819–839, May 2017.

[28] Z. Rong, A.-M. Zhong, B. Dong, F. Tian, and R. Li, "Container-VM-PM architecture: A novel architecture for docker container placement," in *Proc. Int. Conf. Cloud Comput.* Cham, Switzerland: Springer, 2018, pp. 128–140.

[29] C. Fan, Y. Wang, and Z. Wen, "Research on improved 2D-BPSO-based VM-container hybrid hierarchical cloud resource scheduling mechanism," in *Proc. IEEE Int. Conf. Comput. Inf. Technol.*, Dec. 2017, pp. 754–759.

[30] G. Wang, J. Guo, Y. Chen, Y. Li, and Q. Xu, "A PSO and BFO-based learning strategy applied to faster R-CNN for object detection in autonomous driving," *IEEE Access*, vol. 7, pp. 18840–18859, 2019.

[31] H. Lu, Q. Zhou, Z. Fei, and R. Zhou, "Scheduling based on interruption analysis and PSO for strictly periodic and preemptive partitions in integrated modular avionics," *IEEE Access*, vol. 6, pp. 13523–13540, 2018.

[32] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *J. Comput. Syst. Sci.*, vol. 79, no. 8, pp. 1230–1242, 2013.

[33] F. Tian, R. Zhang, J. Lewandowski, K.-M. Chao, L. Li, and B. Dong, "Deadlock-free migration for virtual machine consolidation using Chicken Swarm Optimization algorithm," *J. Intell. Fuzzy Syst.*, vol. 32, no. 2, pp. 1389–1400, 2017.

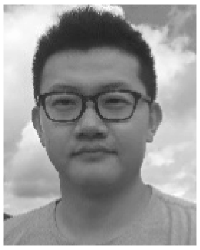
[34] A. Tchana, G. S. Tran, L. Broto, N. Depalma, and D. Hagimont, "Two levels autonomic resource management in virtualized IaaS," *Future Gener. Comput. Syst.*, vol. 29, no. 6, pp. 1319–1332, 2013.

[35] M. K. Hussein, M. H. Mousa, M. A. Alqarni, "A placement architecture for a container as a service (CaaS) in a cloud environment," *J. Cloud Comput.*, vol. 8, no. 1, p. 7, 2019.

[36] G. Carlos, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *J. Grid Comput.*, vol. 16, no. 1, pp. 113–135, 2017.

RONG ZHANG received the B.S. degree in mathematics and applied mathematics from Minnan Normal University, Zhangzhou, China, in 2004, and the M.S. degree in fundamental mathematics from Guangxi University for Nationalities, Nanning, China, in 2007. He is currently pursuing the Ph.D. degree with the MOE Key Laboratory for Intelligent Networks and Network Security, Xi'an Jiaotong University. His research interests include cloud computing and big data analytics.

YAXING CHEN received the B.S. degree in software engineering from Northwestern Polytechnical University, in 2012. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Xi'an Jiaotong University. He was a Visiting Student with Virginia Tech, from 2016 to 2018. His research interests include data security and privacy and cloud computing, with focus on data access control and trusted computing.



BO DONG received the Ph.D. degree in computer science and technology from Xi'an Jiaotong University, in 2014, where he completed Postdoctoral Research with the MOE Key Laboratory for Intelligent Networks and Network Security, from 2014 to 2017, and currently serves as the Research Director of the School of Continuing Education. His research interests include intelligent e-Learning and data mining.

FENG TIAN received the B.S. degree in industrial automation and the M.S. degree in computer science and technology from the Xi'an University of Architecture and Technology, Xi'an, China, in 1995 and 2000, respectively, and the Ph.D. degree in control theory and application from Xi'an Jiaotong University, Xi'an, in 2003. He has been with Xi'an Jiaotong University, since 2004, where he is currently with the National Engineering Laboratory of Big Data Analytics and also with the Systems Engineering Institute as a Professor. He is also a member of the Satellite-Terrestrial Network Technology Research and Development Key Laboratory, Shaanxi. His research interests include cloud computing, big data analytics, learning analytics, and system modeling and analysis. He is a member of CCF. He received the Second Class Prize of National Science and Technology Progress Award of China (4th position, in 2017), the First Class Price of MOE Science and Technology Progress Award of China (4th position, in 2015), the First Class Prize of Science and Technology Progress Award of the Chinese Electronical Society (5th position, in 2013), and the TOP Grade Prize of Teaching Achievement Award of Shaanxi Province Award, China (2nd position, in 2017).

QINGHUA ZHENG received the B.S. degree in computer software, the M.S. degree in computer organization and architecture, and the Ph.D. degree in system engineering from Xi'an Jiaotong University, in 1990, 1993, and 1997, respectively. He completed Postdoctoral Research at Harvard University, in 2002, and was a Visiting Professor with The University of Hong Kong, from 2004 to 2005. Since 1995, he has been with the Department of Computer Science and Technology, Xi'an Jiaotong University, where he is currently a Professor and serves as the Vice President. His research interests include intelligent e-Learning, network security, and trusted software.