Cluster Computing
https://doi.org/10.1007/s10586-022-03663-7

MO-FreeVM: multi-objective server release algorithm for cluster resource management

Shiyan Zhang[1] · Yuchao Zhang[1] · Ran Wang[1] · Xiangyang Gong[1]

Received: 22 December 2021 / Revised: 11 May 2022 / Accepted: 14 June 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract
With the rise of 5G/6G and cloud computing, cluster management has become increasingly popular. Elastic cluster resources allow cloud clients to dynamically scale their resource requirements over time. Existing researches of cluster schedulers focus on improving resource scheduling speed, increasing cluster utilization, compacting the number of active physical machines (PMs) and time satisfaction function (TSF) within a cluster. The TSF is applied as a time to measure the parallel-VM scheduling problem. However, completing execution time (makespan) of task requests is often neglected, which results in inaccurate scheduling and unreasonable total cost computation. The total cost involves PM cost, migrate cost, and balance cost. To solve the problem of inaccurate scheduling of task requests and total cost billing in cluster management, in this paper, we propose an innovative heuristic algorithm, namely, multi-objective two-stage variable neighborhood searching (MO_STVNS), which aims at minimizing total cost while also considering TSF for active PMs. Moreover, we design a Multi-Objective FreeVM (MO-FreeVM) scheduler based on resource prediction, which incorporates a variety of algorithms to work in collaboration to provide near-optimal resource management for cluster. We evaluate MO_STVNS in different real traces and measure it through extensive experiments. The experimental results show that compared with state-of-art methods, the average total cost and average TSF of MO_STVNS are reduced by 33.75% and 60.67% respectively.

Keywords Cluster management · Cloud · Multi-objective · Virtual machine · Scheduling

1 Introduction

With the rapid development of virtualization technologies [1], Internet of Things (IoT) [2] and artificial intelligence-driven applications [3], massive amounts of data are expected to be transferred among data centers (DCs). Cluster management is crucial in cloud computing, since it usually needs to tackle a complex multi-objective optimization problem that requires considering not only cluster load balancing, cluster utilization, and energy consumption from the machine perspective, but also the execution time of task requests from the user perspective.

Cluster management solutions may involve many conflicting and interactive objectives, which makes the relevant optimization problems nontrivial. Recently, latest related works aiming at improving cluster performance are conducted from different perspectives. Researchers need to obtain optimal values for these objectives simultaneously. For example, an Online VM Prediction based Multi-objective Load Balancing (OP-MLB) [4] employs a proactive approach to place and migrate virtual machines. The goal of this framework is to efficiently utilize oversubscribed cloud environments to reduce power consumption and improve resource utilization while minimizing the risk of Service Level Agreement (SLA) violations. Guerrero et al.

✉ Yuchao Zhang
yczhang@bupt.edu.cn

Shiyan Zhang
zshiyan@bupt.edu.cn

Ran Wang
wangranse@bupt.edu.cn

Xiangyang Gong
xygong@bupt.edu.cn

[1] The State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, No.10 Xitucheng Road, Haidian District, Beijing 100089, China

Published online: 03 August 2022

[5] propose a genetic algorithm approach for optimizing container allocation and resilience management using the Non-dominated Sorting Genetic Algorithm-II (NSGA-II), which is based on the optimization of four objectives, namely, balanced cluster usage, a tight distribution of microservices workload along their container replicas, reduction of network cost, and reliability. Multiopt [6] is proposed to solve the performance optimization problem of Docker container resource scheduling, which considers five key factors: CPU and memory usage of every server, the time consumption of image transmission on the network, association between containers and severs, the clustering of containers. The current state-of-the-art works [5, 7–11] have attempted to optimize tasks assignment within the static clusters from various perspectives. In this paper, we study the dynamic cluster management problem, where the cluster size and composition must be adjusted as needed, and also considering the performance related to task requests. Thus, it leads to the above works cannot directly solve our problem.

In addition, there are also outstanding recent works concentrating on upgrading the performance of task requests. Such as, an efficient heuristic algorithm named Cost and Makespan Scheduling of Workflows in the Cloud (CMSWC) [12] is proposed to solve the workflow scheduling problem, which minimizes execution cost and makespan of workflows simultaneously. Zhou et al. [13] study the workflow scheduling problem of minimizing total monetary cost of executing all tasks and makespan. A fuzzy dominance sort based heterogeneous earliest-finish-time (FDHEFT) algorithm is designed for the workflow scheduling problem in cloud environment. The literature [14–17] have investigated the trade-off between quality of service (QoS) of user requirements and workflow execution costs. However, the existing multi-objective optimization algorithms either consider the goals regarding the utilization, balance, and energy consumption of the cluster from the point of view of machines, or only consider the cost and makespan of tasks execution from the point of view of users. Little literatures focus on improving the execution time of task requests and at the same time maintaining the high performance of machines.

In order to simultaneously improve both cluster performance and users satisfaction, researchers have to meet the following new challenges. Firstly, certain task requests can only run on specific hardware, which increases the interdependence between task requests and machines in a multi-resource heterogeneous cluster [18, 19]. Secondly, it is automatical for the cluster manager to strive the goal of decreasing monetary cost and execution makespan, in order to obtain more profits and ensure QoS. The cost-aware related challenges of task scheduling in cloud computing are categorized based on QoS performance (e.g., makespan and delay), system functionality and system architecture [20]. Thirdly, existing global search metaheuristics are effective in solving task scheduling optimization problems, however they often lead to local optimality and introduce larger time and space complexities.

To address the above challenges, a Multi-Objective FreeVM (MO-FreeVM) framework is proposed, which tackles resources prediction problem for each scheduling cycle while accounting for both servers state performance and users QoS. We deploy MO-FreeVM on both Google Cluster Data (GCD) and Alibaba Traces (AT). Then, a large number of experiments are conducted to evaluate the performance of the proposed MO_STVNS algorithm. The algorithm can effectively reduce the total cost and time satisfaction function (TSF) while ensuring QoS (e.g., capacity and delay). Besides, it maximizes the balance achieved between trade-offs in reducing energy consumption and minimizing waste of resources.

In summary, the key contributions of this paper are as follows:

- We disclose a new problem (i.e., Concurrently considering servers state and users QoS in cluster management issues). Specifically, we design a resource prediction based MO-FreeVM scheduler that integrates multiple algorithms working in concert to provide efficient resource management for clusters.
- We propose a multi-objective variable neighborhood search (VNS [21]) algorithm that minimizes total cost and TSF by using a specific neighborhood structure. The algorithm is a metaheuristic method for solving multi-objective combinatorial optimization problems. It explores the neighborhood of the current solution and then moves to a new solution if and only if the improvement is made. The placement and redistribution of task requests within the cluster is accomplished by designing the MO_STVNS.
- To evaluate the performance of the proposed MO_STVNS algorithm, we compare two different scenarios, i.e., heterogeneous cluster Scenario (GCD) and homogeneous cluster Scenario (AT), also implement another three state-of-the-art algorithms. The experimental results show that the average total cost and average TSF of MO_STVNS are reduced by 33.75% and 60.67% respectively, compared with state-of-art methods.

The rest of this paper is structured as follows. Section 2 presents a brief overview of related works. Definitions of scheduling models and problem formulation are given in Sect. 3. The proposed MO-FreeVM is presented in Sect. 4. Section 5 compares our algorithm with stateof-the-art algorithms by extensive evaluations. At last, Sect. 6 concludes the paper.

2 Related work

The issue of deploying task requests on homogeneous or heterogeneous clusters explored in this article covers the following research topics.

2.1 Single objective optimization

A simple single-objective optimization problem which can be expressed as minimize F(x), where $x \in \mathbb{Y}$, where $\mathbb{Y}$ is the set of constraints. In the environment of heterogeneous virtualized server clusters, an application placement Controller pMapper [22], which achieves the objective of power minimization under the performance constraint of merging power supplies using virtualization mechanisms. Ferdaus etal. [23] present an ACO metaheuristic-based server consolidation mechanism to solve the problem of minimizing power consumption and resource waste in large virtualized data centers. AlloX [24] and Gandivafair [25], optimize for a single scheduling goal and tightly couple their scheduling mechanisms to that goal (e.g., maxmin fairness). However, they cannot be used easily to support more complicated policies.

2.2 VM placement and migration

The process of mapping the VMs to the PMs is called the VMP problem and is known to be NP-hard. Different metaheuristic algorithms are proposed to optimize the placement of virtual machines in cloud computing. A series of intelligent optimization algorithms such as genetic algorithms (GA) [26], simulated annealing (SA) [27] and grey wolf optimization (GWO) [28] are used for energy efficient VM placement. To achieve a balance between the communication overhead and overall throughput, [29] proposes a solution called FreeContainer, which uses a new two-phase algorithm to redistribute containers among hosts. Furthermore, FreeContainer does not require hardware modifications and has been extensively evaluated in real environment. Lv, Liang et al. [30] design an efficient communication-aware worst fit decreasing algorithm to place a set of new containers into data centers, and further explored the conflict between container communication and resource utilization in a data center. The migration of containers has also been extensively studied. A joint computing, data transmission and migration energy cost (JCDME) model is proposed in [31], where the overall energy efficiency is improved by optimizing the virtual elements (VE) allocation in a way that introduces weighting parameters. [32] proposes solutions for live migrating Linux containers, while Pickartz et al. [33] propose the techniques for live migrating Docker containers. Nevertheless, the current VMP works are only handled separately as a dynamic VM placement problem, the practical cluster environment also needs to be combined with task scheduling in order to provide an effective solution for cloud subscribers and providers.

2.3 Optimizing costs and makespan

Cloud computing is an emerging pay-per-use business computing model, and with the rapid growth of cloud computing, a large number of applications have migrated from clusters and physical machines (PMs) to IaaS clouds [34]. The cloud possesses an unlimited number of resources, and consumers can expand or scale down the rented resources according to the requirements of applications [35]. Makespan and cost are two key performance measurement criteria assigned by cloud users and considered by cluster scheduler. Makespan is the time from the beginning till the completion of the sequence of tasks in a workflow. [36] presents a workflow scheduling strategy that combined minimal cost and minimal makespan. Su et al. [37] treat the optimization problem of task scheduling with multiple virtual machines and different pricing models as a convex combination of minimizing makespan and monetary costs, and designed two heuristic algorithms to better match the pricing models and the opaque nature of the cloud. A weightbased energy consumption constraint mechanism is proposed in [38] for heterogeneous computing systems and minimizes makespan by assigning task to the processor that allowed the earliest finish time. While these worked jointly optimize makespan and cost, they ignored simultaneous optimization under multiple QoS constraints.

2.4 Cluster management

Modern organizations operate large clusters that are typically shared among several users and applications. Most of these most advanced cluster managers are already mature with advanced features, such as Borg [39], Kubernetes [40], Medea [41] and YARN [42] allocate resources to applications on demand. Cluster scaling involves intelligent algorithms related to task scheduling and migration, such as grey wolf optimization (GWO) [43], genetic algorithms (GA) [44] and simulated annealing (SA) [45, 46]. However, these intelligent algorithms introduce larger iteration delays. In order to elasticize computing resources and support heterogeneous resource management strategies to respond to the dynamic business volumes of various types of workloads. Medea [41] achieves the goal of global optimization by introducing a two-scheduler design and placement constraints. HTAS [47] makes task scheduling and scaling decisions in a cost-effective

manner, preventing resource waste and over-provisioning. Hydra [48] leverages a federated architecture, while centrally coordinating to ensure that tenants receive the correct share of resources. However, existing cluster management solutions, often ignore the dynamically changing cluster environment where any type of task packaging or task collaboration can change cost efficiency and overall system performance.

We have also investigated the latest works on dynamic multi-objective cluster management. Liu et al. [49] consider four optimization objectives for virtual clusters and data centers, namely availability, energy consumption, average resource utilization and resource load balancing, and propose an evolutionary algorithm to weigh these four optimization objectives. Li et al. [50] present a dynamic multi-objective optimized replica placement and migration strategy for edge cloud-based SaaS applications. Ji et al. [51] have designed an adaptive ranking multi-objective differential evolutionary algorithm to solve dynamic multi-objective non-linear equations and find optimal solutions. Under time-based server maintenance in cloud data centers, Patel et al. [52] propose an online truthful double auction technique for balancing multi-objective trade-offs between energy, revenue and performance in IaaS clouds. Devi et al. [53] have used a genetic algorithm based on coded chromosomes (GEC-DRP) to manage dynamic resource scheduling. However, the above works have only studied a part of the problems in dynamic cluster management. There are no comprehensive analyses in terms of constraints, algorithm complexity, and prediction of PM start-stop in dynamic clusters.

The MO-FreeVM framework is a pragmatic and challenging solution compared to existing works. MO-FreeVM develops and integrates all required operations into a unified platform, allowing them to interact and optimize each other to improve the overall performance of the cloud data centers.

3 Problem formalization

Existing studies of task scheduling and virtual machine placement in cluster management often lack comprehensive optimization regarding the performance of clustered PMs and the makespan of task requests. In this paper, we consider the energy consumption associated with VM placement/migration while considering makespan for tasks scheduling. Therefore, we view this study as a multi-objective optimization problem with the objective of minimizing total cost and time satisfaction function for active PMs.

3.1 Problem definition

In this subsection we describe the task model and the cloud resource management model. In addition, a detailed description of all indices, input parameters and decision variables are listed in Table 1.

3.1.1 Task model

The two types of workflows involved in the data center are Long-running services and batch jobs. A workflow can be described by the task instance model and each task is configured to a single virtual machine. A task is equivalent to a VM request, and each task needs to be placed on an appropriate PM. Assuming that arbitrary N task requests arrive per scheduling cycle, the set of task requests is defined as $\mathscr{TR} = \{TR_1, TR_2, ..., TR_i, ..., TR_N\}$, where $i \in [1, N]$ and N is the total number of task requests. Each task request TR_i is described as $TR_i = \langle sz_i, dl_i, st_i \rangle$, where sz_i, dl_i and st_i represent the size of task request that is measured by million of instructions (MI), task request deadline and start time of task request, respectively.

3.1.2 Cloud resource manage model

Our cloud data center platform consists of heterogeneous virtual machines (VMs) and PMs, which are similar to Amazon Elastic Compute Cloud (EC2) for scheduling different types of workflows. Moreover, VMs are configured with different resource parameters, and they can be placed in different PMs . Each VM is characterized by its own CPU and memory configuration. Assuming that each task T_j is configured to a VM, the processing time of the task T_j assigned to a particular VM is negligible. Set of VMs is defined as $\mathscr{VM} = \{VM_1, VM_2, \ldots, VM_v, \ldots, VM_N\}$, where $v \epsilon [1, N]$ and N is the total number of VMs. Set of PMs is defined as $\mathscr{PM} = \{PM_1, PM_2, \ldots, PM_k, \ldots, PM_M\}$, where $k \epsilon [1, M]$ and M is the total number of PMs. The problem of resource management in cloud platforms is the process of placing different types of VMs into the appropriate PMs. On the basis of the problem description, we introduce the MO-FreeVM problem in detail.

3.2 Total cost

Previously related work, cost usually refers to the total cost that a user needs to pay to the cloud provider to rent a virtual machine. Nevertheless, a range of costs such as the cost of booting the machines in the cluster and VM migration cost are often ignored. Hence, in this paper, we have quantified the total cost of VM allocation in terms of

Table 1 Summary of parameters and their descriptions

Variable	Definition
M	Number of PMs in the datacenter
N	Number of task requests
$P_{on}(t)$	The duration of the enabled PM
$D_v^r(t)$	The migration time of resource r within resource request v
$P(v)$	PM allocated for resource request v
$P^{'}(v)$	PM reallocated for resource request v
$f(v)$	Binary parameter: 1 if resource request v is still on the same PM after migration, 0 otherwise
T_{make}	the time of the maximum active PMs
T_{ave}	the average time of all active PMs.
α	the weight parameter of T_{make}
β	the weight parameter of T_{ave}
t_j^{com}	the completing execution time of PM j
sz_i	the size of task instance i
C_{ij}^{pow}	when task i is placed to PM j, the computing power of PM j
E_{ij}^{com}	the expected time for PM j to process VM i
z_i	The PM to execute the ith virtual machine
i, k	Task requests indices; i,k = 1,2, …, N when i or k=0, 0 represents the dummy node
Y_{ijk}	Binary parameter: 1 if VM i immediately produces before VM k in PM j, 0 otherwise
δ_{ij}	Binary parameter: 1 if task request i is assigned to PM j, 0 otherwise
A	Unit time cost of processing with PMs

two components, which are PM cost and QoS cost. Among them, QoS cost includes migrate cost and balance cost.

3.2.1 PM cost

When a batch requests arrive, all PMs have Active/Inactive status. A PM in On state consumes power, here, we convert time to billing. When the PM is On, it charges C_{on} per minute; PM in Off state is not billed. The PM cost is then expressed as:

$$Pcost = C_{on} * |P_{on}(t)| \tag{1}$$

3.2.2 Migrate cost

Real-time migration of virtual machines makes it possible to transfer a VM from the source node to the destination node with minimal downtime and hang-up time. Although dynamic migration is transparent to end users, it may result in resource-related performance degradation of applications running in the VM. If two CPU-hungry VMs are placed on the same PM, the resource utilization on the PM may be unbalanced, e.g., there is very little CPU left, but a lot of memory left. If the new VM is not placed on this PM with unbalanced resources, it cannot degrade the performance of any entity. To quantify migration costs, we have:

$$\begin{aligned} Mcost &= C_M * \sum_{r\in R}\sum_{v\in V} D_v^r(t) * f(v) \\ f(v) &= \begin{cases} 1, & P(v) \neq P^{'}(v) \\ 0, & P(v) = P^{'}(v) \end{cases} \end{aligned} \tag{2}$$

$D_v^r(t)$ indicates the migration time of resource r within resource request v. $f(v)$ is used to determine whether resource request v is still on the same PM after migration. C_M denotes the migration overhead per unit of time. $P(v)$ denotes the PM assigned to resource request v. $P'(v)$ denotes the PM reassigned for resource request v.

3.2.3 Balance cost

If the PM has no available resources for future VM allocations or upcoming requests, so the residual amount of multiple resources should be balanced. Balance Ratio [54] represents the target ratio of resource r_i and resource r_j. In addition, the Balance Ratio dynamically varies per cycle.

$$
\begin{aligned}
Bcost &= \sum_{(r_i,r_j),\forall r_i \neq r_j} cost(r_i, r_j) \\
cost(r_i, r_j) &= \sum_{p \in \mathscr{PM}} |p(r_i) + vmr_i - (p(r_j) + vmr_j) * \\
&Balance\_Ration| - |p(r_j) - p(r_j) * Balance\_Ration| \\
Balance\_Ratio &= t(r_i, r_j)
\end{aligned} \tag{3}
$$

$\mathscr{PM}$ denotes the set of PMs, p denotes any machine within $\mathscr{PM}$, p(r_i) denotes the target ratio of resource r_i on PM p, and vmr_i denotes the size of resource r_i on VM. Similarly, p(r_j) denotes the target ratio of resource r_j on PM p, and vmr_j denotes the size of resource r_j on VM.

According to the above definitions, the objective function to be minimized can be defined as the weighted sum of all the costs, i.e.,

$$Cost = w_P * Pcost + w_M * Mcost + w_B * Bcost \tag{4}$$

Generally speaking, the proportion of different costs are adjusted to the actual requirements and the optimal VM scheduling is chosen by setting different weight coefficients (e.g. w_P, w_M and w_B), while ensuring that the total cost is minimum.

3.3 Time satisfaction function

We define time satisfaction function (TSF) as the time used to measure the parallel-VM scheduling problem, consisting of T_{make} and T_{ave} two components, where T_{make} refers to the maximum makespan of all active PMs and T_{ave} refers to the average makespan of all active PMs. Since these two parameters can be measured in the same units (time units), we have:

$$TSF_{ij} = \alpha T_{make} + \beta T_{ave} \tag{5}$$

α indicates the weight parameter of T_{make} and β indicates the weight parameter of T_{ave}. In the following, Eq. (5) is introduced in Eq. (11).

The maximum sum of E_{ij}^{com} of all active PMs , we have:

$$T_{\text{make}} = \max\left\{ \sum_{[i|z_i=j]} E_{ij}^{com} \right\} \tag{6}$$

E_{ij}^{com} denotes the expected time for PM j to process VM i. $[i \mid z_i = j]$ represents the VMs assigned to PM j.

As T_{ave} has higher order of magnitude over T_{make}, it is normalized by M, we have:

$$T_{ave} = \frac{\sum_{j=1}^{M} t_j^{com}}{M} \tag{7}$$

The expected execution time E_{ij}^{com} for all VM i assigned to PM j are calculated as follows:

$$E_{ij}^{com} = \sum_{i \in N} \frac{sz_i}{C_{ij}^{pow}} \tag{8}$$

sz_i denotes the size of the task i, measured in Million Instruction (MI); the computing power of PM j is defined as C_{ij}^{pow}, computing power is equal to the processing rate of task i by PM j, measured in million instruction per second (MIPS).

Based on the above analysis, calculating the makespan of all submitted task requests, we have:

$$t_j^{com} = \sum_{[i|z_i=j]} E_{ij}^{com} \tag{9}$$

$$z = \{z_1, z_2, \ldots, z_N\}, \forall z_i \in [1, M], \forall i \in [1, N], \forall j \in [1, M] \tag{10}$$

t_j^{com} indicates the completing execution time of PM j.

3.4 Multi-objective model

Dynamic cluster management involves a variety of conflicting objectives that should be optimized simultaneously. Based on the above definitions and models, dynamic cluster management can be represented as a multi-objective optimization problem with multiple decision variables and objectives. We aim to minimize the total cost Equation (4) as well as the time satisfaction function (Eq. 5) of active PMs as our optimization objectives, have the following formal definitions:

$$\text{Minimize } : \mathbb{Z} = \sum_{j=1}^{M} \sum_{i=1}^{N} \text{Cost}_{ij} \delta_{ij} + \sum_{j=1}^{M} \sum_{i=1}^{N} A \times TSF_{ij} \tag{11}$$

Subject to:

$$\sum_{i=1}^{N} Y_{0ij} \leq 1, \forall j \tag{12}$$

$$\sum_{j=1}^{M} \sum_{i=1, i \neq k}^{N} Y_{ijk} = 1, \forall k \tag{13}$$

$$\sum_{k=1, i \neq k}^{N} Y_{ijk} \leq \delta_{ij}, \forall i, j \tag{14}$$

$$\sum_{i=1,i\neq k}^{N} Y_{ijk} = \delta_{kj}, \forall k, j \tag{15}$$

$$\sum_{j=1}^{M} \delta_{ij} = 1, \forall i \tag{16}$$

Note that in Eq. (11) contains the multiplication of two decision variables, f(v) and δ_{ij}. Considering the heterogeneity of PMs and the stochastic nature of VM requests, the problem is transformed into a multi-objective non-linear programming problem. Specifically, a heterogeneous cluster contains multiple PMs with different hardware configurations (e.g. number of CPU cores). The different task requests may be independent of each other or have dependencies.

The objective function in Eq. (11) attempts to minimize total cost used to schedule all VMs and the time satisfaction function cost per working hour. Equations (12) and (13) illustrate that all VMs must be allocated to at most one PM, and the start of all allocated VMs must be VM 0 (dummy VM). Equation (14) is the relationship constraint of (Y_{ijk}) and (δ_{ij}). Equation (15) verifys that VM i is processed before VM k in PM j only when VM k is allocated to PM j. Equation (16) assures that a VM can be allocated to at most one PM.

3.5 Constraints

To minimize $\mathbb{Z}$, VMs should be placed or reassigned to the most appropriate PM, but this process should satisfy the following six strict constraints.

Constraint 1: (Capacity) In each PM, resource assignment cannot exceed its capacity.

Each cluster usually contains a limited number of PMs. For the resources (cpu, memory, etc.) of each PM, the total amount of resources allocated to all the VMs cannot exceed the resource capacity of the PM. Assume that a VM request includes two resource types (CPU, memory). Let the sets of VMs and PMs be denoted by V and P, respectively. Without loss of generality, let $V = \{v_1, v_2, \ldots, v_N\}$ and $P = \{p_1, p_2, \ldots, p_M\}$. For each requested VM v, let α_v be the number of CPUs required and let β_v be the memory requirement (in GiB). For each PM p, let C_p be the number of CPUs it can support, M_p be the amount of memory (in GiB). In addition, each $v \in V$ and each $p \in P$, let be x_{vp} the binary assignment variable, which takes the value 1 if VM v is assigned to PM p and 0 otherwise. The following constraints are required:

$$\sum_{p\in P} x_{vp} = 1, \forall v \in V \tag{17}$$

$$\sum_{v\in V} \alpha_v x_{vp} \leq C_p, \forall p \in P \tag{18}$$

$$\sum_{v\in V} \beta_v x_{vp} \leq M_p, \forall p \in P \tag{19}$$

Among them, (17) ensures that every VM is assigned to exactly one PM. (18) and (19) are the constraints on the resource capacity of the number of CPUs and the total memory size of each PM p, respectively.

Constraint 2: (Re-execute) When task instances are migrated to a new PM, they need to be re-executed. To calculate the impact of migration on the execution of task instances, here we define the following migration time as follows:

$$et_{ij} = \frac{sz_i}{ps_j} \tag{20}$$

where et_{ij} indicates the execution time of the task instance i on p_j and ps_j is the processing speed of p_j.

$$mt_i = v^i_{FT} - v^i_{ST} \tag{21}$$

$$mt_i + et_{ij} \leqslant dl_i \tag{22}$$

where v^i_{FT} indicates the migration finish time of the running task instance i, v^i_{ST} represents the migration start time of running task instance i and mt_i means the time overhead of one instance due to migration.

Constraint 3: (Transfer) Time-sensitive services (such as long-running services) have latency requirements for critical data transfers between VMs.

To meet the ultra-low latency requirements, VMs with critical frequent interactions should be assigned to the same PM. Let $\zeta(v, v')$ denote whether v and v' should be colocated on the same PM, with $\zeta(v, v') = 1$ indicating yes and $\zeta(v, v') = 0$ otherwise. The transfer constraint can be expressed as:

$$\zeta(v, v') = 1 \Rightarrow P(v) = P(v'), \forall v, v' \in P, v \neq v' \tag{23}$$

P(v) denotes the PM assigned to VM v , $P(v')$ denotes the PM assigned to VM v'.

Constraint 4: (Transient) During the migration process, the VM instance will not be destroyed before it is established on the new PM.

The reassignment of a VM is achieved through a live migration, which means that the VM is transferred to the final machine while keeping it running on the original one. Such resources (e.g., CPU and memory) are needed on both machines (initial and final machines) during a live migration, as the processes use the resources on both machines during the reassignment.

Constraint 5: (Spread) Safety-related services need to assign their VMs to different PMs.

A specific function in a high performance application is usually implemented on multiple VMs to support concurrent operations. Since some VMs are CPU sensitive, but not sensitive to memory, they cannot be placed on the same PM , otherwise other resource (such as memory) would be seriously wasted.

Constraint 6: (Trait) The task request for the specified traits can only be placed on the PM that the traits match.

4 MO-FreeVM

In this section, we present the design of MO-FreeVM. The goal of MO-FreeVM is to find a VM redistribution solution that minimizes the overall cost (Equation (4)) and TSF (Equation (5)). MO-FreeVM is consisted by two stages, first placement (FP) and variable neighborhood searching (VNS), collectively known as multi-objective two-stage variable neighborhood searching (MO_STVNS). FP entails simple and fast placement of VMs, along with placing as many user requests as possible; VNS is aimed to reduce the number of running PMs, which is mainly to vacate the "cold" PMs and process the VM on the "hot" PMs, so as to improve the resource utilization of PMs.

4.1 System architecture

In the previous practice, one scheduling mode is mostly used, but it is not good enough to minimize the number of active PMs while improving the resource utilization of PMs. Our design scheme proposed two modes of primary scheduling and secondary scheduling, and added resource prediction module at the same time. Although the added module brings time complexity to the design and development, it can be found through comparative analysis that the solution in this paper can more accurately predict the number of task requests arriving in each scheduling cycle and improve the resource utilization.

Figure 1 shows the relationship between the core modules of the cloud scheduling platform, as explained below:

(1) Heterogeneous resource requests include task requests for long-running services and batch jobs, where the amount of data requested is different for each scheduling cycle and is specified by using a metadata file for the name, type, VM image and number of resources requested.

(2) The parameter configuration module is used to store and read the status information of the PMs, which include the position of the VMs in the PMs and the active/inactive status of the PMs.

(3) The status information of PMs and task requests in each scheduling cycle are initially delivered to the first placement (FP) module. The fewest number of active PMs are exploited to place as many vm (requests) as possible to achieve energy saving and thus obtain the minimum placement cost. The primary scheduling module is Virtual Machine Placement (VMP), a VMP scheme where each virtual machine can only be placed on one PM.

(4) The secondary scheduling module refers to Virtual Machine Migration (VMM), which aims to find a VM redistribution scheme to maximize the reduction of the total cost and TSF. MO_STVNS uses Shift, Swap, and Replaces to perform neighborhood search. Eventually, the effect of simultaneously compacting "hot" and "cold" PMs are achieved, which frees up underutilized machines and further reduces energy consumption.

(5) Determining which PMs are active and which are inactive from the secondary scheduling results. The PM active/inactive module records the status of all PMs and the billing of PMs.

(6) The resource prediction module predicts the number of resource requests for the next cycle based on the number of historical resource requests released before the current cycle, further predicts how many resources will be released in the current cycle.

(7) The MO-FreeVM scheduler module is used to obtain all PMs and task requests information, execute primary and secondary scheduling, PM active/inactive policies, and calculate the cost for each scheduling cycle.

4.2 First placement

Each resource request contains two attributes in addition to the basic information (ID number): qualitative attributes (characteristics, e.g. the operating system is Windows/

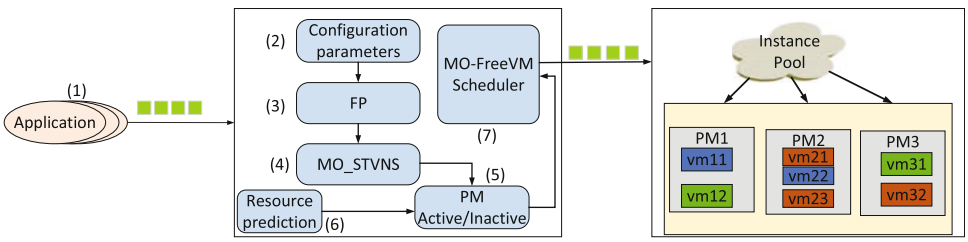


Fig. 1 MO-FreeVM framework

Linux) and quantitative attributes (resources, e.g. CPU requirements). Accordingly, each PM also possesses explicit qualitative attributes, the current status (active/inactive) and the current amount of resources remaining. The FP algorithm requires to locate the PM that corresponds the qualitative and quantitative attributes of the task request and then placing the resource request to the selected PM. For a given VM request, there are generally multiple PMs that satisfy the requirements. Furthermore, for a given PM, there are usually multiple VM requests competing for available PM resources simultaneously. Therefore, we need to consider how to deploy these VM requests so that VMs placement can be completed as quickly as feasible. To reduce the amount of work for subsequent VNS, we strive not to use underutilized PMs as much as possible during the FP.

The objective of FP is to place as many task requests (especially from batch jobs) into the PMs as quickly and as much as possible. Firstly, the long-running services and batch jobs are sorted in positive order according to the size task requests. Secondly, a set of valid PMs are obtained according to the principle of consistency between VM requests and PMs qualitative attributes, and then the PMs are ranked from "hot" to "cold" according to the lowest utilization of the multiple resources. Finally, in order to ensure that more VM requests are placed on the most appropriate PMs, we place sequentially ordered requests one by one on the "hottest" PMs corresponding to them. The above description is the FP algorithm. The one-stage (FP) of MO-FreeVM that we have designed can be applied to various VMP algorithms. The purpose of first placement is to optimize a batch of candidate PMs, thus diminishing the search space for VNS.

4.3 Variable neighborhood searching

Considering that pure VNSs are still susceptible to local optima, we are motivated to design a two-stage variable neighborhood search algorithm to further improve the global search capability and overcome local optima. Our first-stage algorithm (First Placement) reduces the search space for the second-stage algorithm (Variable Neighborhood Searching) by quickly vacating underutilized PMs, thus improving the second-stage search capability.

Variable Neighborhood Search is a meta-heuristic method proposed a few years ago [55], which is based on a simple principle: the neighborhood is systematically changed during the search process. It has evolved very rapidly, with dozens of papers published or forthcoming. Many extensions have been made, mainly to solve large problem instances. In most schemes, there is an effort to maintain the simplicity of the basic scheme. [56] proposed a VNS algorithm for the task assignment problem with constraints and compared the performance with other local search algorithms. [57] proposes a hybrid genetic algorithm and variable neighborhood search algorithm to reduce the total cost of task execution without increasing the maximum completion time of the system. The VNS algorithm has received relatively little attention when studying the scheduling problem of VMs in cluster management. The MO_STVNS proposed in this paper aims to consider the VMs scheduling problem in dynamic cluster management, and we study the multi-objective optimization problem with the objective of minimizing the total cost and time satisfaction functions. Simultaneously, it is further demonstrated that our MO_STVNS algorithm can find the minimum number of active PMs.

The MO_STVNS is able to find the optimal solution in the current neighborhood and to jump out of the current neighborhood to find a better solution. Therefore, it has a high probability of converging on the global optimal solution as long as the neighborhood structure is set appropriately. MO_STVNS with three neighborhood structures, in the case of the same initial solution, to deepen the search space by means of a two-level search, if the optimal solution can be found in the current neighborhood, then update the current solution and jump out of the local optimal solution (See Algorithm 1 for details). In the process of continuous optimization, the MO_STVNS algorithm quickly finds the approximate global optimum solution by successive iterations (see Algorithm 2 for details).

After first placement is accomplished, the resource utilization of the overall cluster is first estimated. When the average resource utilization of PMs surpasses a certain threshold and satisfies constraint, the VNS algorithm is triggered to tune the migration of VMs that have been placed VMs on the PMs, liberating the underutilized PMs to decrease energy consumption, while making the residual resources on the PMs more balanced to promote the total resource utilization of the cluster.

As Fig. 2 illustrates the scheduling process for VM requests, we have selected three PMs available for placing VM requests. The VMs redistribution is a process of migrating as many VMs as possible from the "cold" PMs to the "hot" PMs, which relieves the "cold" PMs and thus saves energy. Our suggested VNS algorithm is employed for VMs migration, and the VNS search process covers three solutions: shift, swap, and replace. As shown in Fig. 2, shift means reassigning a VM from one PM to

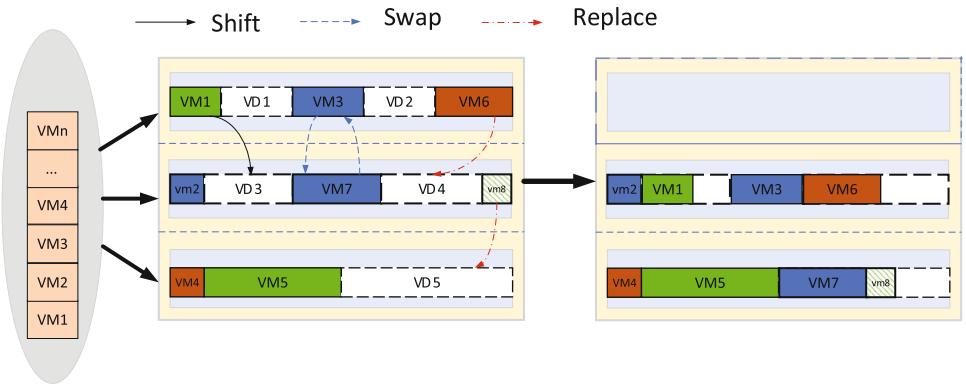


Fig. 2 The instance of scheduling with n vms

another; swap means exchanging two VMs on different PMs; replace means to have attempted to move a zombie VM8 from PM2 to PM3 and then reassign VM6 from PM1 to PM2.

The neighborhood is the crucial to the local search algorithm. Generally speaking, the neighborhood is defined as the space of all solutions obtained by performing a predefined move. Different neighborhoods can be constructed depending on the move defined. Three neighborhoods are used in our proposed variable neighborhood search process, which are: shift neighbourhood, swap neighbourhood, replace neighborhood.

1. *Shift* A shift move describes the reallocation of a VM from one PM to another PM. The shift neighbourhood is defined as the set of reallocation schemes that can be achieved by a shift move. A VM is randomly picked from an existing VMs allocation scheme (the current solution) to place it into the vacant degree in another PM, forming a solution within the shift neighbourhood, which is then continuously optimised. The vacant degree (VD) is the region in a PM available for placing VMs, a PM may have more than one VD. In addition, when placing VMs in the VD of a PM, it is essential not to cause any overlap. As shown in Fig. 2, VD1–VD5 are able to place VM requests, and VD5 is the largest VD. Since the upper bound on the fundamental number of shift neighbourhood is O(N). Therefore, the time complexity of the local search iterations is upper bounded by $O(N * \tau)$ when using shift neighbourhood. τ is the time spent exploring an adjacent solution which is approximately equal to the time spent in the continuous optimization process.
2. *Swap* A swap move refer to the exchange of two VMs allocations on different PMs. The swap neighborhood is defined as the set of reallocation schemes that can be achieved by a swap move. A solution in the swap neighborhood is formed by swapping the placement of two similar VMs in a given solution and then continuously optimizing. After sorting all VMs by size, then the two adjacent VMs are of similar size. VMs with the same or similar size are not recommended to take the swap move, as swapping their placement does not change the solution. As shown in the figure, the neighborhood of the solution is obtained by swapping VM3 and VM7 , and then the optimal solution is obtained by successive optimization. Depending on the distribution of VMs, the cardinal number of swap neighborhood varies from O(N) to O(N^2). Therefore, the time complexity of the local search iteration when using the swap neighborhood is from O($N * \tau$) to O($N^2 * \tau$).
3. *Replace* The replace neighborhood is a subset of the shift neighborhood. When using the replace neighborhood, the neighborhood of a given solution is optimized by picking a small VM from it and putting that VM back to one of the largest vacant points, and then continuous optimization. The upper bound of the cardinal number of replace neighborhood is O(N^2), and the upper bound of the time complexity of the local search iteration is O($N^2 * \tau$).

VNS is a two-level search algorithm that attempts to vacate underutilized PMs one by one from “cold” PMs to “hot” PMs, while ensuring all VMs placed on a PM are vacated at minimal cost. When the total overhead of vacating all the VMs placed on a PM is less than the total revenue, the operation to vacate the PM is performed, otherwise the vacating operation is stopped. The one-level

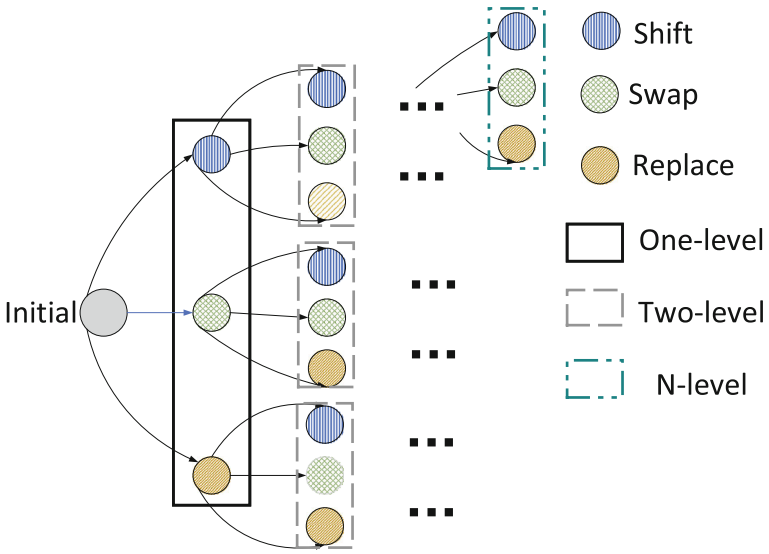


Fig. 3 The process of variable neighborhood search

search is devised to tackle "cold" PMs, i.e., to migrate VMs on a "cold" PM to a "hot" PM. However, the secondary search is to further condense the "hot" PMs. Therefore, VNS is conceived to pursue the second-level search on the basis of the primary search and eventually identify the optimal solution within the two-level search neighborhood.

As we have analyzed above, the upper bounds on the time complexity of the local search iterations using three neighborhoods (shift neighborhood, swap neighborhood, replace neighborhood) are $O(N * \tau)$, $O(N^2 * \tau)$ and $O(N^2 * \tau)$ respectively. Thus, the upper bound on the time complexity of the VNS process is an upper bound on the time complexity of the three candidate neighborhoods: $O(N^2 * \tau)$. While the upper bound on the time complexity of MO_STVNS is expressed as an upper bound on the time complexity of the process: $O(N^3 * \tau)$.

VNS is a modified local search algorithm, as illustrated in Fig. 3, which first declares N levels of neighborhood for the preliminary solution; then the search is performed using the neighborhood structure(e.g., one-level) until a local optimal solution is discovered. The variable neighborhood search algorithm that we have designed is composed of a two-level search to seek the optimal solution in three branches, and then pick the optimal one of them, each branch is a two-level search procedure. There are three moving approaches that can be applied to neighborhood searching, which are shift, swap, and replace. The primary level of the two-level search algorithm is a stochastic move of the current VMs which begins with the current placement, and it explores all three moving approaches. The second level of search allowed another move based on the outcomes of the first level of search, which is not random, but indicated the solution that has the least cost of each possible moving scheme and that enables the total cost of the two-level search less than the cost of the first level of search. Once such a settlement is achieved in the current movement, it is returned directly as the optimal solution for this branch, without any further consideration of the residual moves.

The pseudo-code of multi-objective two-stage variable neighborhood searching (MO_STVNS) is shown in Algorithm 1. Line 2-4 represent a random selection of placement scheme s' in one of the neighbors of the current placement state s (e.g., shift). when a neighborhood change is employed, TSF is calculated based on Equation (5). From line 5,6 indicates doing a secondary search on top of the primary search. Find the local optimal solution s'' in a kind of neighborhood of s' (e.g., shift) for that neighborhood. Where lines 7-13 explore the optimal solution, and if s'' is better than s', then s' is returned as the optimal solution of this branch and update the TSF at the same time; otherwise the other neighborhoods of s' are searched for a solution better than s'. Lines 15-20 show that if there is no better solution than s', then s' is returned as the optimal solution of this branch. Meanwhile, the TSF of the current solution as Max_TSF.

Algorithm 1 MO_STVNS Algorithm

Require: First-Placement scheme s
Ensure: Best found solution

$best \leftarrow s, Max\_TSF \leftarrow \infty$
for each neighborhood $N_i(s)$ of solution s **do**
  $s^{'} \leftarrow$ randomly choose a new solution in $N_i(s)$
  $TSF \leftarrow \alpha T_{make} + \beta T_{ave}$
  for each neighborhood $N_j(s^{'})$ of solution $s^{'}$ **do**
    $s^{''} = FindBest(N_j(s^{'}))$
    if $Cost(s^{''}) < Cost(s^{'})$ and $TSF < Max\_TSF$ **then**
      $s^{'} = s^{''}$
      $Max\_TSF = TSF$
      break
    else
      $j = j + 1$
    end if
  end for
  if $Cost(s^{'}) < Cost(s)$ **then**
    $best \leftarrow s^{'}$
  end if
  if $TSF > Max\_TSF$ **then**
    $TSF \leftarrow Max\_TSF$
  end if
  $i = i + 1$
end for
return best

4.4 PM start-stop

At the end of the VNS algorithm, some underutilized PMs are vacated. In order to save energy, the vacated PMs need to be shut down. As more and more VM requests come in, randomly to deactivate the vacated PMs or to activate more of PMs. This part of the start-stop policy that we have described determines the state of the PMs (active/inactive).

First, the VNS algorithm generates a list of the vacated start-up PMs. The start-stop policy then uses a custom Slidingk algorithm [32] to predict how much resource R_l will be released on the PM at the end of the current cycle and how much of the customer's task request R_q will be reached in the next cycle. Assuming that the total resource of the current remaining PMs is R_m. If $(R_m + R_l)$ is greater than R_q, it indicates that the current machine resource can satisfy the request in the next cycle. It is also a waste of power to turn on vacated PMs, so we need to shut down some of the vacated PMs, with the resources on the shut down PMs are not available. Since some PMs in the active PMs list are selectively shut down, the total resources of these active PMs should be less than $|R_q - R_m - R_l|$. Once $(R_m + R_l)$ is less than R_q, it indicates that the resources of the current machine are not available for the next cycle , that it is not possible to shut down the vacated PMs. Instead, we need to reactivate a batch of PMs to satisfy the requests for the next cycle.

4.5 The proof of closest lower bound for the number of active PMs

In the process of optimizing the solution, it is usually allowed to produce infeasible solutions in the intermediate process. Specifically, we introduce the evaluation function $\left\langle H_{\text{com}}^{lb}, \mathbb{E}(X) \right\rangle$ for finding the optimal feasible solution. In the following, we describe the two components of the evaluation function in detail.

Given a rectangular container $\mathbb{S}$ (Space for active PMs) of width $W \geqslant w_n$ and the height of $\mathbb{S}$ is H (number of active PMs). In this context, H is a number, not a set. The set of VM requests is defined as $\mathscr{VM} = \{VM_1, VM_2, \ldots, VM_i, \ldots, VM_N\}$ with fixed width $w_1, w_2, \ldots, w_i, \ldots, w_N$, where $i \in [1, N]$. The set of PMs is defined as $\mathscr{PM} = \{PM_1, PM_2, \ldots, PM_p, \ldots, PM_M\}$, where $p \in [1, M]$. Without loss of generality, Let the side of $\mathbb{S}$ be

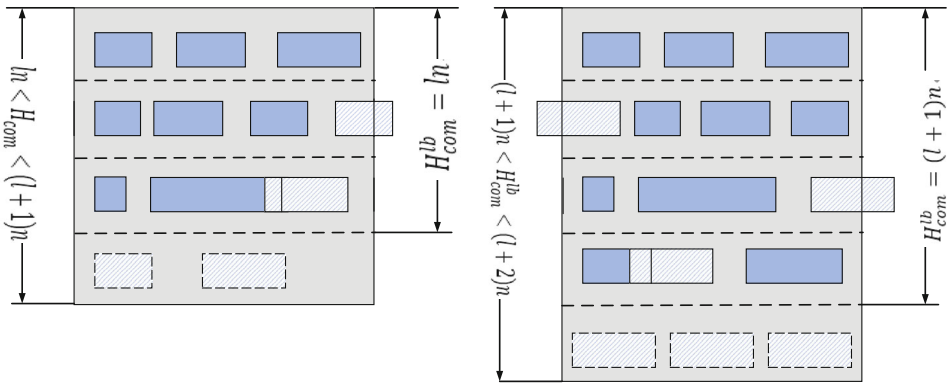


Fig. 4 The H^{lb}_{com} of solution $\mathbb{S}_i$ is ln, the H^{lb}_{com} of solution $\mathbb{S}_j$ is (l+1)n

parallel to the coordinate axis, and let the coordinate of the center of VM_i is (x_i, y_j).

$\mathbb{E}$(X) is an energy function that measures the infeasibility of the optimization process, where X is its configuration denoted by $(x_1, y_1, \ldots, x_i, y_i, \ldots, x_n, y_n)$. The $\mathbb{E}$(X) is defined as follows:

$$\mathbb{E}(X) = \sum_{p=1}^{M}\sum_{i=1}^{N}\frac{1}{2}V_{pi}^2 + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\frac{1}{2}V_{ij}^2 \tag{24}$$

Each V represents the overlap. V_{pi} denotes the overlap of VM_i with PM_p:

$$V_{pi} = \max\left\{|x_i| + \frac{w_i}{2} - \frac{W}{2}, 0\right\} \tag{25}$$

V_{ij} is the overlap between VM_i and VM_j:

$$V_{ij} = \max\left\{\frac{w_i}{2} + \frac{w_j}{2} - \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, 0\right\} \tag{26}$$

Definition 1 To find the optimal solution, assume that the solution space $\mathbb{S}$ is a rectangular container. If the rectangular height of the solution $\mathbb{S}$ is to be set to H, and then its configuration X is continuously optimized, $\mathbb{S}$ becomes feasible, but by setting the rectangular height to any $H - \varepsilon$ ($\varepsilon > 0$) and then continuously optimizes its configuration X, $\mathbb{S}$ is infeasible, then H is the compact height of $\mathbb{S}$, denoted by H_{com} of $\mathbb{S}$.

Generally speaking, the optimal feasible solution can be obtained by only continuously optimizing H_{com}. Depending on the target of the problem (to gain the feasible solution with the smallest possible height of rectangular), the quality of the solution can be judged using H_{com} of the solution. When comparing two feasible solutions, the smaller H_{com}, the better the optimization is obviously. Unfortunately, it is time consuming to determine the H_{com} exactly. So we make certain compromises and approximately denote it with the closest lower bound of H_{com} (represented as H^{lb}_{com}).

Definition 2 Suppose a solution $\mathbb{S}$ has compact height H_{com}, and $l * n < H_{com} \leq (l+1) * n$, $l \in M - 1$ and $l > 0$, then $l * n$ is the closest lower bound height (represented as H^{lb}_{com}) of $\mathbb{S}$ with regard to interval n. Here, l denotes the number of active PMs and the interval n ($n > 0$) denotes the height of an active PM. As shown in Fig. 4, the grey rectangle represents the solution space , where the height of each PM is denoted by n. It is assumed that all PMs have the same height.

Theorem 1 *For any two feasible solutions* $\mathbb{S}(i)$ *and* $\mathbb{S}(j)$, *if* $H^{lb}_{\mathrm{com}}(j) > H^{lb}_{\mathrm{com}}(i)$, *then* $\frac{\partial H_{com}(j)}{\partial n} > \frac{\partial H_{com}(i)}{\partial n}$.

Proof According to Definition 2, let $H^{lb}_{com}(i) = l_i *$ $H^{lb}_{com}(j) = l_j * n, l_i, l_j \in M - 1$.

The first-order derivative of H_{com}, and constraints are given as follows:

$$l_i < \frac{\partial H_{\mathrm{com}}(i)}{\partial n} \leq l_i + 1$$

$$l_j < \frac{\partial H_{com}(j)}{\partial n} \leq l_j + 1$$

subject to the constraints:

$$\because H^{lb}_{com}(j) > H^{lb}_{com}(i)$$
$$\therefore l_i > l_j, \therefore l_j \geq l_i + 1$$

Hence, the following result can be achieved:

$$\frac{\partial H_{com}(j)}{\partial n} > l_j \geq l_i + 1 \geq \frac{\partial H_{com}(i)}{\partial n}$$

The closest lower bound height (H^{lb}_{com}) is an approximation of the compact height. When comparing two solutions $\mathbb{S}_i$ and $\mathbb{S}_j$, if $\mathbb{S}_i$ has a smaller H^{lb}_{com}(i), then its compact height will also be smaller (Theorem 1), which implies that $\mathbb{S}_i$ is better. But if they have the same H^{lb}_{com} , we will further compare their $\mathbb{E}$(X) (infeasibility) to determine which one is better. For example, in Fig. 4, the H^{lb}_{com} of solution $\mathbb{S}_i$ is ln, the H^{lb}_{com} of solution $\mathbb{S}_j$ is (l+1)n. So the compact height of $\mathbb{S}_i$ is between ln and (l+1)n, and the compact height of $\mathbb{S}_j$ is between (l+1)n and (l+2)n. So the compact height of $\mathbb{S}_i$ is certainly smaller, so $\mathbb{S}_i$ is obviously better. After the

completion of Algorithm 2, we have transformed the final solution into a feasible and compact solution. □

To boost the search performance of the MO_STVNS algorithm, we further integrate it into the iterative local search framework. Iterative local search is a prominent heuristic algorithm that is commonly used to raise the search performance of local search algorithms. When the local search process terminates, it partially modifies the current solution and resumes the local search. Algorithm 2 gives the pseudo-code of the iterative local search MO_STVNS for task requests scheduling in cluster management. line4-line12 represents the continuous optimization process involving all the neighborhoods. In each iteration, MO_STVNS is called to improve the current solution. If MO_STVNS outperforms the existing solution, an update to the existing solution is accepted. The continuous optimization process is terminated when the maximum iteration time (IterationLimit) is reached. The process finds the lower bound of the number of active PMs

Algorithm 2 Iterative local search MO_STVNS Algorithm

Require: $X, H = l * n(l \in N)$
Ensure: the best solution
1: $H \leftarrow$ Initialize the number of active PMs
2: $X \leftarrow$ Random placement of n VMs within active PMs
3: $Iterationtimes \leftarrow 0$
4: **while** Iterationtimes < IterationLimit **do**
5: $(H^{''}, X^{''}) \leftarrow MO\_STVNS(H, X)$
6: **if** $(H^{''}, X^{''})$ is better than (H, X) **then**
7: $(H, X) \leftarrow (H^{''}, X^{''})$
8: $Iterationtimes \leftarrow 0$
9: **else**
10: $Iterationtimes \leftarrow Iterationtimes + 1$
11: **end if**
12: **end while**
13: **return** (H, X)

H_{com}^{lb} (In Algorithm 2, H_{com}^{lb} is abbreviated to H.), i.e., it finds the optimal solution.

5 Evaluation

This section describes our experimentation. Since MO-FreeVM is essentially an algorithm for allocation, our testbed is built in python for efficiency.

5.1 Experimental set-up

The testbed is associated with a mysql database in order to visualize the placement and migration of the VMs and to facilitate access to the data. Furthermore, MO-FreeVM has a series of external interfaces that can be interfaced with any data center cluster system.

The algorithm is compared with GWO [43], SA [45] and (GA) [44] with two different traces from publicly available real workloads, Google Cluster Data (GCD) and Alibaba Traces (AT). Simulations are conducted on a PC with a Core i7, 3.4GHz CPU, 16GB RAM, Windows 10 and Python3.

Machines of Google cluster share a common cluster management system, which distributes workloads to the machines [58]. The heterogeneous workload is comprised of two types of task requests. The first type is long-running services used to handle short-lived latency-sensitive requests, such as web search and Gmail. The second type is batch jobs such as MapReduce [59] and machine learning submitted by internal users, which have a lifetime ranging from a few seconds to a few days. Each batch job consists of one or more tasks that perform different computational logic, and directed acyclic graph (DAG) can present their existing dependencies [60].

In 2017, Alibaba [61], the largest cloud service provider in China, released a publicly accessible dataset. The dataset consists of 1.3K machines running long-running services and batch jobs over a 12-hour period. Unlike the heterogeneity of Google cluster, the specification of all servers is the same, with 96 cores and 1 unit of memory standardized. The usage file records the usage of runtime resources, including CPU, memory, Network and IO.

The trace providers indicate that the priority of task requests is related to the type of workload, long-running services have a higher priority. There are two reasons for the frequent scheduling of task requests. One is that there are many long-running services being scheduled. The other is that tasks are terminated and batch jobs need to be rescheduled. Fortunately, despite the large number of jobs starting and stopping, these long-running services do not have a significant impact on usage. Therefore, the scheduler can safely ignore long-running services when predicting cluster utilization.

5.2 MO_STVNS over existing solutions

The parameter settings of the comparison algorithm are described.

5.2.1 GWO algorithm parameter setting

GWO is inspired by the gray wolf population, in which each possible solution is assumed to be a wolf, and the highest scoring solution are the dominant wolves. The hunting of the gray wolves is guided by alpha (α), beta (β) and delta (δ) wolves. Consequently, the best solution are generated by the alpha (α) wolves, followed by beta (β) and delta (δ) wolves. The rest of the candidate solutions are assumed to be omega (ω) wolves. To facilitate the calculation, the initialization parameters for GWO are given below [62, 63]: it is assumed that the positions of the wolves of α and β are set to α is 0.5 and $\beta \in (0, 2)$ respectively, with β being mainly responsible for assisting α in the decision making. Meanwhile, in order to reduce the number of iterations in the calculation, we assume that the number of wolves seeking value is 5 and the number of grids per dimension is 10.

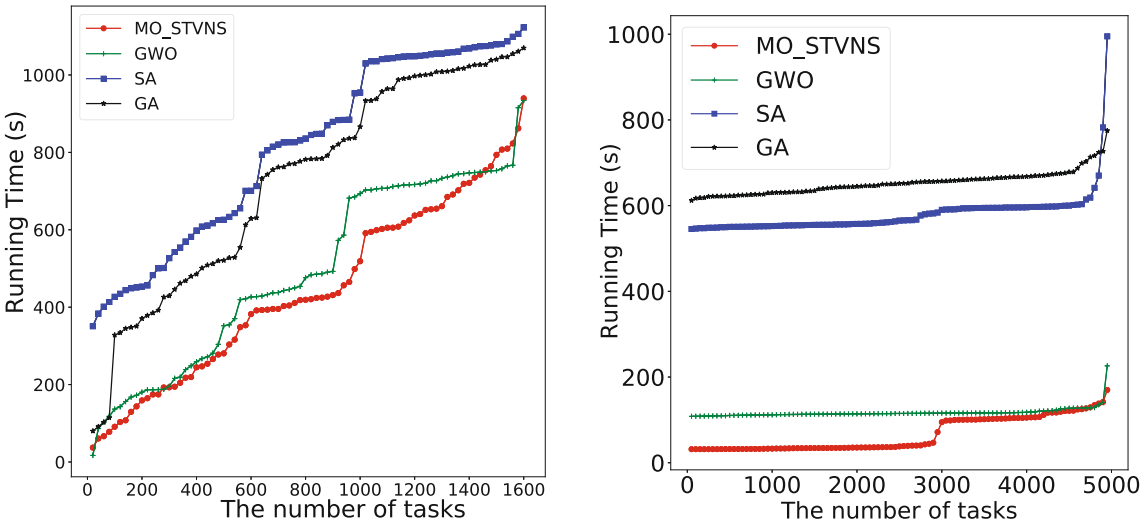
5.2.2 SA algorithm parameter setting

The simulated annealing is inspired by the annealing process in metalwork. We initially set the temperature very high and then let it cool slowly as the algorithm runs. The core of the SA algorithm is to accept the current non-optimal solution with a certain probability, thus jumping out of the local optimal solution and continuing the search, which leads to the global optimal solution. The initial parameters for SA are given below [46], where the desirability of a source PM j as a sigmoid-like shaped function $D_{src}(u_j) = e^{-a\cdot(u_j)^3}$ (a=6) of its utilization u_j and the desirability of destination PM h is defined as a gaussian function $D_{dst}(u_h) = b \cdot e^{-c\cdot\left(u_h-\frac{1}{2}\right)^2}$ ($b = 0.85,\ c = 20$) of its utilization u_h.

5.2.3 GA algorithm parameter setting

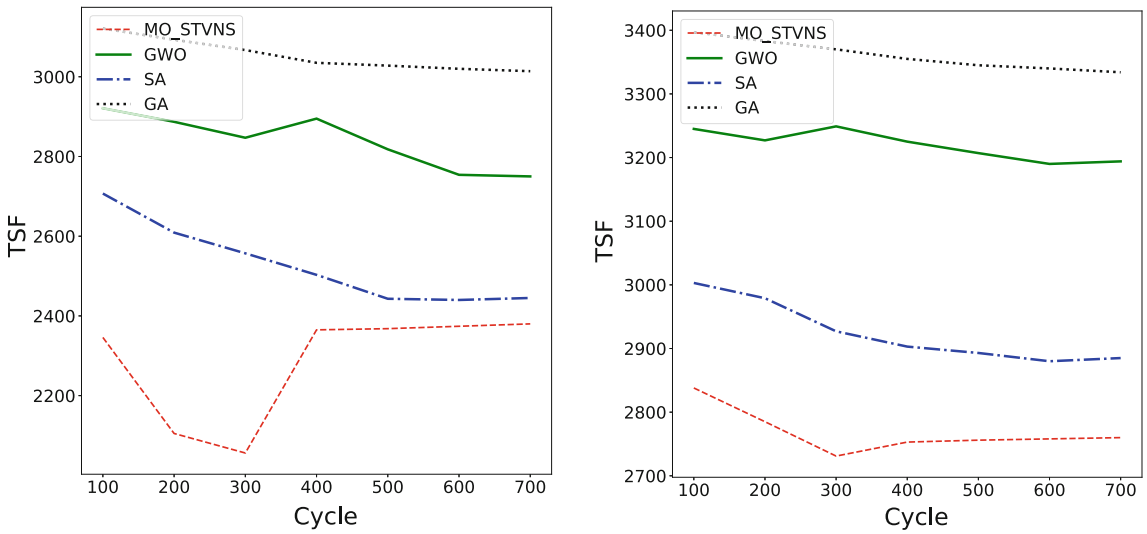
Genetic algorithms are inspired by the process of natural selection. The GA improves the structures in this population by performing selection, followed by crossover and mutation. After several generations, sufficiently good solutions will be formed in the population. Consider our scenario and the existing parameter settings for the works associated with VM scheduling using the GA method [26, 64]. The population size is set to 200, with crossover and mutation probabilities of 0.95 and 0.05, respectively.

Our proposed MO_STVNS algorithm involves the relevant parameters of cost and TSF. The parameters of cost



(a) Google Cluster Data (b) Alibaba Traces

Fig. 5 Comparing the runtime overhead of different algorithms



(a) Google Cluster Data (b) Alibaba Traces

Fig. 6 Convergence of different algorithms

include w_P, w_M and w_B, we set w_P=2 and w_M=w_B=1, due to the fact that Pcost has a larger share compared to Mcost and Bcost. The parameters of TSF include α and β, and we set $\alpha + \beta = 1$ and α =0.25.

5.3 Experimental results and analysis

Figure 5a shows the runtime overhead when executing different number of tasks under the GCD; Fig. 5b shows the runtime overhead when executing different number of tasks under the AT. The unit of runtime overhead in Fig. 5 are seconds. By comparing the algorithm data with the runtime overhead of the best solution obtained by each algorithm, it can be concluded that the runtime overhead of each algorithm will increase with the increase of task data. The MO_ STVNS and the GWO have a very low runtime overhead due to their relatively simple allocation operations, while the GA performs a series of more complex operations and therefore has a higher overhead. As can be seen from Fig. 5a, when the number of tasks volume is small, the runtime overhead of MO_STVNS algorithm is similar to that of GWO algorithm, but as the tasks volume increases, the average cost increase speed of GWO is 14.65% faster than that of MO_STVNS. Figure 5b shows that the runtime overhead variation of each algorithm is relatively flat as the tasks volume increases, and the runtime overhead of the MO_STVNS algorithm gradually becomes larger when the tasks volume is larger, but it also has a smaller runtime overhead than the other algorithms. This is due to the large number of heterogeneous tasks in the GCD with great difference in the task size, as well as batch jobs account for a large proportion. However, the AT has homogeneous tasks, more long-running services, and almost the same size of tasks. Therefore, in terms of algorithm running time overhead, the MO_STVNS algorithm outperforms the other algorithms, and the GA algorithm has the largest runtime overhead. From the above analysis, we can see that our proposed MO_STVNS algorithm is advantageous in terms of runtime overhead.

Figure 6a and b show the results of TSF in the Google Cluster Data and Alibaba Traces for MO_STVNS, GWO, SA and GA algorithms. Among them, each iteration cycle of GCD and AT reach 20 and 50 task requests respectively. The results illustrate that the convergence speed of MO_STVNS algorithm is faster than other algorithms. By comparing Fig. 6a and b , it is observed that the TSF of SA and GA gradually decreases with the increase of the number of iteration cycles. For GCD and AT, the average TSF decline rate of GA is 60.67% to 50.77% slower than that of SA. Compared with the SA, the GA suffers from poor local search capabilities and a tendency to fall into premature convergence. The convergence speed of GWO is slower than that of other algorithms. When the iteration cycle increases to about 600, the TSF remains small change

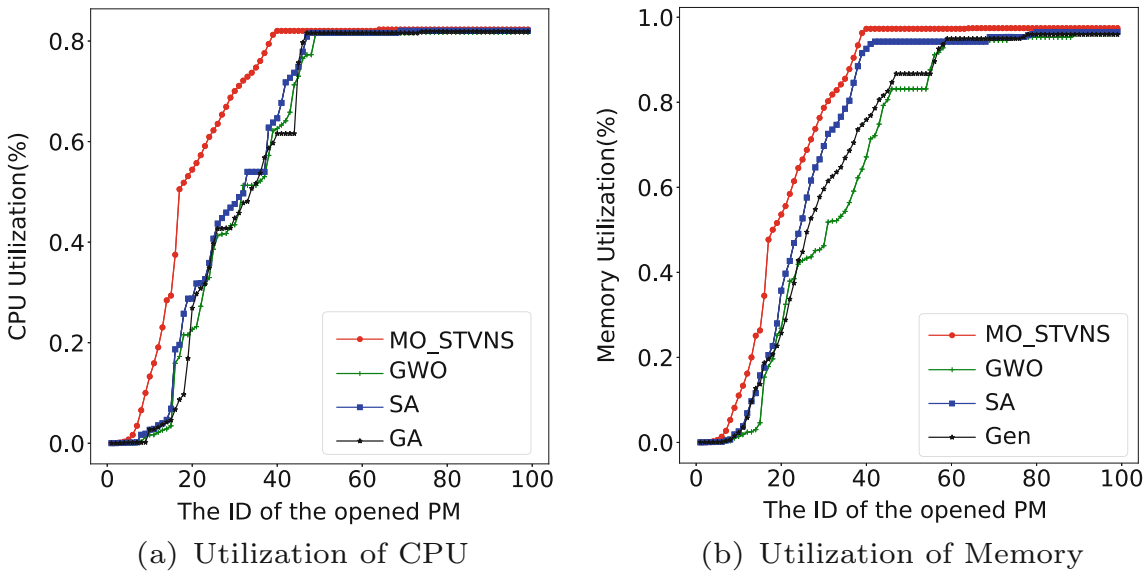
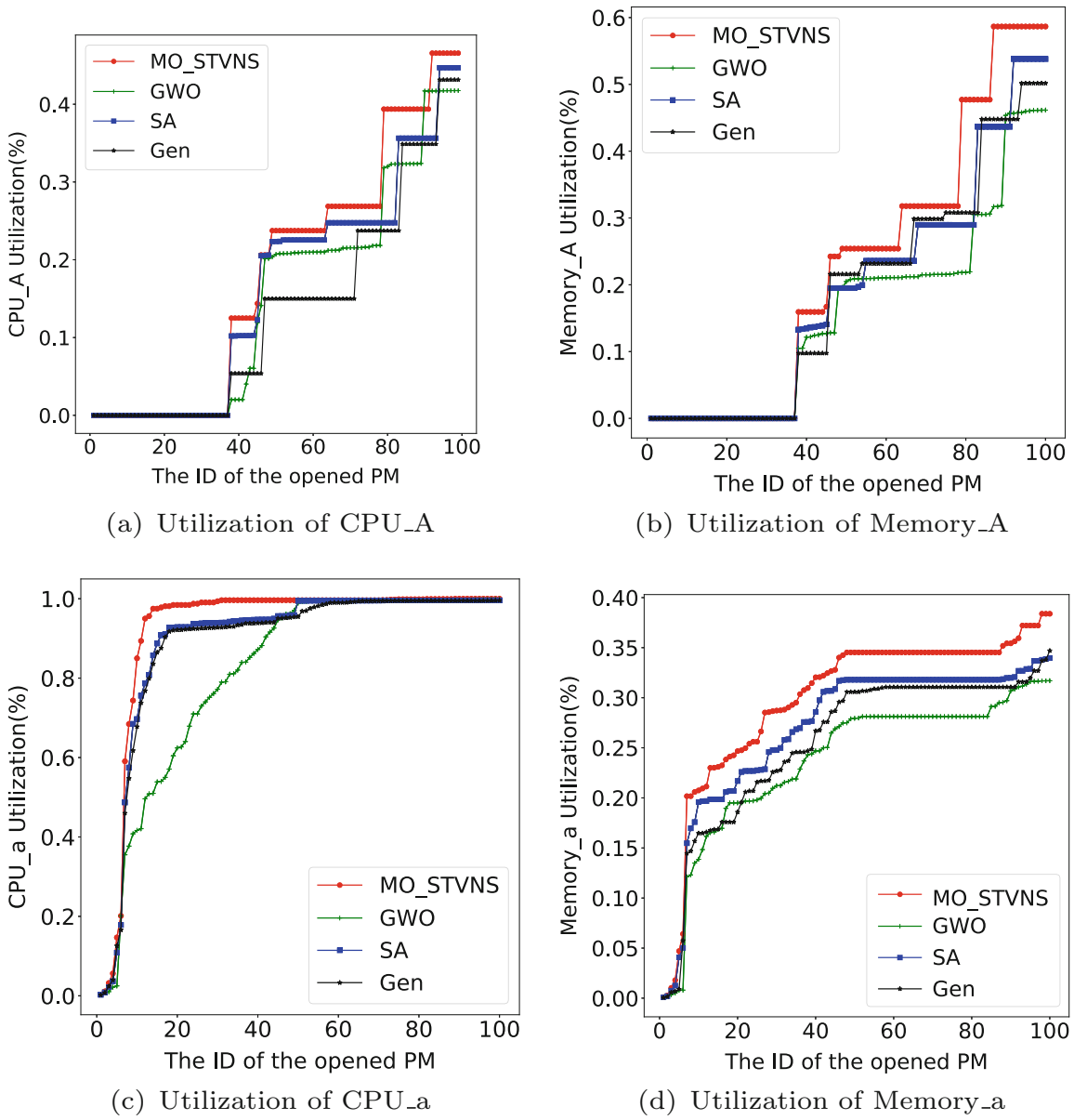


Fig. 7 Comparison of resource utilization of different algorithms under Alibaba traces

and GWO is close to convergence. When the iteration cycle of MO_STVNS algorithm is greater than 400, the algorithm tends to converge because MO_STVNS reduces the search space and further reduces the search time. The MO_STVNS algorithm tends to converge when the iteration period of the algorithm is greater than 400. This is due to the fact that VMs will be grouped based on historical experience before using MO_ STVNS to find the most suitable PM. Infeasible solutions are updated and continuously corrected in each search iteration, so that MO_STVNS has the least convergence cycles. Between 300 and 400 cycles, the previously arriving VM requests are not fully released and a large number of PMs need to be enabled to handle the newly arriving VM requests, resulting in an increasing TSF. Our proposed MO_STVNS algorithm has a better vacating effect than other algorithms, so that the minimum number of active PMs is reached at 300 cycles. For the heterogeneous Google Cluster Data, between 300 and 400 cycles, the MO_STVNS algorithm no longer performs vacating PMs and needs to enable more heterogeneous PMs, so there is a jump in the TSF of our proposed MO_STVNS algorithm from 300 to 400 cycles.

Figures 7 and 8 represent the resource utilization of CPU and memory for different algorithms with the same number of active PMs for the Google dataset and the Ali dataset, respectively. Figures 8(a)–(d) show the utilization of different attribute resources in a heterogeneous environment, taking CPU a resource as an example, Figure 8 shows the utilizations of 100 active PMs. Taking CPU a resources as an example, there are about 36 active PMs whose CPU utilizations exceed 23% under the SA 28 active PMs under the GA and only 21 active PMs under the GWO. But when leveraging MO_STVNS, the highest CPU utilization ascends to 47%, which is much better than other algorithms. Figure 7 shows the utilization of different resources in a homogeneous environment, and we observe that the resource usages of PMs reach convergence. Taking CPU as an example, when analyzing the convergence speed of the CPU, MO_STVNS is 20.11% faster than SA and GA, and 25.06% faster than GWO. Furthermore, the average utilizations of the top 60 "hot" PMs under MO_STVNS, GWO, SA and GA are 82.11%, 80.65%, 81.33% and 80.22% respectively. To enhance resource utilization of PMs, as many VMs as possible must be hosted on each PM, thus minimizing the number of active PMs in the cloud data center.

The boxplot is consisted by five numerical points: minimum (min), lower quartile (Q1), median (median), upper quartile (Q3), and maximum (max). The mean can also be added to the boxplot. In this case, the lower quartile, median, and upper quartile form a "box with compartments". In the boxplot, there is a line in the middle of the box that represents the median of the data. The top and bottom of the box are the upper quartile and the lower quartile of the data respectively, which means that the box

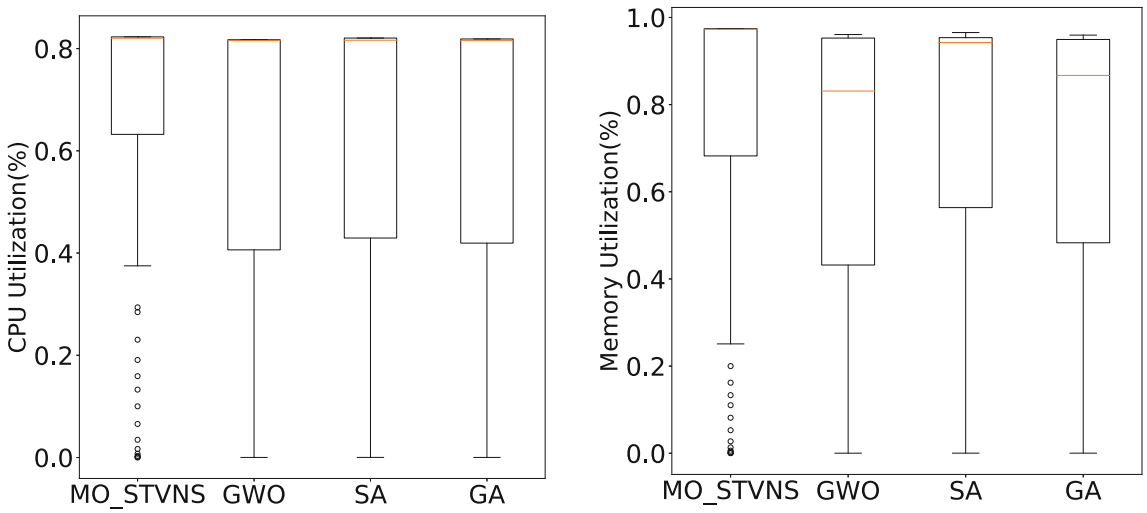


(a) Utilization of CPU_A (b) Utilization of Memory_A

(c) Utilization of CPU_a (d) Utilization of Memory_a

Fig. 8 Comparison of resource utilization of different algorithms under Google Cluster data

contains 50% of the data. The height of the box therefore reflects, to some extent, the degree of volatility of the data. In addition, an extension line is created between the upper quartile and the maximum value, which becomes a "whisker". If there is no number larger than the maximum observation, the upper whisker limit is the maximum value. Figures 9 and 10 reflect the balance of different resource utilization under different data sets. Taking Fig. 10a for example, when analyzing the balance of CPU A resource utilization, it is observed that MO_STVNS outperforms GWO by 51.97%, SA by 21.56% and GA by 58.27%. To maximize resource utilization and balance the use of resources (CPU and memory), MO_STVNS can reallocate VMs to PMs, after adding VM requirements to the used PM resources, then it can rank the VMs hosted by overloaded PMs and select the most imbalanced resource utilization based on the absolutely difference between the CPU and memory requirements of these VMs. Figures 7



(a) The balance of CPU (b) The balance of Memory

Fig. 9 Comparison of the balance of different algorithms under Alibaba Traces

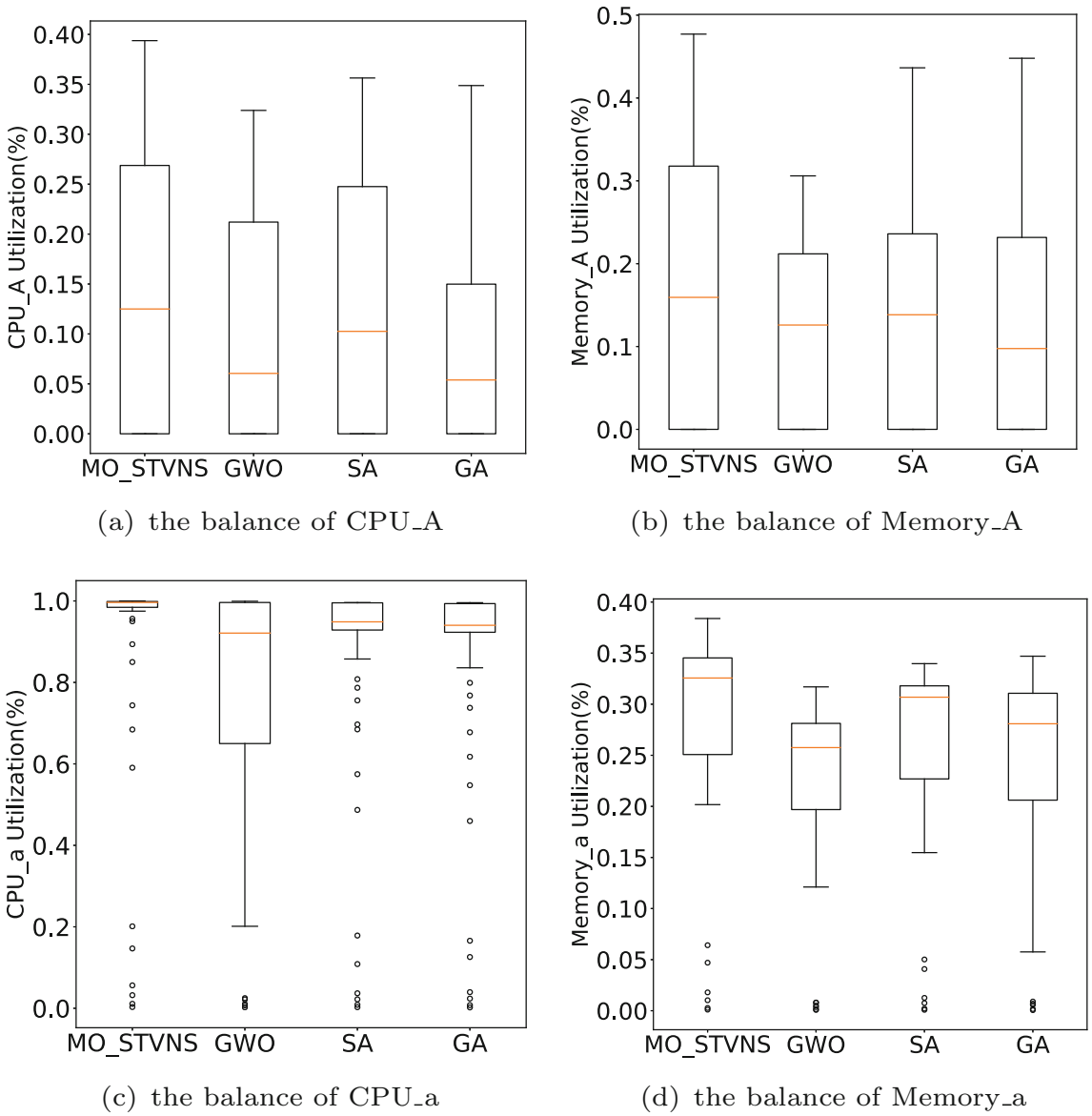
and 8 show all active PMs sorted from lowest to highest resource utilization, which reflects the trend of PMs resource utilization. The boxplots shown in Figs. 9 and 10 show a set of statistical plots of data dispersion, mainly used to reflect the characteristics of the original data distribution, and also to allow comparison of the characteristics of multiple data distributions.

Figures 11a and b compare the average number of active PMs for different algorithms with different data sets. When calculating the average number of active PMs, the average value of five consecutive measurements under the same iteration period is considered. We can infer that the average number of active PMs increases with the iteration period for both heterogeneous and homogeneous environments, and the MO_STVNS algorithm has a significant effect in terms of energy saving and is superior in the homogeneous environment. As shown in Fig. 11a, with the increment of iteration period MO_STVNS saves on average 25.94% PMs compared with GWO, 10.29% PMs compared with SA, and 20.30% PMs compared with GA. In Fig. 11b , with the increment of iteration period MO_STVNS saves on average 35% PMs compared with GWO, 27.02% PMs compared to SA, and 32.47% PMs compared with GA. This difference is mainly due to the longer lifecycle of task requests in the AT and the fact that the number of task requests released per iteration cycle is higher than that of the GCD.

It can be seen from Tables 2 and 3, all the cost of MO_STVNS outperform other comparative algorithms in both GCD and AT scenarios for different iteration cycles generating different numbers of task requests. Taking Table 2 for example, in terms of average total cost, MO_STVNS is reduced by 26.92% compared to GWO, 24.02% compared to SA and 33.75% compared to GA. MO_STVNS, GWO, SA and GA employ different rescheduling strategies to clean underutilized PMs and improve utilization, whereas MCost depict their task rescheduling cost. Compared to the others, MO_STVNS enjoys the lowest PM migration cost, reduces migration frequency, and improves resource utilization more effectively. Moreover, having a larger BCost under GCD is due to the larger Balance Ratio of CPU and Memory resources under Google dataset. From the impact of PCost, it is shown that the TSF of MO_STVNS algorithm is lower than GWO, SA and GA, respectively, which means that MO_STVNS brings performance improvement in minimizing the TSF.

6 Conclusion

More and more cloud service providers are deploying their services to data centers. Nevertheless, cluster management in large Internet data centers needs to both guarantee



(a) the balance of CPU_A

(b) the balance of Memory_A

(c) the balance of CPU_a

(d) the balance of Memory_a

Fig. 10 Comparison of the balance of different algorithms under Google cluster data

cluster performance and improve the execution time of task requests. In this paper, we analyze cluster management issues for colocated task requests and propose two new strict objectives: minimizes both the total cost and TSF. A MO_STVNS algorithm based on resource prediction is proposed to solve the problem of resilient resource management, and a MO-FreeVM scheduler based on this algorithm is implemented. The performance evaluation shows that MO_STVNS maximizes resource utilization and minimizes the number of active PMs, running overhead, TSF, and total cost. All these results are experimentally validated with two different datasets and compared with state-of-the-art algorithms. When the number of tasks volume is small, the RO of MO_STVNS algorithm is similar to the result of GWO algorithm, but as the task volume increases, the average cost increase speed of GWO is 14.65% faster than that of MO_STVNS. For GCD and AT, the average TSF decline rate of GA is

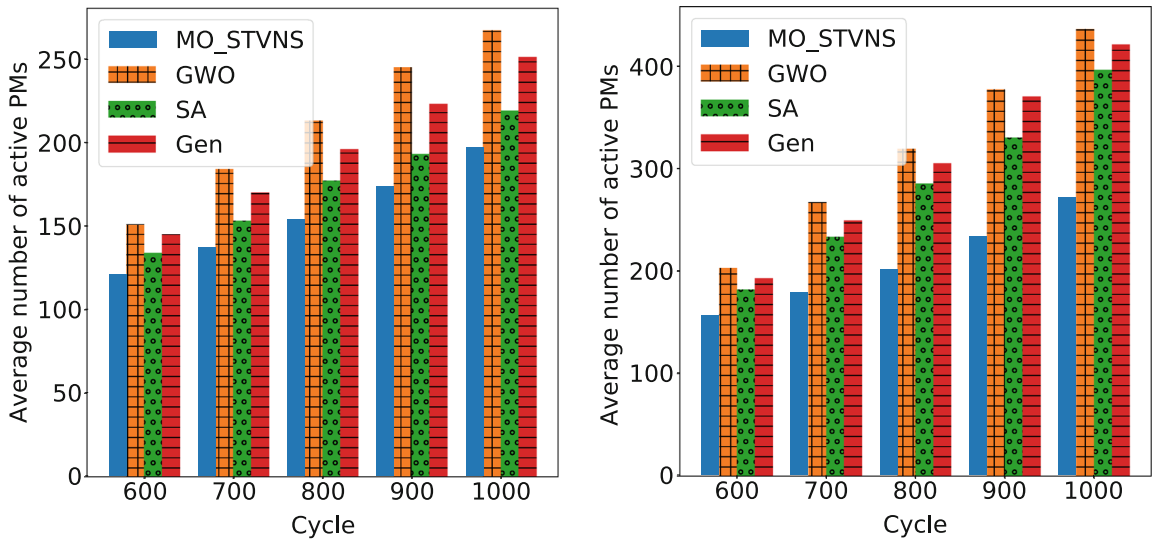


Fig. 11 The average number of active PMs in different algorithms for both homogeneous and heterogeneous environments

Table 2 Comparison of all costs of different algorithms under Google cluster data

Cost	Cycle	MO_STVNS	GWO	SA	GA
TCost	100	4.02	4.38	4.55	4.8
	200	7.35	10.91	9.52	11.07
	300	10.45	14.35	14.24	15.95
	400	13.4	20.18	19.07	22.43
	500	15.26	22.76	21.61	27.32
MCost	100	2.15	2.16	2.34	2.41
	200	3.45	4.95	4.12	4.87
	300	5.42	6.40	7.11	7.67
	400	6.69	9.93	9.54	12.24
	500	7.37	11.01	10.87	14.47
BCost	100	0.87	0.93	0.89	0.99
	200	1.98	2.11	2.02	2.63
	300	2.53	3.21	2.92	3.82
	400	3.75	4.98	4.55	5.16
	500	4.39	5.72	5.03	6.87
PCost	100	1.00	1.29	1.32	1.40
	200	1.92	3.85	3.38	3.57
	300	2.50	4.74	4.21	4.46
	400	2.96	5.27	4.98	5.03
	500	3.50	6.03	5.71	5.98

Table 3 Comparison of all costs of different algorithms under alibaba traces

Cost	Cycle	MO_STVNS	GWO	SA	GA
TCost	100	4.63	6.03	5.53	5.8
	200	7.65	10.99	9.27	11.04
	300	12.26	16.03	14.57	16.56
	400	17.01	21.64	19.06	22.84
	500	23.69	29.12	26.53	32.03
MCost	100	2.90	3.18	2.99	3.12
	200	3.77	4.80	4.07	4.93
	300	6.52	7.81	7.54	8.05
	400	8.96	9.37	9.06	10.23
	500	13.15	15.06	13.76	15.70
BCost	100	0.3	0.42	0.39	0.45
	200	0.86	1.31	1.17	1.53
	300	1.14	2.32	2.03	2.45
	400	2.11	3.53	3.27	3.64
	500	3.05	4.97	4.45	5.40
PCost	100	1.43	2.43	2.15	2.23
	200	3.02	4.88	4.03	4.58
	300	4.60	5.90	5.00	6.06
	400	5.94	8.74	6.73	8.97
	500	7.49	9.09	8.32	10.93

60.67% to 50.77% slower than that of SA. The average utilizations of the top 60 "hot" PMs under MO_STVNS, GWO, SA, and GA are 82.11%, 80.65%, 81.33%, and 80.22% respectively. When analyzing the balance of CPU_A resource utilization, it is observed that MO_STVNS outperforms GWO by 51.97%, SA by 21.56%, and GA by 58.27%. With the increment of iteration period, MO_STVNS saves on average 25.94% PMs compared with GWO, 10.29% PMs compared with SA, and 20.30% PMs compared with GA. In terms of the average

total cost, MO_STVNS is 26.92% lower compared to GWO, MO_STVNS is 24.02% lower compared to SA, and MO_STVNS is 33.75% lower compared to GA.

In future work, we need to consider the impact of correlation between different tasks on dynamic cluster management. In addition, the practical cloud environment, there are more constraints involved as well as more objectives. Therefore, we attempt to make these improvements as part of our future work.

Author contributions Methodology, Shiyan Zhang; Writing-original draft preparation, Shiyan Zhang; Writing review and editing, Shiyan Zhang, Ran Wang; Funding acquisition, Yuchao Zhang. All authors have read and agreed to the published version of the manuscript.

Funding The work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62172054, the Key Project of Beijing Natural Science Foundation under M21030, the NSFC under Grant 62072047, and the National Key R&D Program of China under Grant 2019YFB1802603.

Data availability All data generated or analysed during this study are included in the article.

Code availability Open source software is used.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants performed by any of the authors.

References

1. Birke, R., Podzimek, A., Chen, L.Y., Smirni, E.: Virtualization in the private cloud: state of the practice. IEEE Trans. Netw. Serv. Manag. **13**(3), 608–621 (2016)
2. Stoyanova, M., Nikoloudakis, Y., Panagiotakis, S., Pallis, E., Markakis, E.K.: A survey on the internet of things (iot) forensics: challenges, approaches, and open issues. IEEE Commun. Surv. Tutor. **22**(2), 1191–1221 (2020)
3. Wan, J., Li, X., Dai, H.-N., Kusiak, A., Martínez-García, M., Li, D.: Artificial-intelligence-driven customized manufacturing factory: key technologies, applications, and challenges. Proc. IEEE. **109**(4), 377–398 (2020)
4. Saxena, D., Singh, A.K., Buyya, R.: Op-mlb: An online vm prediction based multi-objective load balancing framework for resource management at cloud datacenter. IEEE Trans. Cloud Comput. (2021). https://doi.org/10.1109/TCC.2021.3059096
5. Guerrero, C., Lera, I., Juiz, C.: Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. J. Grid Comput. **16**(1), 113–135 (2018)
6. Liu, B., Li, P., Lin, W., Shu, N., Li, Y., Chang, V.: A new container scheduling algorithm based on multi-objective optimization. Soft Comput. **22**(23), 7741–7752 (2018)
7. Kaewkasi, C., Chuenmuneewong, K.: Improvement of container scheduling for Docker using ant colony optimization. In: 2017 9th International Conference on Knowledge and Smart Technology (KST), pp. 254–259. IEEE (2017)
8. Taherizadeh, S., Stankovski, V.: Dynamic multi-level auto-scaling rules for containerized applications. Comput. J. **62**(2), 174–197 (2019)
9. Kehrer, S., Blochinger, W.: Tosca-based container orchestration on mesos. Comput. Sci. Res. Dev. **33**(3), 305–316 (2018)
10. Yin, L., Luo, J., Luo, H.: Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. IEEE Trans. Industr. Inf. **14**(10), 4712–4721 (2018)
11. Xu, X., Yu, H., Pei, X.: A novel resource scheduling approach in container based clouds. In: 2014 IEEE 17th International Conference on Computational Science and Engineering, pp. 257–264. IEEE (2014)
12. Han, P., Du, C., Chen, J., Ling, F., Du, X.: Cost and makespan scheduling of workflows in clouds using list multiobjective optimization technique. J. Syst. Arch. **112**, 101837 (2021)
13. Zhou, X., Zhang, G., Sun, J., Zhou, J., Wei, T., Hu, S.: Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft. Future Gener. Comput. Syst. **93**, 278–289 (2019)
14. Kaur, N., Aulakh, T.S., Cheema, R.S.: Comparison of workflow scheduling algorithms in cloud computing. Int. J. Adv. Compute. Sci. Appl. **2**(10), 81 (2011)
15. Liu, K., Jin, H., Chen, J., Liu, X., Yuan, D., Yang, Y.: A compromised-time-cost scheduling algorithm in swindew-c for instance-intensive cost-constrained workflows on a cloud computing platform. Int. J. High Perform. Comput. Appl. **24**(4), 445–456 (2010)
16. Wu, Z., Liu, X., Ni, Z., Yuan, D., Yang, Y.: A market-oriented hierarchical scheduling strategy in cloud workflow systems. J. Supercomput. **63**(1), 256–293 (2013)
17. Abrishami, S., Naghibzadeh, M.: Deadline-constrained workflow scheduling in software as a service cloud. Sci. Iran. **19**(3), 680–689 (2012)
18. Barroso, L.A., Clidaras, J., Hölzle, U.: The datacenter as a computer: an introduction to the design of warehouse-scale machines. Synt. Lect. Comput. Architect. **8**(3), 1–154 (2013)
19. Alshahrani, R., Peyravi, H.: Modeling and simulation of data center networks. In: Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, pp. 75–82 (2014)
20. Alkhanak, E.N., Lee, S.P., Khan, S.U.R.: Cost-aware challenges for workflow scheduling approaches in cloud computing environments: taxonomy and opportunities. Future Gener. Comput. Syst. **50**, 3–21 (2015)
21. Zhang, S., Zhang, Y., Gong, X., Wang, R.: Freevm: A server release algorithm in datacenter network. In: ICC 2021-IEEE International Conference on Communications, pp. 1–6 (2021). IEEE
22. Verma, A., Ahuja, P., Neogi, A.: pmapper: power and migration cost aware application placement in virtualized systems. In: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, pp. 243–264. Springer (2008)
23. Ferdaus, M.H., Murshed, M., Calheiros, R.N., Buyya, R.: Virtual machine consolidation in cloud data centers using aco metaheuristic. In: European Conference on Parallel Processing, pp. 306–317. Springer (2014)

24. Le, T.N., Sun, X., Chowdhury, M., Liu, Z.: Allox: compute allocation in hybrid clusters. In: Proceedings of the Fifteenth European Conference on Computer Systems, pp. 1–16 (2020)
25. Chaudhary, S., Ramjee, R., Sivathanu, M., Kwatra, N., Viswanatha, S.: Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In: Proceedings of the Fifteenth European Conference on Computer Systems, pp. 1–16 (2020)
26. Joseph, C.T., Chandrasekaran, K., Cyriac, R.: Improving the efficiency of genetic algorithm approach to virtual machine allocation. In: 2014 International Conference on Computer and Communication Technology (ICCCT), pp. 111–116 (2014). IEEE
27. Wu, Y., Tang, M., Fraser, W.: A simulated annealing algorithm for energy efficient virtual machine placement. In: 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 1245–1250. IEEE (2012)
28. Zhang, X., Lin, Q., Mao, W., Liu, S., Dou, Z., Liu, G.: Hybrid particle swarm and grey wolf optimizer and its application to clustering optimization. Appl. Soft Comput. **101**, 107061 (2021)
29. Zhang, Y., Li, Y., Xu, K., Wang, D., Li, M., Cao, X., Liang, Q.: A communication-aware container re-distribution approach for high performance vnfs. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 1555–1564. IEEE (2017)
30. Lv, L., Zhang, Y., Li, Y., Xu, K., Wang, D., Wang, W., Li, M., Cao, X., Liang, Q.: Communication-aware container placement and reassignment in large-scale internet data centers. IEEE J. Select. Areas Commun. **37**(3), 540–555 (2019)
31. Canali, C., Chiaraviglio, L., Lancellotti, R., Shojafar, M.: Joint minimization of the energy costs from computing, data transmission, and migrations in cloud data centers. IEEE Trans. Green Commun. Netw. **2**(2), 580–595 (2018)
32. Ran, W., Yuchao, Z., Wendong, W., Ke, X., Laizhong, C.: Algorithm of mixed traffic scheduling among data centers based on prediction. J. Comput. Res. Dev. **58**(6), 1307 (2021)
33. Pickartz, S., Eiling, N., Lankes, S., Razik, L., Monti, A.: Migrating linux containers using criu. In: International Conference on High Performance Computing, pp. 674–684. Springer (2016)
34. Rizvi, N., Dharavath, R., Edla, D.R.: Cost and makespan aware workflow scheduling in iaas clouds using hybrid spider monkey optimization. Simul. Model. Pract. Theory **110**, 102328 (2021)
35. Sahni, J., Vidyarthi, D.P.: A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Trans. Cloud Comput. **6**(1), 2–18 (2015)
36. Wu, K.: A tunable workflow scheduling algorithm based on particle swarm optimization for cloud computing (2014)
37. Su, S., Li, J., Huang, Q., Huang, X., Shuang, K., Wang, J.: Cost-efficient task scheduling for executing large programs in the cloud. Parall. Comput. **39**(4–5), 177–188 (2013)
38. Quan, Z., Wang, Z.-J., Ye, T., Guo, S.: Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems. IEEE Trans. Parall. Distrib. Syst. **31**(5), 1165–1182 (2019)
39. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., Wilkes, J.: Large-scale cluster management at google with borg. In: Proceedings of the Tenth European Conference on Computer Systems, pp. 1–17 (2015)
40. Burns, B., Beda, J., Hightower, K.: Kubernetes: up and Running: Dive Into the Future of Infrastructure. O'Reilly Media, ??? (2019)
41. Garefalakis, P., Karanasos, K., Pietzuch, P., Suresh, A., Rao, S.: Medea: scheduling of long running applications in shared production clusters. In: Proceedings of the Thirteenth EuroSys Conference, pp. 1–13 (2018)
42. Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S.: Apache hadoop yarn: yet another resource negotiator. In: Proceedings of the 4th Annual Symposium on Cloud Computing, pp. 1–16 (2013)
43. Al-Moalmi, A., Luo, J., Salah, A., Li, K.: Optimal virtual machine placement based on grey wolf optimization. Electronics **8**(3), 283 (2019)
44. Zhang, L., Li, K., Li, C., Li, K.: Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. Inf. Sci. **379**, 241–256 (2017)
45. Khalilzad, N., Faragardi, H.R., Nolte, T.: Towards energy-aware placement of real-time virtual machines in a cloud data center. In: 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, pp. 1657–1662 (2015). IEEE
46. Marotta, A., Avallone, S.: A simulated annealing based approach for power efficient virtual machines consolidation. In: 2015 IEEE 8th International Conference on Cloud Computing, pp. 445–452 (2015). IEEE
47. Zhong, Z., Buyya, R.: A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources. ACM Trans. Internet Technol. (TOIT) **20**(2), 1–24 (2020)
48. Curino, C., Krishnan, S., Karanasos, K., Rao, S., Fumarola, G.M., Huang, B., Chaliparambil, K., Suresh, A., Chen, Y., Heddaya, S.: Hydra: a federated resource manager for data-center scale analytics. In: 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), pp. 177–192 (2019)
49. Liu, X., Cheng, B., Wang, S.: Availability-aware and energy-efficient virtual cluster allocation based on multi-objective optimization in cloud datacenters. IEEE Trans. Netw. Serv. Manag. **17**(2), 972–985 (2020)
50. Li, C., Wang, Y., Tang, H., Luo, Y.: Dynamic multi-objective optimized replica placement and migration strategies for saas applications in edge cloud. Future Gener. Comput. Syst. **100**, 921–937 (2019)
51. Ji, J.-Y., Wong, M.L.: An improved dynamic multi-objective optimization approach for nonlinear equation systems. Inf. Sci. **576**, 204–227 (2021)
52. Patel, Y.S., Malwi, Z., Nighojkar, A., Misra, R.: Truthful online double auction based dynamic resource provisioning for multi-objective trade-offs in iaas clouds. Clust. Comput. **24**(3), 1855–1879 (2021)
53. Devi, K.L., Valli, S.: Multi-objective heuristics algorithm for dynamic resource scheduling in the cloud computing environment. J. Supercomput. **77**(8), 8252–8280 (2021)
54. Liu, Q., Yu, Z.: The elasticity and plasticity in semi-containerized co-locating cloud workload: a view from alibaba trace. In: Proceedings of the ACM Symposium on Cloud Computing, pp. 347–360 (2018)
55. Hansen, P., Mladenović, N., Moreno Perez, J.A.: Variable neighbourhood search: methods and applications. 4OR 6(4), 319–360 (2008)
56. Lusa, A., Potts, C.N.: A variable neighbourhood search algorithm for the constrained task allocation problem. J. Oper. Res. Soc. **59**(6), 812–822 (2008)
57. Kardani-Moghaddam, S., Khodadadi, F., Entezari-Maleki, R., Movaghar, A.: A hybrid genetic algorithm and variable neighborhood search for task scheduling problem in grid environment. Proc. Eng. **29**, 3808–3814 (2012)
58. Google trace. https://github.com/google/cluster-data (2011)
59. Tripathi, A.K., Sharma, K., Bala, M.: A novel clustering method using enhanced grey wolf optimizer and mapreduce. Big Data Res. **14**, 93–100 (2018)

60. Tariq, R., Aadil, F., Malik, M.F., Ejaz, S., Khan, M.U., Khan, M.F.: Directed acyclic graph based task scheduling algorithm for heterogeneous systems. In: Proceedings of SAI Intelligent Systems Conference, pp. 936–947. Springer (2018)
61. Google trace. https://github.com/alibaba/clusterdata (2017)
62. Fatima, A., Javaid, N., Anjum Butt, A., Sultana, T., Hussain, W., Bilal, M., Hashmi, M.A.U.R., Akbar, M., Ilahi, M.: An enhanced multi-objective gray wolf optimization for virtual machine placement in cloud data centers. Electronics 8(2), 218 (2019)
63. Singh, P., Rizvi, M.A.: Virtual machine selection strategy based on grey wolf optimizer in cloud environment: a study. In: 2018 8th International Conference on Communication Systems and Network Technologies (CSNT), pp. 108–112. IEEE (2018)
64. Kaaouache, M.A., Bouamama, S.: An energy-efficient vm placement method for cloud data centers using a hybrid genetic algorithm. J. Syst. Inf. Technol. 20, 430–445 (2018)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Shiyan Zhang received her M.S. degree from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2015. She is currently pursuing a Ph.D. degree at the Beijing University of Posts and Telecommunications (BUPT), Beijing, China. Her research interests mainly include time sensitive networks, data center networks, and edge computing.

Yuchao Zhang received her Ph.D. degree from the Computer Science Department of Tsinghua University in 2017. In 2012, she received her B.S. degree in computer science and technology from Jilin University. Her research interests include largescale datacenter networks, content delivery networks, data-driven networks, and edge computing. She is currently with the Beijing University of Posts and Telecommunications as an associate professor.

Ran Wang received her B.E. degree from the software engineering department, Hunan University, Changsha, China, in 2018. She is currently a master student in the software engineering department of Beijing University of Posts and Telecommunications, Beijing, China. Her current research interests include traffic engineering (TE) and datacenter resource management.

Xiangyang Gong received his B.E. and M.E. degrees from Xi'an Jiaotong University, China, in 1992 and 1995, respectively, and a Ph.D. degree from the Beijing University of Posts and Telecommunications in 2012. He is now a professor at Beijing University of Posts and Telecommunications. His research interests include IP QoS, video communications, novel network architecture, artificial intelligence, and mobile internet.