Be Good Neighbors: A Novel Application Isolation Metric Used to Optimize the Initial Container Placement in CaaS

RONG ZHANG\textsuperscript{1,2}, YAXING CHEN\textsuperscript{1,2}, FA ZHANG\textsuperscript{1,2}, FENG TIAN\textsuperscript{1,2,3}, (Member, IEEE), AND BO DONG\textsuperscript{3,4}, (Member, IEEE)

\textsuperscript{1}Institute of Electronic and Information Science, Xi’an Jiaotong University, Xi’an 710049, China
\textsuperscript{2}Shaanxi Province Key Laboratory of Satellite and Terrestrial Network Technology Research and Development, Xi’an Jiaotong University, Xi’an 710049, China
\textsuperscript{3}National Engineering Lab for Big Data Analytics, Xi’an Jiaotong University, Xi’an 710049, China
\textsuperscript{4}School of Continuing Education, 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 2018YFB1004500, in part by the National Natural Science Foundation of China under Grant 61877048 and Grant 61532015, 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 Project of China Knowledge Centre for Engineering Science and Technology, in part by the Fundamental Research Funds for the Central Universities under Grant sxzd012020003, and in part by the Project of XJTU-SERVYOU Joint AI Innovation Center.

ABSTRACT Many cloud providers benefit from fine-grained resource management in Container as a Service (CaaS), e.g., energy savings. However, while optimizing power consumption, cloud providers also need to consider service performance. Few studies in the literature performed a balanced optimization of the two objectives in the initial container placement (ICP). As such, the obtained solutions are often locally optimal. In this work, we model the ICP as a bi-objective optimization problem. Unfortunately, directly applying a conventional performance modeling method in solving the problem is complex and costly, especially for a large-scale application scenario. It also fails to capture the impacts of some non-resource constraints such as high availability constraints. To tackle these issues, we propose a novel application isolation metric to quantify the overall service performance. With regard to the power consumption objective, we opt for a non-linear model since a linear one makes power consumption indistinguishable in a homogeneous setting. With the two building blocks, we introduce an optimization algorithm named First Fit based improving Genetic Algorithm (denoted by FF-based-IGA) to find the best ICP solution. Our experimental results are as follows: 1) the proposed application isolation metric is effective at improving service performance, and 2) FF-based-IGA achieves a desirable balance between power consumption and service performance.

INDEX TERMS CaaS, container placement, application interference, anti-affinity constraint.

I. INTRODUCTION Container as a Service (CaaS) \textsuperscript{[1]–[5]}, is a promising cloud service delivery paradigm that has been widely adopted by industry. A typical CaaS deployment model (adopted by, for example, the Amazon Web Services (AWS) Fargate \textsuperscript{[6]}) is depicted in Fig. 1, where applications are packaged in containers, containers run on virtual machines (VMs), and VMs are hosted on physical machines (PMs). CaaS lies between Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). As a more flexible and efficient approach for resource sharing, CaaS offers easier manipulation for upper-layer applications than IaaS and more flexible support for the bottom-layer system than PaaS. Since containers are lighter than VMs and allow for high-density deployment, fine-grained resource management to reduce the servers’ energy consumption has attracted increasing attention \textsuperscript{[7]–[9]}. Generally, using more PMs means more power consumption. To minimize the power consumption, containers packaging applications should be assigned to suitable VMs that occupy the smallest number of PMs. However, when making scheduling decisions to accomplish power savings, the cloud providers also need to consider the service performance.
“Crowded” containers on a minimum number of PMs increase the risk of violating Service Level Agreement (SLA). On the contrary, “loose” containers can meet the requirements of SLA, but it implicates that more PMs will be occupied, resulting in inefficient energy use. Therefore, how to allocate these containers in PMs has become the key to balance the minimum power consumption and the best service performance.

Our previous work [10] indicates that the container placement optimization plays an essential role in energy-saving problem. However, few studies in the literature take both the two objectives into consideration for initial container placement (ICP) in CaaS. On the other hand, the underlying performance modeling methods adopted by most related works [11]–[17] usually establish a relationship function between performance and resource requirements. Due to the required massive tests and complex computations, these methods are always blocked in a large-scale application scenario. Furthermore, they fail to capture the impacts of some non-resource constraints such as high availability constraints.

It is unarguable that the service performance in CaaS closely correlates to the resource competitions amongst real workloads (i.e. applications) co-hosted by the same PM; an intensified competition can greatly downgrade the service performance. Refer to [11], this performance degradation phenomenon is know as Interference. So an application when being deployed is often “excluded” by others co-located in the same PM due to resource competitions; a servere Interference means an intensified competition. As such, an Interference often has an adverse impact on the service performance, e.g., lengthening the application response time [17]–[22]. As noted in [23], the best way to prevent Interference is “don’t let such quarreling people be neighbors” in the initial stages. Thus, the better the isolation of these “quarreling” applications is, the slighter the Interference will be (the better the service performance will be). Based on above analysis, we were struck by a novel idea in solving the aforementioned problem, i.e., utilizing application isolation to model service performance. Thus, the service performance objective in the bi-objective optimization problem is converted to the application isolation objective. This is a feasible equivalent transformation. Alibaba proposed a kind of constraint named the anti-affinity constraint based on its historical data observations and parameter setting experience in Alibaba’s Global Scheduling Algorithm Competition 2018 [24] to optimize service performance through application isolation management. The anti-affinity constraint gives an upper threshold that limits the number of a specified type of application from its co-located applications. Here, instances used to package applications can be container instances or VM instances and one visualized instance is set to host only one application. In this paper, we adopt container instances by default. The advantage of the anti-affinity constraints method is that it can bypass a huge amount of accurate performance modeling work and make it easier to make a container placement decision. It can also be used to formalize the application isolation requirements of some non-resource constraints such as high availability constraints. However, the anti-affinity constraints method cannot quantify the effect of application isolation and may neglect application isolation optimization, which may lead to a top-level “crowded” containers situation close to the acceptable supremum. The high “crowded” situation may cause a violation or a high risk of default due to the experience-based anti-affinity constraints. In short, the anti-affinity constraint method is an effective method to achieve application isolation, but the current research based on anti-affinity constraints ignores the application isolation quantification. A novel method to quantify service performance that uses anti-affinity constraints still face challenges.

In this paper, we try to strike a balance between the minimum power consumption and the best application isolation for the ICP in CaaS. First, we model this as a bi-objective optimization problem and design an application isolation metric model for a container placement solution based on anti-affinity constraints. We also prove that the proposed metric model is monotonous and nondecreasing, which satisfies a common property of a generic performance function. Second, we set the application isolation metric as one objective model and introduce a non-linear power consumption model as the other objective model. To find the best container placement solution, we propose an algorithm named First Fit based improving Genetic Algorithm (FF-based-IGA) that uses the two objectives to guide the container placement. FF-based-IGA encodes container processing sequences as chromosomes, uses the First Fit (FF) algorithm for rapid container placement and uses the improving Genetic Algorithm (IGA) to optimize the container processing sequences. In the experiment, we compare FF-based-IGA with the other four algorithms, including a FF algorithm; two kinds of Best-Fit (BF) algorithms (including a bi-objective version and a single objective version that uses or does not use our proposed application isolation metric model), and a FF-based-IGA with a single objective. The experimental results demonstrate that the use of an application isolation metric can effectively help to alleviate the “crowded” situation. Meanwhile, both the FF-based-IGA with and without our application isolation metric model are superior in minimizing the power consumption compared to other algorithms. The proposed FF-based-IGA can achieve a desirable balance
between the two objectives. The main contributions are summarized as follows.

- We research on a bi-objective optimization problem based on anti-affinity constraints for the ICP in CaaS.
- We utilize application isolation to model service performance and convert the service performance optimization problem to the application isolation optimization problem.
- We propose an application isolation metric to quantify the application isolation effect for a container placement solution and prove it satisfies a common property of a generic performance function.
- We propose an algorithm named FF-based-IGA, which takes the rapid placement advantage from a FF algorithm and the evolutionary mechanism advantage from a IGA algorithm.
- We carry out extensive experiments based on the container datasets from the Alibaba Competition 2018, and the results confirm that the proposed application isolation metric is effective in application interference mitigation. Meanwhile, FF-based-IGA achieves a desirable balance between the two objectives.

The rest of this paper is organized as follows. Section 2 introduces the design of the application isolation metric and the optimization models put forward in this study. The proposed FF-based-IGA is described and evaluated in Sections 3 and 4, respectively. Section 5 discusses the related work. We summarize the entire paper and our future work in the last section.

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 deployment model that is introduced in [1]. One container can host just one application. The resource requirements or specifications of the containers running the same type of application are identical. We only focus on homogeneous PMs and homogeneous VMS scenario in this paper, as it is the simplest and most manageable configuration mode and is more convenient for fair comparison of different algorithms. Heterogeneous PMs scenarios are more complex than homogenous PMs scenarios, since the anti-affinity constraints of the same application may be different on different types of PMs. Only two resources, the CPU and memory, are considered in this paper. For simplicity, we set all the container Operating System types as the same, and so every container can fit in any VM. The number of applications, containers, VMs, and PMs are set as Anum, Cnum, VMnum, PMnum, respectively.

A. ANTI-AFFINITY CONSTRAINT IN CaaS

The Anti-affinity constraint in our CaaS deployment model can be introduced as follows. The anti-affinity constraint between two types of applications is denoted by an ordered triple as \(\langle app_y, app_x, N_{IXY} \rangle\). It means that if any container instance of packaging type \(Y\) application (denoted as \(app_y\)) has been assigned to a PM \(PM_i\), then the number (denoted as \(n_{IX}\)) of container instances of packaging type \(X\) applications (denoted as \(app_x\)) on the same PM cannot be larger than the threshold number of container instances, which is denoted as \(N_{IXY}\). Since one container hosts just one application, the number of containers belonging to the same type of application is the number of this type of application. Therefore, the number of \(app_x\) and the threshold number of applications \(app_x\) for \(app_y\) in \(PM_i\) can also be denoted as \(n_{IX}\) and \(N_{IXY}\), respectively. Moreover, in order to describe the problem clearly, we call the second constrained application \(app_x\) in \(\langle app_y, app_x, N_{IXY} \rangle\) as an interference-risk type application and call the first one \(app_y\) as a neighbor application (of the interference-risk application). If an \(app_x\) is not an interference-risk application for \(app_y\), then we define the threshold number of applications \(N_{IXY}\) as \(+\infty\) for \(app_y\). By these setting, some non-resource constraints with requirements of application isolation can be transformed to anti-affinity constraints. For example, we can use \(\langle app_y, app_x, 0 \rangle\) to express a high-availability constraint, which requires that two container instances of an application \(app_x\) cannot be placed on the same PM so as to avoid out-of-service when the PM goes down.

B. APPLICATION ISOLATION METRIC

It is difficult to judge whether interference has occurred for the placement of an \(app_y\) at its threshold number of applications \(N_{IXY}\) for \(app_y\) in \(PM_i\), because \(N_{IXY}\) is an indicator of performance tolerance and given based on experience. However, we notice that the closer the number of interference-risk type applications is to its threshold, the greater the risk or intensity of the interference from its neighboring applications, and also the worse the effect of the application isolation. On the contrary, the bigger the gap between the number of an interference-risk type application and its threshold number of applications, the better the application isolation effect. Therefore, the comparison between the number of type \(X\) applications \(n_{IX}\) and its threshold number of applications \(N_{IXY}\) becomes the key to observing the application isolation effect. We named the ratio of \(n_{IX}\) to \(N_{IXY}\) as the application crowding degree \((ACD)\) of \(app_x\) for \(app_y\) in \(PM_i\).

1) ACD

**Definition 1**: The ACD of an application \(app_x\) for its neighboring application \(app_y\) in a PM \(PM_i\) is denoted as \(ACD_{XY}\) and is calculated as follows:

\[
ACD_{XY} = \begin{cases} 
+\infty, & N_{IXY} = 0 \\
\frac{n_{IX}}{N_{IXY}}, & 0 < N_{IXY} < +\infty \\
0, & N_{IXY} = +\infty \text{ or } n_{IX} = 0
\end{cases}
\]  

(1)

where \(n_{IX}\) is the number of applications \(app_x\) in \(PM_i\), \(N_{IXY}\) is the threshold number of applications \(app_x\) for \(app_y\), and \(0 \leq n_{IX} \leq N_{IXY}\).
Obviously, a smaller $ACD_{XY}$ reflects better application isolation between $app_x$ and $app_y$. However, it is impossible to distinguish the quality of different placements when their $ACD$ sums are the same. As shown in the Fig. 2, there are 6 containers denoted as $c_1, \ldots, c_6$, respectively. Two types of applications are packaged in the containers. While one $app_y$ is assigned to one PM, the other $app_y$ must be assigned to another PM due to the constraint from $<app_y, app_y, 0>$. The sum of the $ACD$ for each placement can be calculated as follows:

\[
\begin{align*}
S_{ACD_1} &= 0 + 4/5 = 4/5, \\
S_{ACD_2} &= 1/5 + 3/5 = 4/5, \\
S_{ACD_3} &= 2/5 + 2/5 = 4/5,
\end{align*}
\] (2)

where $S_{ACD_1}$ has a zero term because there is only one application in PM1 that belongs to Placement 1; as such, the number of other application is zero and the $ACD$ of other application for $app_y$ in PM1 is 0.

In order to distinguish their values more easily, an exponential function can play a good role. The sums of the New ACDs for each placement can be calculated as follows:

\[
\begin{align*}
S_{NACD_1} &= e^0 + e^{4/5} = 3.2255, \\
S_{NACD_2} &= e^{1/5} + e^{3/5} = 3.0435, \\
S_{NACD_3} &= e^{2/5} + e^{2/5} = 2.9836.
\end{align*}
\] (5)

We noticed that Placement 3 is obviously better than the other two placements in application isolation, and this also can be observed easily in numerical comparisons of the sums of their New $ACD$. Next, we will improve the use of $ACD$ and establish four metric models. The four metric models are designed to measure an application’s received “repulsion” from another application, the maximum received “repulsion” from other applications, the sum of each type of applications’ maximum received “repulsion” in a PM, and the sum of the above received “repulsion” in each PM for a container placement solution, respectively. For convenience, we called these metrics a joint name as the unacceptable tendency strength (UTS) and denote the four metrics as $UTS1$, $UTS2$, $UTS3$ and $UTS4$ respectively.

2) $UTS1$

The definition of $UTS1$ is given as follows.

Definition 2: In a container placement solution $s$, the $UTS$ of a $app_x$ for $app_y$ in a single PM $PM_i$, i.e., $UTS1(s, PM_i, app_y, app_x, N_{IXY}, n_{IX})$, is defined as follows:

\[
\begin{align*}
UTS1(s, PM_i, app_y, app_x, N_{IXY}, n_{IX}) &= \begin{cases} 
eq 0, & N_{IXY} = +\infty \quad \text{or} \quad n_{IX} = 0 \\
\left( e^{n_{IX}/N_{IXY}}, \quad 0 < N_{IXY} < +\infty \right) \quad \text{or} \quad n_{IX} = 0 \\
1, & N_{IXY} = 0
\end{cases}
\end{align*}
\] (8)

where $0 \leq n_{IX} \leq N_{IXY}$.

There are three cases in the design of $UTS1$.

Case 1: If the threshold $N_{IXY}$ is $+\infty$ or $n_{IX} = 0$, it means that $app_x$ is not an interference-risk type application or there is only one application. So $ACD_{XY} = 0$ and $UTS1$ can be set as $e^{ACD_{XY}} = e^0$.

Case 2: If $0 < N_{IXY} < +\infty$, it means that $app_x$ is an interference-risk type application, and we use $e^{ACD_{XY}}$, i.e. $e^{n_{IX}/N_{IXY}}$, as the $UTS1$.

Case 3: If $N_{IXY} = 0$, it means that $app_x$ is interference-risk application, and then $e^{ACD_{XY}} = e^{+\infty} = +\infty$. Notice that infinity is not convenient to compare values, and so we use a big enough constant $c1$ to replace it, where $c1 \gg 1$. In this paper, we set $c1 = 100$.

3) $UTS2$

One PM may run several types of applications, and so one $app_x$ may be affected by multiple neighboring applications co-located in the same PM. In order to better isolate an $app_x$ from other applications, the maximum $UTS1$ of $app_x$ should be minimized.

Definition 3: In a container placement solution $s$, the maximum $UTS1$ of $app_x$ regarding its neighboring applications on a single PM $PM_i$, i.e. $UTS2(s, PM_i, app_x, n_{IX})$, is defined as follows:

\[
\begin{align*}
UTS2(s, PM_i, app_x, n_{IX}) &= \max\{UTS1(s, PM_i, app_y, app_x, N_{IXY}, n_{IX}) | app_y \in NappSet\}.
\end{align*}
\] (9)

where $NappSet$ is the neighboring application set of $app_x$ in $PM_i$.

Then, the maximum $UTS2(s, PM_i, app_x, n_{IX})$ of all the applications co-located on $PM_i$ can be defined and calculated as $MaxUTS2(s, PM_i)$:

\[
\begin{align*}
MaxUTS2(s, PM_i) &= \max\{UTS2(s, PM_i, app_y, n_{IX}) | app_y \in PM_i, NappSet\}.
\end{align*}
\] (10)

where $PM_i, NappSet$ is the application type set of all the applications in $PM_i$.

Similarly, the maximum $MaxUTS2(s, PM_i)$ for all PMs in a container placement solution $s$ can be calculated as follows:

\[
\begin{align*}
MMaxUTS2(s) &= \max\{MaxUTS2(s, PM_i) | \forall PM_i \in PMSet\}.
\end{align*}
\] (11)
where $PM_{set}$ is the set of all the PMs in container placement solution $s$. $MMaxUTS2(s)$ is the maximum $UTS1$ among all the PMs, and it can intuitively display the worst application isolation level for this PM.

4) $UTS3$

In order to balance the application isolation optimization for each type of application co-located in the same PM, all the maximum $UTS2$ of each type application need be considered. Therefore, the definition of $UTS3$ is given as follows.

**Definition 4**: The $UTS$ of a container placement solution $s$ in a PM $PM_i$ is defined as follows:

$$UTS3(s, PM_i) = \sum_{appX \in PM_{appSet}} UTS2(s, PM_i, appX, n_{iX}).$$ (12)

5) $UTS4$

A container placement solution is made up of all sub-parts of container placement in each PM. In order to identify the worst isolation level of a container placement solution, we set the sum of each PM's $UTS3$ as the application isolation metric of the container placement solution.

**Definition 5**: In a container placement solution $s$, the $UTS$ of a container placement solution $s$ in a PM $PM_i$ is defined as follows:

$$UTS4(s) = \sum_{i=1}^{PNum} UTS3(s, PM_i).$$ (13)

The smaller the $UTS4(s)$ is, the better the application isolation effect for the container placement solution. As illustrated in Section 1, we can set $UTS4$ as the indicator of service performance.

$UTS4$ evaluates the service performance in view of the overall placement, while $MMaxUTS2$ evaluates the worst service performance locally on a single PM in view of the local placement. To better understand the four metrics, we respectively evaluate the three placements in Fig. 2 based on the four metrics in the follows. Note that, the three placements are denoted by $P1$, $P2$ and $P3$.

**Example 1**: The application isolation metric calculation of $P1$.

Since there is only one application (i.e. $appY$) on $PM_1$ that belongs to $P1$, we note $appN$ as the neighbor application of $appY$ and note $N_{iNY}$ as the threshold number of applications $appX$ for $appY$, then the number of neighbor application $appN$ is 0 (i.e. $n_{iNY} = 0$). Here, both of the $appY$ and $N_{iNY}$ have no practical meaning. According to Definition 2, $UTS1(P1, PM_1, appY, appN, N_{iNY}, n_{iNY})$ can be calculated as follow:

$$UTS1(P1, PM_1, appY, appN, N_{iNY}, 0) = e^0;$$
$$UTS2(P1, PM_2, appY, appX, 5, 4) = e^{4/5};$$
$$UTS2(P1, PM_1, appY, appN, 0) = max(e^0) = e^0;$$
$$UTS2(P1, PM_2, appY, appX, 4) = max(e^{1/5}) = e^{1/5};$$
$$UTS3(P1, PM_1) = e^0;$$
$$UTS3(P1, PM_2) = e^{4/5};$$

$$UTS4(P1) = e^0 + e^{4/5} = 3.2255;$$
$$MaxUTS2(P1, PM_1) = max(e^0) = e^0;$$
$$MaxUTS2(P1, PM_2) = max(e^{4/5}) = e^{4/5};$$
$$MMaxUTS2(P1) = max(e^0, e^{4/5}) = e^0/5;$$

$Example 2$: The $UTS4$ and $MMaxUTS2$ calculation of $P2$.

$$UTS4(P2) = e^{1/5} + e^{4/5} = 3.0435;$$
$$MMaxUTS2(P2) = max(e^{1/5}, e^{4/5}) = e^{4/5} = 2.2255.$$  

$Example 3$: The $UTS4$ and $MMaxUTS2$ calculation of $P3$.

$$UTS4(P3) = e^{2/5} + e^{2/5} = 2.9836;$$
$$MMaxUTS2(P3) = max(e^{2/5}, e^{2/5}) = e^{2/5} = 1.4918.$$  

It is obvious that $P3$ has the smallest $UTS4$. This is consistent with the fact that $P3$ overall is better than the other two placements in terms of application isolation. $P3$ also has the smallest $MMaxUTS2$, which is consistent with the fact that $P3$ is better than the other two in terms of the worst isolation performance of a single application on a single PM. The worst “crowding” case of a single application on a single physical machine in $P3$ is worse than the other two placements; this phenomenon can be shown from the comparison of the number of $appX$ on each PM2 of the three placements. Therefore, when $UTS4$ is set as an indicator of service performance, $MMaxUTS2$ can be used as a reference indicator of service performance.

6) THE PROOF OF THE MONOTONOUS AND NONDECREASING PROPERTY

To verify that the four kinds of $UTS$ we proposed conform to the fact that the bigger the gap between the number of an interference-risk type application and its threshold number of applications is, the better the application isolation effect, we prove that the four kinds of metric functions are monotonous and nondecreasing functions for the variable $n_{iX}$.

**Proposition**: $UTS1(s, PM_i, appY, appX, N_{iXY}, n_{iX})$, $UTS2(s, PM_i, appY, n_{iX})$, $UTS3(s, PM_i)$ and $UTS4(s)$ are monotonous and nondecreasing functions for the variable $n_{iX}$.

**Proof**: It is obvious that $UTS1$ is a constant function in Case 1 and Case 3, and it is a monotone and increasing function in Case 2; therefore, $UTS1(s, PM_i, appY, appX, N_{iXY}, n_{iX})$ is a monotonous and nondecreasing function of variable $n_{iX}$.

Assuming that $s_1$ and $s_2$ are two container placement solutions, $n_{i1}$ and $n_{i2}$ are the corresponding numbers of applications of $appX$ in one PM $PM_i$, and $0 \leq n_{i1} < n_{i2}$, then we have the following:

$$UTS1(s, PM_i, appY, appX, N_{iXY}, n_{i1}) \leq UTS1(s, PM_i, appY, appX, N_{iXY}, n_{i2}).$$ (14)

Since (13) is valid for each $appY$ in NappSet, then we have the following:

$$UTS2(s, PM_i, appY, n_{i1}) = max\{UTS1(s, PM_i, appY, appX, N_{iXY}, n_{i1})\} \exists appY \in NappSet$$
$$\leq max\{UTS1(s, PM_i, appY, appX, N_{iXY}, n_{i2})\} \exists appY \in NappSet$$
$$= UTS2(s, PM_i, appY, n_{i2}).$$ (15)
Therefore, $UTS2(s, PM_i, appx, n_{i2})$ is a monotonous and nondecreasing function of variable $n_{iX}$.

Similarly, since (15) is valid for each type of application $app_k$ in NappSet, $\forall n_{iX} \geq 0$

$$UTS3(s_1, PM_i) = UTS2(s_1, PM_i, appx, n_{i1}) + \sum_{appx \in PM_i \cup \text{appSet}(appx)} UTS2(s_2, PM_i, appx, n_{i2})$$

$$\leq UTS2(s, PM_i, appx, n_{i2}) + \sum_{appx \in PM_i \cup \text{appSet}(appx)} UTS2(s_2, PM_i, appx, n_{i2}) = UTS3(s_2, PM_i).$$

Therefore, $UTS3(s_1, PM_i)$ is a monotonous and nondecreasing function of variable $n_{iX}$.

Similarly, it is easily proven that $UTS4(s)$ is a monotonous and nondecreasing function of variable $n_{iX}$. This completes the proof.

C. POWER CONSUMPTION MODEL

As demonstrated in our previous work [10], a linear power consumption models of the resource utilization makes different container placements undistinguishable with regard to energy consumption in a homogeneous PM environment that has the same numbers of active PMs. However, energy savings can be achieved by optimizing the placement of containers under a nonlinear energy consumption model. Therefore, we adopt a polynomial power consumption model that was proposed by [25], which precisely reflects the influence of both the CPU and memory resource utilization on the server’s power consumption. The power consumption of a single PM $PM_i$ is calculated as follows:

$$\text{pcost}(i) = 155.0537 + 375.855 \times \text{cpu}^{\text{used}}_{PM_i} - 401.0088 \times (\text{cpu}^{\text{used}}_{PM_i})^2 + 164.4327 \times (\text{cpu}^{\text{used}}_{PM_i})^3 - 30.6192 \times \text{mem}^{\text{used}}_{PM_i} + 41.8946 \times (\text{mem}^{\text{used}}_{PM_i})^2 - 19.8122 \times (\text{mem}^{\text{used}}_{PM_i})^3.$$ \hspace{1cm} (17)

$$\text{cpu}^{\text{used}}_{PM_i} = \sum_{VM = 1}^{Cnum} \sum_{j = 1}^{VMnum} \delta_j^i \tau_j^i \text{cpu}^{\text{cont}}_{k}, \quad i = 1, \ldots, PMnum.$$ \hspace{1cm} (18)

$$\text{mem}^{\text{used}}_{PM_i} = \sum_{VM = 1}^{Cnum} \sum_{j = 1}^{VMnum} \delta_j^i \tau_j^i \text{mem}^{\text{cont}}_{k}, \quad i = 1, \ldots, PMnum.$$ \hspace{1cm} (19)

where

$$\delta_j^i = \begin{cases} 1, & \text{if container } k \text{ is placed in } VM_j \\ 0, & \text{otherwise} \end{cases}$$

and

$$\tau_j^i = \begin{cases} 1, & \text{if } VM_j \text{ is placed in } VM_i \\ 0, & \text{otherwise} \end{cases}$$

The power consumption of all the PMs is calculated as follows:

$$\text{PowerC} = \sum_{i = 1}^{PMnum} \text{pcost}(i).$$ \hspace{1cm} (20)

D. BI-OBJECTIVE OPTIMIZATION CONTAINER PLACEMENT MODEL

Based on the above works, we use the following optimization model to formalize the container placement optimization problem.

$$\text{Minimize } \text{PowerC} = \sum_{i = 1}^{PMnum} \text{pcost}(i).$$ \hspace{1cm} (21)

$$\text{Minimize } UTS4(s) = \sum_{i = 1}^{PMnum} UTS3(s, PM_i).$$ \hspace{1cm} (22)

subject to

$$\text{cpu}^{\text{used}}_{VM_j} = \sum_{k = 1}^{Cnum} \delta_j^i \tau_j^i \text{cpu}^{\text{cont}}_{k} \leq \text{cpu}^{\text{total}}_{VM_j}, \quad j = 1, \ldots, VMnum.$$ \hspace{1cm} (23)

$$\text{mem}^{\text{used}}_{VM_j} = \sum_{k = 1}^{Cnum} \delta_j^i \tau_j^i \text{mem}^{\text{cont}}_{k} \leq \text{mem}^{\text{total}}_{VM_j}, \quad j = 1, \ldots, VMnum.$$ \hspace{1cm} (24)

anti-affinity constraints $< appx, appx, N_{iXY} >$ \hspace{1cm} (25)

Notably, constraints (23) and (24) guarantee that the resource requirement of the containers does not exceed the resource limitations of the assigned VM, and constraint (25) guarantees that the container placement satisfies the anti-affinity constraints.

III. OUR ALGORITHM

The container placement optimization problem in this paper refers to assigning containers to suitable PM nodes that satisfy the application anti-affinity constraints and choose the suitable VM nodes that satisfy the multi-dimensional resource constraints. Therefore, it can be described as a bin packing problem. As we know, the bin packing problem is an NP-hard problem, and heuristic algorithms such as the GA owning a good evolutionary mechanism is an effective method to solve this combinatorial optimization problem. However, a randomly generated container placement solution is generally not a feasible solution since it would not always satisfy all constraints. The FF algorithm [26], [27] is a rapid placement algorithm, and it also suitable for container placement. It assigns each container to the first PM that satisfies the application anti-affinity constraints and chooses the first
VM that can provide enough resources for it. Therefore, it can produce only one container placement solution, no matter how many times the experiment is repeated. However, the quality of its container placement solution depends on the container processing sequence. Therefore, the combination of the IGA and FF algorithm may have a better effect. That is, we can use an IGA to randomly generate a group of container processing sequences, use the FF algorithm to conduct rapid container placement based on the given container processing sequence, and then use the evolutionary mechanism of the IGA to optimize the container processing sequence.

A. FF ALGORITHM

In this paper, since the container placement not only needs to satisfy the resource constraints but also needs to satisfy the anti-affinity constraints, the pseudocode of the FF algorithm can be shown as Algorithm 1.

Algorithm 1 FF Algorithm

**Input:** a container processing sequence, VMList, PMList, anti-affinity constraints

**Output:** a container placement solution

1. for each container in the container processing sequence do
2. flag=0;
3. while flag==0 then
4. for each PMj in PMList do
5. if flag==1 then
6. break
7. else
8. for each VMj held on PMj do
9. if the container placement satisfies the anti-affinity constraints on PMj and the VMj has enough resources for the container then
10. assign the container on this VMj;
11. flag==1;
12. break
13. end if
14. end for
15. end if
16. end for
17. end while
18. end for
19. Return a container placement solution

B. CHROMOSOME ENCODING AND POPULATION INITIALIZATION

IGA is a population-based algorithm. Since the IGA is used to optimize the container processing sequence, each container processing sequence needs to be encoded as an individual, i.e., a chromosome, and then a group of individuals can form a population. In this paper, we record the containers’ IDs in the container processing sequence order and set it as its corresponding chromosome. Assuming that the size of the population is *popsize*, then we just need to randomly generate *popsize* individuals to initialize the population.

C. FITNESS AND RANKING

Since one container processing sequence can just correspond to only one container placement solution after using the FF algorithm for rapid placement, we can use the ranking Score of a chromosome in the population as its fitness, i.e., the ranking Score of the corresponding container placement solution in a solution group based on the two objectives PowerC and UTS4. To facilitate decision-making, we use the variant of the modified non-dominated ranking method proposed by [28]. The ranking system is shown as algorithm 2. The smaller the ranking Score of a chromosome is, the better the result.

Algorithm 2 A Ranking System

**Input:** population, objectives (i.e., PowerC, UTS4)

**Output:** rank_Score

1. rank_Score = 0
2. for each pair (chrom_i, chrom_j) do
3. for each objective do
4. if the objective of chrom_i is better than chrom_j then
5. rank_Score [j] +=1;
6. else if the objective of chrom_i is worse than chrom_j then
7. rank_Score [i] +=1;
8. end if
9. end for
10. end for
11. Return rank_Score

D. EXCHANGE MUTATION OPERATION

The crossover and mutation operations of the GA play important roles in the population evolution phase. Note that the resource utilizations are very high after using the FF algorithm for container placement, and this may lead to an inefficient evolution for the original GA. Fortunately, our previous work [10] has showed that the mutation operation we proposed is well suited for solving the container placement problem when the resource utilizations are high. The mutation operation consists of two kinds of exchange operations and one control parameter to select the operation. This operation is used for better realizing the local search. Its evolution is in line with the idea of Iterated Local Search (ILS) algorithm [29], so it can be regarded as a specific implementation of an ILS algorithm. The mutation operation is introduced below.

1) TWO-POINT EXCHANGE MUTATION OPERATION (TPEMO)

The TPEMO randomly selects two different positions of a container processing sequence for exchange (see Fig. 3(a)),

VOLUME 8, 2020
which simulates a lightweight adjustment (mutation) for two containers in a processing sequence. The newly derived processing sequence has little difference from its origin, and it is suitable for local search optimization.

2) TWO-SEGMENT EXCHANGE MUTATION OPERATION (TSEMO)

The TSEMO randomly selects two adjacent genome segments for the position exchange (see Fig. 3(b)). It simulates an adjustment for some pairing containers, which is like a medium level mutation for a processing sequence. The newly derived processing sequence has a significant distinction from the original one, and it is suitable for jumping out of the local optimum.

3) CONTROL PARAMETER

Since the two kinds of exchange mutation operations have their own advantages, a control parameter function named CP(T) is used to control the number of individuals performing the two mutation exchange operations to guide the evolution of the population. The function CP(T) is shown as follows:

\[
CP(T) = 1 - \frac{1}{3} \left( \frac{2T}{MaxT} - 1 \right)^2, \quad T \in [1, MaxT],
\]

where \(T\) is the number of iterations, and \(MaxT\) is the maximum number of iterations. The graph of \(CP(T)\) is shown in Fig. 4. If \(CP(T)\) is larger than a random number range in \([0, 1]\), then TPEMO is chosen; otherwise, TSEMO is chosen. It prefers to perform a small local search in the early stage of the search to make the search near the initial solution more even, and perform more intense mutations to jump out of the local optimal in the mid-stage, and then perform a small local search in the later stage. More details can be seen in [10].

The two kinds of exchange mutation operations are special mutation operations since their operators operate on a single chromosome by changing the value(s) in one or more gene positions of the chromosome. They can also be treated as special crossover operations too since the operators operate on a parent chromosome and keep all the values of the genes and their numbers the same. Therefore, we only use the two kinds of mutation exchange operations in this paper due to their crossover operation and mutation operation properties, respectively.

4) FF-BASED-IGA

We named the algorithm that combines the FF algorithm and IGA as the FF-based-IGA. It makes full use of the rapid placement advantage of the FF algorithm and makes full use of the evolutionary mechanism advantage from the IGA. The flowchart of our proposed algorithm is shown in Fig. 5.

![Image of the flowchart of the FF-based-IGA.](image-url)
are made. The two objectives of the container placement solutions and their fitnesses can be calculated. In step 3, after the fitness calculation, the individual in the population with the best fitness is set as the optimal individual. In step 4, the exchange mutation operation is carried out and a new temporary population is created. In step 5, the corresponding container placement solutions of the new temporary population are made by the FF algorithm, and the two objectives of each individual in the new temporary population are calculated. In step 6, in the population update step, we take just one new individual to replace its original one in the original population for one update comparison between the new and original individuals, and then choose the one with the smallest rank in the original population as the new individual in the next generation population. Meanwhile, the individual in the next generation population with the best fitness is set as the optimal individual. At last, the termination conditions are assessed. If the termination conditions are met, then exit the loop; otherwise, go to Step 4.

IV. PERFORMANCE EVALUATION

In order to better demonstrate the application isolation effectiveness of our method, we evaluate the proposed FF-based-IGA on the container datasets from the Alibaba Competition 2018 and a synthetic anti-affinity constraint dataset. In the experiments, we compare the performance of the proposed algorithm with the 4 other algorithms mentioned in section I. All the algorithms have been evaluated using PyCharm 2019 and were carried out on a 64-bit Windows operating system with an Intel Core i7 CPU and 16 GB of memory.

A. COMPARISON ALGORITHMS

The FF algorithm is the main competitor in practical use, and it only focuses on satisfying the source constraints and anti-affinity constraints. Different from the FF algorithm, the BF algorithm further utilizes the two objective models we proposed to assign the containers one by one. The other two algorithms with a single objective only focus on minimizing the power consumption objective.

The significant difference between the FF and other algorithms is that it does not need any objective to guide the container placement. One significant difference between the FF-based-IGA and the BF algorithm is that the former chooses the best container placement solution from a group of evolutionary solutions and the latter constructs the best container placement solution by assigning containers to their best nodes one by one. One is from a global optimal perspective and the other is from a current optimal perspective. The two algorithms with a single objective are chosen and compared with the two algorithms with bi-objectives and the other to test the effect of the application isolation metric model.

B. THE GENERATION OF ANTI-AFFINITY CONSTRAINTS

The algorithm for generating an anti-affinity constraints data set is shown in algorithm 3. Here, AppNum is the number of all applications; percentage01 and percentage02 are two parameters to control the upper bounds of the interference-risk applications’ percentage with respect to all applications and the percentage of interference-risk applications’ neighboring applications, respectively; and Nn is a constant indicating the maximum number of containers allowed to place on a PM.

Algorithm 3 Anti-Affinity Constraint List Generation Algorithm

\[\text{Input: AppNum, percentage01, percentage02, a constant } N n\]
\[\text{Output: anti-affinity constraint List}\]
\[\text{1 anti-affinity constraint List } \leftarrow \{\}\];
\[\text{2 Set the number of interference-risk applications as } \text{AppNum } \ast \text{percentage01, and then randomly generate an interference-risk app set App_interf_List};\]
\[\text{3 for each appx in App_interf_List do}\]
\[\text{4 Randomly generate a neighbor application set App_neighbor_List and limit its number to (the size of App_interf_List) } \ast \text{percentage02 at most};\]
\[\text{5 for each appy in App_neighbor_sub do}\]
\[\text{6 Randomly generate an integer } N \text{ range in } [1, N n];\]
\[\text{7 anti-affinity constraint List } \leftarrow \text{anti-affinity constraint List } \cup \{< \text{appy, appx}, N >\};\]
\[\text{8 end for}\]
\[\text{9 end for}\]
\[\text{10 Return anti-affinity constraint List}\]

C. PARAMETERS SETTING

The resource specification for each PM is set as \(cpu_{PM}^{total} = 16 \) vcpu and \(mem_{PM}^{total} = 64 \) GiB. The resource specification for each VM is set as \(cpu_{VM}^{total} = 4 \) vcpu and \(mem_{VM}^{total} = 16 \) GiB. The application and container data are taken from Global Scheduling Algorithm Competition 2018 [24], [30]. The synthetic anti-affinity constraint data are published in GitHub (https://github.com/feather077/FF-based-IGA#ff-based-iga). We choose the app IDs from 1 to 150 and deleted the apps requiring resources over 4 vcpu or 16 GiB. Finally, we kept 1023 containers. We set the \(\text{popsize}\) in the FF-based-IGA as 20 and the \(\text{MaxT}\) as 1000.

D. RESULTS AND ANALYSIS

For the sake of fairness, the best ICP solutions of the five algorithms are based on the same collection of randomly initialized container processing sequences. Each experiment is repeated 15 times. In order to show the performance of each algorithm more comprehensively, we added \(MMaxUTS\)2, time cost and the number of active PMs as supplementary comparison indicators. The results of the two objectives and supplementary indicators are shown in Table 1 and Fig. 6.

As shown in Table 1 and Fig. 6, the two algorithms using our proposed IGA searching method (i.e. FF-based-IGA and FF-based-IGA_single) have better performance on power
consumption than the other three algorithms without using our proposed searching method. They achieve at least a 2.14% of power saving gain. This confirms that the combination of FF algorithm with the mutation operation successfully enhances the searching capability by taking advantage of their own advantages.

The two algorithms using our proposed application isolation metric (i.e. BF and FF-based-IGA) have better service performance than the other three algorithms without using our proposed application isolation metric. This confirms that the application isolation metric we proposed is effective. The BF algorithm even achieved the best state of application isolation, since both $UTS_4$ and $MMaxUTS_2$ are 0. Our FF-based-IGA also showed a good performance on application isolation. Its $UTS_4$ is reduced by 67.7% compared with the FF algorithm. The average $MMaxUTS_2$ of the FF-based-IGA is 1.634, and its corresponding average max ACD is 49.1%, which is only about half of the performance tolerance supremum, i.e., 100%.

From the comparison of FF-based-IGA and FF-based-IGA_single and the comparison of BF and BF_single, we can conclude that although the using of our proposed application isolation metric cannot bring a bigger gain on power consumption than the algorithm without using application isolation metric, but the use of our proposed application isolation metric can greatly improve the performance on application isolation.

In the overall view, the FF algorithm has the worst average PowerC, $UTS_4$ and $MMaxUTS_2$ amongst the five algorithms since it does not consider the optimized container placement for the fine-grained isolation of interference-risk applications. The average $MMaxUTS_2$ of the FF algorithm is 2.33, and its corresponding average max ACD is 84.59%, which is much bigger than FF-based-IGA and BF algorithm. However, the FF algorithm has the best time cost due to its simple design. The BF algorithm has the best $UTS_4$ and best $MMaxUTS_2$, both of which are 0; and it achieves the best isolation for the interference-risk applications. The PowerC of the BF algorithm is almost the worst one, but it is still better than the FF algorithm. Moreover, the time cost is the worst of the five algorithms due to its ergodicity search strategy. The BF_single algorithm has a little bit smaller PowerC than the BF algorithm, but its $UTS_4$ is much worse than BF algorithm. The FF-based-IGA has a smaller PowerC than the FF and BF algorithms. Besides, its $UTS_4$ and $MMaxUTS_2$ are slightly smaller than that of the FF algorithm. Its average number of active PMs is 132.6, while that for both the FF and BF algorithms are 135. The evolutionary search from one of the experiments is as shown in Fig. 7. The FF-based-IGA_single has the smallest PowerC and number of active PMs, but its best ICP solution is too “crowded”. Its $MMaxUTS_2$ is 2.71828, and its corresponding average max ACD is 100%, which means that at least one application achieves its performance tolerance supremum. To sum up, the FF-based-IGA achieves a desirable balance between the minimum power consumption and the best service performance. In other words, the application isolation metric we proposed is effective to solve our target problem.

V. RELATED WORK

Quantifying the isolation amongst applications is one way to improve performance [31]–[34]. Notably, Gaurav Somani [34] found that it is difficult to predict the performance when real-time applications run in Xen; therefore, the authors executed various benchmarks simultaneously to evaluate the isolation strategy provided by Xen so as to understand the behavior of the scheduler towards different kinds of workloads and the behavior of the involved application placement in Xen.

Different from the above work, some studies model the impact of performance interference and utilize the models...
to select the best host for workloads according to the value of the interference indicator [13]–[16], [35]. Zhao Hui [13] constructs VM performance models to assess the trend of VM performance degradation. It also models the relationship between power consumption and CPU utilization to minimize PM power consumption and guarantee VM performance. The objectives of [13] are similar to ours; the main difference is that our work improves the application performance through container placement optimization while [13] guarantees performance through VM placement optimization. Fei Xu [15] takes MapReduce as a representative cloud application and proposes a heterogeneous and interference-aware VM provisioning framework named Heifer for tenant applications. Heifer leverages a lightweight performance model to predict the performance of MapReduce applications based on the online measured resource utilization and the captured VM interference. Although the method of modeling performance interference can help to accurately find the best node for an instance, Kartik Joshi [36] states that the performance of containers running in co-located VMs would be unpredictable when some VMs excessively consume the shared hardware resources. It thus proposes a lightweight subscriber-centric mechanism named Sherlock to detect performance interference and it defines a metric named IScore to estimate its impact on cloud services. The work studies container performance detection from the perspective of services for users. By comparison, our work is performed from the view of cloud service providers.

Different from previous work, Luis Carlos Jersak [11] applies different interference measures to VM instances based on the importance of applications, and then introduces an algorithm using an adjustable VM interference threshold to consolidate servers. The adjustable VM interference threshold method allows users to enjoy the benefits of getting a better trade-off between the number of resources and the on-demand performance. The other similar approach can be found in Alibaba’s Global Scheduling Algorithm Competition in 2018 [24]. The competition uses anti-affinity constraints to show the tolerance to application interference effects. These anti-affinity constraints are given based on the historical data observations and the parameter setting experience for given resources. However, it does not address the application isolation optimization. Our work is based on anti-affinity constraints and tries to solve this problem while minimizing the power consumption. Besides, another key difference between our scenario and the Alibaba competition is the adopted instance deployment model. [24] assigns the instances to the suitable PM nodes, where the instances could be containers or VMs, and we use the CaaS deployment model that is proposed in [1], i.e., applications are packaged in containers, containers are running on virtual machines (VMs), and VMs are placed on PMs. Tao Shi [37] propose a TMPSO algorithm for container consolidation, but they are only concerned with the energy efficiency.

VI. CONCLUSION AND FUTURE WORK

In this work, we try to strike a balance between the optimal power consumption and the best service performance for the ICP in CaaS. We describe the container placement optimization problem as a bi-objective optimization problem and design an application isolation metric for a container placement solution (i.e., UTS4) based on anti-affinity constraints. We introduce a nonlinear power consumption model as the other objective model. Then, the container placement optimization problem is converted to find the best container placement solution with the minimum power consumption and minimum UTS4. Next, we propose an algorithm named the FF-based-IGA to find the best container placement solution. It leverages the advantage of the FF algorithm for rapid container placement and the evolutionary mechanism of the IGA for optimizing the container processing sequence. The results of the simulation-based experiments show that our proposed application isolation metric method is effective, and our proposed algorithm achieves a compelling advantage in minimizing the power consumption compared to some classic algorithms.

The current work still needs to be improved in terms of the following aspects. First, more objectives such as low resource waste and high availability should be considered. [38] summarized hot issues and interesting algorithms that can be a great help to our future work. Second, we only consider the homogeneous PMs and homogeneous VMs scenario in order to better verify the effectiveness of our proposed method. Therefore, future work will consider the scenario of heterogeneous PMs, heterogeneous VMs and different demands on the type of operating system. Third, our algorithm was only applied to an ICP problem, and it is unable to support the dynamic relocation scheduling; thus, we hope to extend this work to dynamic container consolidation scenario in the future.

REFERENCES

[1] 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.
[2] S. F. Praghati, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, “ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers,” Softw. Pract. Exp., vol. 47, no. 4, pp. 505–521, Apr. 2017.

[3] 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.

[4] R. Zhang, A.-M. Zhong, B. Dong, F. Tian, and R. Li, “Container-VM-PM architecture: A novel architecture for docker container placement,” in Proc. Cloud Comput. (CLOUD), 2018, pp. 128–140.

[5] F. Paraiso, S. Challita, Y. Al-Dhuraibi, and P. Merle, “Model-driven management of docker containers,” in Proc. IEEE 9th Int. Conf. Cloud Comput. (CLOUD), Jun. 2016, pp. 718–725.

[6] AWS. AWS Fargate. Accessed: Jul. 20, 2020. [Online]. Available: https://aws.amazon.com/fargate/?nc=1_sh_is.

[7] L. Li, T. Tang, and W. Chou, “A REST service framework for fine-grained resource management in container-based cloud,” in Proc. IEEE 8th Int. Conf. Cloud Comput., New York, NY, USA, Jun. 2015, pp. 645–652.

[8] Ö. E. Demirkol and A. Demirkol, “Energy efficiency with an application container,” Turkish J. Electr. Eng. Comput. Sci., vol. 27, no. 2, pp. 1041–1051, Mar. 2019.

[9] R. Zhang, Y. Chen, B. Dong, F. Tian, and Q. Zheng, “A genetic algorithm-based energy-efficient container placement strategy in CaaS,” IEEE Access, vol. 7, pp. 121360–121373, 2019.

[10] L. C. Jerse and T. Ferroto, “Performance-aware server consolidation with adjustable interference levels,” in Proc. ACM Symp., 2016, pp. 420–425.

[11] P. Luma, S. Wang, X. Zou, and D. Cheng, “Performance isolation of data-intensive scale-out applications in a multi-tenant cloud,” in Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS), Vancouver, BC, Canada, May 2018, pp. 85–94.

[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] H. Zhao, Q. Zheng, W. Zhang, Y. Chen, and Y. Huang, “Virtual machine placement based on the VM performance models in cloud,” in Proc. IEEE 34th Int. Perform. Comput. Commun. Conf. (IPCCC), Nanjing, China, Dec. 2015, pp. 1–8.

[14] F. Xu, F. Liu, and H. Jin, “Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud,” IEEE Trans. Comput., vol. 65, no. 8, pp. 2470–2483, Aug. 2016.

[15] R. C. Chiang and H. H. Huang, “TRACON: Interference-aware scheduling for data-intensive applications in virtualized environments,” in Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC), Seattle, WA, USA, 2011, pp. 1–12.

[16] Y. D. Barve, S. Shekh, A. Chhokra, S. Khare, A. Bhattacharjee, Z. Kang, H. Sun, and A. Gokhale, “FECBench: A holistic interference-aware approach for application performance modeling,” in Proc. IEEE Int. Conf. Cloud Eng. (IC2E), Prague, Czech Republic, Jun. 2019, pp. 211–221.

[17] S. A. Javadi and A. Gandhi, “DIAL: Reducing tail latencies for cloud applications via dynamic interference-aware load balancing,” in Proc. IEEE Int. Conf. Autonomic Comput. (ICAC), Columbus, OH, USA, Jul. 2017, pp. 135–144.

[18] Y. Xu, M. Bailey, B. Noble, and F. Jahanian, “Small is better: Avoiding latency traps in virtualized data centers,” in Proc. 40th Annu. Symp. Cloud Comput., Oct. 2013, Art. no. 7.

[19] X. Bu, J. Rao, and C.-Z. Xu, “Interference and locality-aware task scheduling for MapReduce applications in virtual clusters,” in Proc. 22nd Int. Symp. High Perform. Parallel Distrib. Comput., 2013, pp. 227–238.

[20] C. Wang, B. Urgaonkar, A. Gupta, L. Y. Chen, R. Birke, and G. Kesidis, “Effective capacity modulation as an explicit control knob for public cloud profitability,” in Proc. IEEE Int. Conf. Autonomic Comput. (ICAC), Wurzburg, Germany, Jul. 2016, pp. 95–104.

[21] F. Caglar, S. Shekh, A. Gokhale, and X. Koutsoukos, “Intelligent, performance-aware resource management for IoT cloud backends,” in Proc. IEEE 1st Int. Conf. Internet-Things Design Implement. (IoTDI), Apr. 2016, pp. 95–105.

[22] Tiantchi Big Data Research Platform. Interpretation of the Global Scheduling Algorithm Competition. Accessed: Jul. 20, 2020. [Online]. Available: http://dy.163.com/v2/article/detail/DKS1OUG90517C29L.html.
**FA ZHANG** received the B.S. degree in computer science and technology from Jilin University, Xi’an, China, in 2017. He is currently pursuing the M.S. degree with the School of Electronic and Information Engineering, Xi’an Jiaotong University. His research interests include the application of transfer learning in taxation, with focus on machine learning and deep learning research.

**FENG TIAN** (Member, IEEE) 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 Lab of Big Data Analytics and also with the Systems Engineering Institute, as a Professor. He is a member of the Satellite-Terrestrial Network Technology Research and Development Key Laboratory, Shaanxi Province. His research interests include cloud computing, big data analytics, learning analytics, and system modeling and analysis. He is a member of the CCF. He won 2nd class prize of National Science and Technology Progress Award of China (4th position, 2017), 1st class Prize of MOE Science and Technology Progress award of China (4th position, 2015), 1st class prize of Science and Technology Progress award of Chinese Electronical Society (5th position, 2013), and TOP Grade Prize of Teaching Achievement Award of Shaanxi Province award, China (2nd Position, 2017).

**BO DONG** (Member, IEEE) received the Ph.D. degree in computer science and technology from Xi’an Jiaotong University, in 2014. He did postdoctoral research in the MOE Key Lab for Intelligent Networks and Network Security, Xi’an Jiaotong University, from 2014 to 2017. He currently serves as the Research Director with the School of Continuing Education, Xi’an Jiaotong University. His research interests focus on intelligent e-Learning and data mining.