J Supercomput (2018) 74:3016–3038
https://doi.org/10.1007/s11227-018-2349-y

Incentive-aware virtual machine scheduling in cloud computing

Heyang Xu[1] · Yang Liu[1] · Wei Wei[1] · Wenqiang Zhang[1]

Published online: 2 April 2018

Abstract As cloud computing is a market-oriented utility, optimal virtual machine (VM) scheduling in cloud computing should take into account the incentives for both cloud users and the cloud provider. However, most of existing studies on VM scheduling only consider the incentive for one party, i.e., either the cloud users or the cloud provider. Very few related studies consider the incentives for both parties, in which the cost, one of the most attractive incentives for cloud users, is not well addressed. In this paper, we investigate the problem of VM scheduling in cloud computing by optimizing the incentives for both parties. The problem is formulated as a multi-objective optimization model, i.e., maximizing the successful execution rate of VM requests and minimizing the combined cost (incentives for cloud users), and minimizing the fairness deviation of profits (incentive for the cloud provider). The proposed multi-objective optimization model can offer sufficient incentives for the two parties to stay and play in the cloud and keep the cloud system sustainable. A heuristic-based scheduling algorithm, called cost-greedy dynamic price scheduling, is then developed to optimize the incentives for both parties. Experimental results show that, compared with some popular algorithms, the developed algorithm can achieve higher successful

✉ Yang Liu
liu_yang@haut.edu.cn

Heyang Xu
xuheyang124@126.com

Wei Wei
nsyncw@126.com

Wenqiang Zhang
zhangwq@haut.edu.cn

[1] College of Information Science and Engineering, Henan University of Technology, Zhengzhou 450001, Henan, China

execution rate, lower execution cost, smaller fairness deviation and most important, higher degree of user satisfaction in most cases.

Keywords Cloud computing · Virtual machine scheduling · Incentives · Multi-objective optimization model · User satisfaction

1 Introduction

Cloud computing is a large-scale distributed computing paradigm in which a pool of computing resources, e.g., computation, storage and networking, is available to cloud users via the Internet [1, 2]. With the development of virtualization technology, cloud providers enable their users to submit job requests with specific resource demands and software stack (e.g., operation systems and applications) and then package them all together into virtual machines (VMs) [3, 4]. By submitting job requests to cloud providers, users no longer need to purchase and maintain sophisticated hardware for the resource usage in their peak load, thus reducing their total cost of ownership [5]. Cloud computing has now become the most emphasized information and communications technology (ICT) paradigm and is directly or indirectly used by almost every online users [6].

In cloud environments, users can submit their job requests anytime and anywhere. Once receiving a job request, the cloud provider should create a VM on a suitable physical machine and allocate required resources to guarantee quality of service (QoS), according to the user's job demands [7]. How to allocate VMs to suitable physical machines according to cloud users' QoS requirements is the VM scheduling problem studied in this paper. Since cloud computing is a market-oriented utility, optimal VM scheduling in cloud computing should allow the cloud provider and cloud users to focus on their own businesses to optimize their own incentives, respectively [8]. Job requests submitted by different cloud users may demand different amounts of resources. For instance, job requests for high-performance computing require more CPU cores, while big data processing applications require more memory [9, 10]. Furthermore, the cloud provider provisions heterogeneous virtual machine types with different resource configurations. If the cloud provider underestimates the provision of resources to cloud users, it would result in service-level agreement (SLA) violations and thus cause penalties. Otherwise, overestimating the provision would lead to resource under utilization and losing revenue as well.

VM scheduling in cloud computing is a complex problem, in which different concerns should be taken into account and then be properly addressed. Basically, these concerns can be classified into two categories, i.e., concerns of cloud users and those of cloud providers. From the perspective of cloud users, there are two major concerns in VM scheduling, i.e., *successful execution rate of the VM requests* (SERoV) (e.g., [11–14]), and *the combined cost* (which is the total execution cost of all users' job requests) incurred (e.g., [3, 7, 11, 15]). These two concerns are important ones since cloud users generally hope to successfully complete their submitted job requests (i.e., to meet their respective deadlines and budgets), and at the same time, at the lowest possible cost. If, on the contrary, users' job requests frequently miss their deadlines,

or the cost incurred is high, then cloud users experience low degree of user satisfaction and tend to lose interest in the cloud system and may finally leave it. Therefore, increasing SERoV is an incentive for cloud users, and so is reducing combined cost, which shall be considered in VM scheduling.

On the other hand, from the perspective of the cloud provider, the most interesting thing is to make the best use of the provided computing resources to make profits. However, if the workload on a computing resource (CR) is too high, this makes it likely that the VMs residing on the given CR cannot receive the required resources, which may lead to SLA violation and degrade the degree of user satisfaction. Frequent failures to fulfill cloud users' expectations inevitably damage the reputation of the cloud provider, which in the long run will lead to profit loss. In this respect, a major concern of the cloud provider is to make sure all the provided CRs have equal opportunities to offer their resources and make profits according to their resource capacities. In this paper, we call this concern as *profit fairness* among all the provided CRs. Therefore, increasing profit fairness, or equivalently reducing the *fairness deviation of profits* (FDoP, details are shown in Sect. 4.1.2) among all the provided CRs, is desirable from the perspective of the cloud provider [16, 17], which is an incentive for the cloud provider that shall also be considered in VM scheduling.

From the above discussions, it can be observed that, in cloud environments, there are different incentives for the two parties, i.e., cloud users and the cloud provider. These incentives for both parties should be considered and properly addressed so that a more comprehensive VM scheduling algorithm can be developed, which could help to provide both parties with sufficient incentives to stay and play in the cloud, leading to a sustainable system [12, 17]. Much attention has been paid on the field of VM scheduling in cloud computing [3, 7, 9–11, 14]. (Details are given in Sect. 2.) However, most of them are either from the incentive for cloud users or from the ones for the cloud provider to address this problem, and very few studies take into consideration the incentives for both parties.

This paper, in contrast, makes an endeavor to investigate the VM scheduling problem in cloud computing by addressing the major incentives for both parties, i.e., maximizing *the SERoV* and minimizing *the combined cost* (incentives for cloud users), and at the same time minimizing *the FDoP* (incentive for cloud provider). The problem of incentive-aware VM scheduling is thus formulated as a multi-objective optimization problem. Then, we develop a heuristic scheduling algorithm, called *Cost-Greedy Dynamic Price Scheduling* (CGDPS) algorithm, which can effectively allocate VMs to suitable physical resources. Simulation results show that the proposed CGDPS algorithm is effective and can achieve higher SERoV, lower cost, smaller FDoP and most important, higher degree of user satisfaction, compared with some popular algorithms (i.e., PSO [11], FCFS [15], MinCTT [18]) in most cases.

The rest of this paper is organized as follows: A detailed review of related work on VM scheduling in cloud computing is given in Sect. 2. The system model used in the paper is introduced in Sect. 3. Section 4 puts forward the incentives for cloud users and the cloud provider and proposes a heuristic scheduling algorithm called CGDPS algorithm. Simulations, results and analyses are given in Sect. 5. We conclude this paper in Sect. 6.

2 Related work

Many efforts have been put into the research on VM scheduling in cloud computing. For example, Yang and Chen expected [13] to enhance the successful ratio of backup tasks for scheduling MapReduce tasks and proposed an adaptive task allocation scheduler (ATAS) to determine the response time of backup tasks in heterogeneous cloud system. Their research focused on optimizing the execution time for MapReduce tasks. However, in cloud computing, there is another major concern of cloud users other than execution time, i.e., *the combined cost*. In order to reduce cloud users' execution cost, some cost-based approaches [3, 7, 14] have been proposed. For example, by considering different plans for renting resources from the cloud provider, Chaisiri et al. [3] formulated the VM scheduling problem as a stochastic programming model and proposed an OCRP algorithm with the aim of minimizing the total cost of cloud users. Zhang et al. [7] proposed a cost-efficient algorithm (ROSA) to find the feasible schedule that can minimize cloud users' total execution cost. Li et al. [14] also investigated the cost minimization-oriented VM scheduling problem in hybrid clouds and proposed an ODPA algorithm. On the basis of these research works, this paper further takes users' satisfactions into consideration.

To further improve the degree of users' satisfaction, some studies try to optimize both the execution time and cost for cloud users' job requests [11, 18–24]. For example, Garg et al. [18] presented a heuristic algorithm, *min–min cost time trade-off* (MinCTT), to manage the trade-off between the execution time and cost spent to execute users' jobs. Somasundaram and Govindarajan [11] proposed a particle swarm optimization (PSO)-based scheduling mechanism to minimize the total execution time and the total execution cost of users' submitted job requests. In [19], Singh and Chana developed a cloud workload management framework, which is actually a cloud workload analyzer (CWZ). Then, based on the developed CWZ, they [20] proposed a QoS metric-based scheduling algorithm to optimize the execution cost and execution time by analyzing and categorizing the workloads submitted by cloud users. Recently, Ran et al. [21] proposed a dynamic VM provisioning strategy for determining the number of the purchased VMs dynamically in order to minimize the total cost while keeping QoS. Wang et al. [22] also focused on the performance metrics from the view of users and proposed a resource provisioning algorithm, called max–min-cloud algorithm, to minimize the mean of the stochastic response time of users' request. Moreover, researches on workflow scheduling (e.g., [23] and [24]) mostly focused on optimizing the total execution time and the combined cost of users' workflows, as well. It can be noted that, although these researches cloud improve the degree of users' satisfaction to a certain extent, they all only focused on addressing the incentives for cloud users.

Some other existing studies tried to address the VM scheduling problem by considering the incentives for the cloud provider. For example, Zheng et al. [25] studied the VM provisioning jointed with physical servers' maintenance scheduling problem and proposed a heuristic scheduling algorithm to maximize the revenue of the cloud provider. Albagli-Kim et al. [26] proposed a dwindling job scheduling algorithm to maximize the profit of the cloud provider for processing cloud users' job requests. Wei et al. [9, 10] made an endeavor to best utilize cloud provider's physical resources by trying to avoid resource starvation where dominant resources are starved while non-

dominant resources are wasted. They proposed a heterogeneous resource allocation algorithm, SAMR, from the cloud provider's point of view. Yu et al. [27] propose a probabilistic guarantee scheme with the objective of minimizing the total migration overhead caused by inefficient migrations catering to dynamic and bursty resource demands of VMs. Besides, some researches (e.g., [5] and [28, 29]), which tried to optimize energy consumption of cloud data centers, are also from the cloud provider's respective to address the VM scheduling problem.

From the above review of the related works, we know that most of existing studies on VM scheduling in cloud computing only address the incentive for one party, i.e., either the cloud users or the cloud provider. However, cloud users and the cloud provider as the two main participated entities of a cloud system may have the different incentives. These incentives for both parties should be considered and properly addressed for providing the both parties with sufficient incentives to stay and play in the cloud and leading to a sustainable cloud system. Nevertheless, very few studies consider the incentives for both parties. For example, Tian et al. [30] investigated the placement of web applications, which is a similar problem as VM scheduling in cloud computing. They proposed a heuristic algorithm with the objectives of satisfying the resource demands of users' web applications as much as possible and at the same time, keeping load balance among the resource provider's physical machines. Recently, Liu and Shen [31] proposed a dependency-aware and resource-efficient scheduling (DRS) algorithm with the objectives of decreasing the response time and simultaneously improving the resource utilization. Gregory and Majumdar [32] investigated the resource management technique for scheduling batches of MapReduce tasks and proposed a resource management techniques by considering concerns of both cloud users (QoS requirement, i.e., deadline) and the cloud provider (i.e., energy consumption). These researches are from the perspectives of both the users and the resource provider; nevertheless, the combined cost, which is one of the most attractive incentives for cloud users [17], is not considered in deriving the scheduling algorithms. Although much attention has been paid on the field of VM scheduling, there are few studies considering incentives for both cloud users and the cloud provider, in which the combined cost, one of the most attractive incentives for cloud users, is not addressed. To the best of our knowledge, there is no existing research on addressing VM scheduling problem to optimize the SERoV and the combined cost for cloud users and simultaneously optimize FDoP for the cloud provider before. Therefore, this paper investigates the incentive-aware VM scheduling in cloud computing by taking into account the incentives for both cloud users and the cloud provider.

3 System model

This paper investigates the VM scheduling problem in cloud computing, where multiple customers may submit job requests at random instants with various workloads that should be fulfilled before the specified deadlines [7]. The system model generally consists of three main active entities, namely the cloud provider who owns the infrastructure resources, cloud users who are the resource consumers, and cloud scheduler.

Cloud users (such as individuals, small and medium enterprises) have demands to execute their jobs with certain QoS requirements. Generally, some important QoS factors that users most concern about are budget and deadline [7, 15, 17]. Each job may consist of several subtasks, and each subtask requires one CPU core to execute it. Subtasks in the same job should be executed concurrently on the same computing node [5, 12]. Denote by $\mathbf{J} = \{J_1, J_2, \ldots, J_m\}$ the $n(n \geq 1)$ job requests submitted by cloud users during the scheduling interval $[0, T)$. Each job request $J_i (1 \leq i \leq n)$ can be characterized by a six-tuple $J_i = (K_i, L_i, \text{mem}_i, t_i, b_i, d_i)$ [7, 11, 15, 18, 33], in which $K_i (K_i \geq 1)$ is the number of subtasks that job request J_i contains; $W_i = \{w_{ik} | 1 \leq k \leq K_i\}$ is the workload set of job request J_i's K_i subtasks, in which w_{ik} is the workload of the k:th subtask in job request J_i, in terms of millions of instructions (MI); mem_i represents the memory size required by job request J_i; $t_i (0 \leq t_i < T)$ is the arrival time that job request J_i arrives at the cloud provider's data center; b_i is job request J_i's budget constraint, which means that the cost of executing job request J_i must not exceed b_i; d_i represents its deadline by which the cloud user desires job request J_i to be completed. Before the jobs are executed, the desirable resources should be allocated.

The cloud provider owns infrastructure resources in its data center, which could be composed of hundreds or thousands of computing nodes. These computing nodes are heterogeneous since different computing nodes may have different CPU cores, different CPU processing speeds, different memory sizes and different prices. Without loss of generality, suppose that the cloud data center consists of $m(m \geq 1)$ heterogeneous computing nodes, denoted by $\mathbf{CN} = \{\text{CN}_1, \text{CN}_2, \ldots, \text{CN}_m\}$. Each computing node $\text{CN}_j (1 \leq j \leq m)$ can be characterized by a four-tuple $\text{CN}_j = (\text{Core}_j, \text{mem}_j, s_j, p_j)$ [12, 34, 35], in which Core_j and mem_j represent the number of CPU cores and memory size provided by computing node CN_j, respectively; s_j is the processing speed of computing node CN_j's CPU cores, in terms of million instructions per second (MIPS); p_j is the price of a CPU core, which equals to the cost of using a single CPU core of computing node CN_j per second. To provision resources for users' job requests, the cloud provider should first encapsulate all the job requests integrated with their own specific resource demands into different VM requests [3, 36]. Thus, each job request J_i can be packaged as a VM request V_i according to J_i's characterization $J_i = (K_i, L_i, \text{mem}_i, t_i, b_i, d_i)$. The packaged VM requests are then submitted to the cloud scheduler for scheduling. For convenience of reading, we use VM request instead of job request in the rest of the paper.

The cloud scheduler manages the collections of VM requests, gathering all computing nodes' resource information (such as available CPU cores, available memory size, the price, etc.), mapping each VM request to a suitable computing node and provisioning the required resources for the VM requests based on their QoS requirements. Once a VM request is successfully mapped onto a computing node and starts to execute, it will non-preemptively occupy the provisioned resources until its workloads are completed. After successfully completing the workloads of a VM request, the cloud scheduler is also responsible for releasing the provisioned resources. Then, the cloud provider charges cloud users for the provisioned resources.

Generally, the problem of VM scheduling in cloud computing can be described as follows: Suppose that a cloud data center consists of m heterogeneous computing

nodes $\mathbf{CN} = \{CN_1, CN_2, \ldots, CN_m\}$, and there are n VM requests $\mathbf{V} = \{V_1, V_2, \ldots, V_n\}$ submitted by cloud users, the problem VM scheduling is to allocate each VM request to a suitable computing node, as to optimize the incentives for both the cloud users and the cloud provider.

4 Problem formation and proposed algorithm

This paper focuses on the VM scheduling problem, i.e., how to determine the allocations of VM requests to computing nodes, taking into account the QoS guarantees, as well as the incentives for both the cloud users and the cloud provider. Cloud users may have various QoS requirements, and different computing nodes may have different processing speeds and different prices. As the two main active entities of a cloud system, cloud users and the cloud provider may also have different concerned incentives. For example, cloud users always desire their VM requests can be successfully allocated with low cost and high QoS guarantees. However, for the cloud provider, the most important concern is to make full use of its computing node resources to obtain high profit. All the above-mentioned challenges make VM scheduling problem more complex to be resolved.

4.1 Problem formulation

This paper formulates the VM scheduling problem in cloud computing as a multi-objective optimization model. Before presenting the details of the optimization model, we first introduce the incentives for both cloud users and the cloud provider.

4.1.1 Incentives for cloud users

From the perspective of cloud users, many objectives could be defined, but what attracts them most is that their VM requests can be successfully allocated and executed at low cost with high QoS guarantees. If the cost of VM allocation is too high or the job executions frequently miss their deadlines, users will lose interests in the cloud system. Consequently, this paper chooses *the successful execution ratio of VM requests* (SERoV) and *the combined cost* as the cloud users' incentives. The SERoV, denoted by θ, is the value of the number of successfully allocated VM requests divided by the total number of all VM requests submitted by cloud users, which can be given by

$$\theta = \frac{\sum_{i=1}^{n} \varphi_i}{n}, \tag{1}$$

where φ_i indicates whether VM request V_i is completed before its deadline. If VM request V_i is completed before its deadline, then $\varphi_i = 1$; otherwise, $\varphi_i = 0$.

The combined cost, denoted by C, is the sum of the costs for all the successfully allocated VM requests, which can be given by

$$C = \sum_{i=1}^{n} \sum_{j=1}^{m} \left[x_{ij} \cdot K_i \cdot \max_{1 \le k \le K_i} (w_{ik}/s_j) \cdot p_j \right], \tag{2}$$

where x_{ij} indicates whether VM request V_i is allocated to computing node CN_j. If VM request V_i is allocated to computing node CN_j, then $x_{ij} = 1$; otherwise, $x_{ij} = 0$. The term w_{ik}/s_j denotes the execution time of the k:th subtask of VM request V_i on computing node CN_j. The term $\max_{1 \le k \le K_i}(w_{ik}/s_j)$ is the execution time of VM request V_i on computing node CN_j, which is equal to the maximum execution time among VM request V_i's K_i subtasks. It means that if VM request V_i is successfully allocated to computing node CN_j, it will occupy all the provisioned resources until all its workloads are completed.

4.1.2 Incentive for the cloud provider

From the perspective of cloud provider, due to the limitation of the cost of investment, cloud provider's data center consists of many heterogeneous computing nodes. What's more, the cloud provider must offer customizable services, which makes itself face a problem, i.e., how to make full use of all its computing node resources to obtain profit. If a computing node can hardly get the opportunity to execute any VM request, it cannot be profitable for the cloud provider. Otherwise, if there are too many VM requests executing on a computing node, it may violate the QoS guarantees promised to the users, which cloud also reduce the cloud provider's profit. To make sure the fairness of obtaining profit among all the provided computing nodes can effectively avoid the above-mentioned situation [17, 37, 38]. *Fairness Deviation of Profits* (FDoP) among the computing nodes can ensure that each computing node should have equal opportunity to offer its resources and gain a fair profit according to its resource capacity. It means that a computing node can obtain the share of profit proportional to the capacity that it invests to the cloud system. The FDoP is attractive to both computing nodes with low resource capacity and those with high capacity. Therefore, this paper chooses the FDoP as the incentive for the cloud provider.

The obtained profit of computing node CN_j, denoted by $profit_j$, is the sum of the costs charged from the cloud users for successfully completing their VM requests, so $profit_j$ can be given by

$$\mathrm{profit}_j = \sum_{i=1}^{n} \left[x_{ij} \cdot K_i \cdot \max_{1 \le k \le K_i} (w_{ik} / s_j) \cdot p_j) \right]. \tag{3}$$

The profit rate of computing node CN_j, denoted by μ_j, is defined as the ratio of the obtained profit of CN_j divided by its total CPU capacity, so the calculation of μ_j can be given by

$$\mu_j = \frac{\mathrm{profit}_j}{\mathrm{Core}_j \cdot s_j}. \tag{4}$$

The FDoP among the computing nodes can be measured by the standard deviation of the profit rates of all the computing nodes. Denote by σ the FDoP among the computing nodes, then we have

$$\begin{aligned} \sigma &= \mathrm{std\_dev}(\mu_1, \mu_2, \ldots, \mu_m) \\ &= \sqrt{\frac{1}{m}\sum_{j=1}^{m}(\overline{\mu} - \mu_j)^2}, \end{aligned} \tag{5}$$

where $\overline{\mu}$ is the average value of the profit rates of all the computing nodes.

4.1.3 Multi-objective optimization model

In this paper, we make an endeavor to investigate the problem of VM scheduling by addressing the major incentives for both parties in cloud computing, i.e., *maximizing the SERoV* and *minimizing the combined cost* (incentives for cloud users), and at the same time *minimizing the FDoP* (incentive for the cloud provider). The problem of incentive-aware VM scheduling in cloud computing can thus be formulated as the following multi-objective optimization model.

Objectives:

$$\mathrm{Max} \quad \theta = \frac{\sum_{i=1}^{n}\varphi_i}{n} \quad \text{(I)}$$

$$\mathrm{Min} \quad C = \sum_{i=1}^{n}\sum_{j=1}^{m}\left[x_{ij} \cdot K_i \cdot \max_{1\le k\le K_i}(w_{ik}/s_j) \cdot p_j\right]; \quad \text{(II)}$$

$$\mathrm{Min} \quad \sigma = \sqrt{\frac{1}{m}\sum_{j=1}^{m}(\overline{\mu} - \mu_j)^2}. \quad \text{(III)}$$

Subject to:

(i)

$$x_{ij} \in \{0, 1\}, \quad \forall i \in \{1, 2, \ldots, n\}, \ j \in \{1, 2, \ldots, m\};$$

(ii)

$$\sum_{j=1}^{m} x_{ij} \leq 1, \quad \forall i \in \{1, 2, \ldots, n\};$$

(iii)

$$\sum_{i=1}^{n} x_{ij} \cdot K_i \leq \text{Core}_j, \quad \forall j \in \{1, 2, \ldots, m\};$$

(iv)

$$\sum_{i=1}^{n} x_{ij} \cdot \text{mem}_i \leq \text{Mem}_j, \quad \forall j \in \{1, 2, \ldots, m\};$$

(v) $\forall i \in \{1, 2, \ldots, n\}$, if $x_{ij} = 1$, then computing node CN_j must satisfy the following two restrictions:

$$wt_i + \max_{1 \leq k \leq K_i} (w_{ik}/s_j) \leq d_i; \tag{6}$$

$$K_i \cdot \max_{1 \leq k \leq K_i} (w_{ik}/s_j) \cdot p_j \leq b_i. \tag{7}$$

In the proposed multi-objective optimization model, optimization objective (I) maximizes the successful execution rate of VM requests, which tries to make sure that all the VM requests can be successfully executed. Optimization objective (II) minimizes the combined cost, which is another incentive for cloud users. It tries to reduce the total execution cost of all cloud users for their submitted job requests. Optimization objective (III) minimizes the fairness deviation of profits among cloud provider's all computing nodes. This objective tries to ensure that all the computing nodes' profit rates are the same, which means that all the computing nodes can get profits for the cloud provider according to their resource capacity. The proposed multi-objective optimization model can offer sufficient incentives for both the cloud users and the cloud provider, encourage the two parties to stay and play in the cloud system and keep the cloud system sustainable.

The first constraint defines the feasible range of decision variable x_{ij}. Constraint (ii) ensures that a VM request should be assigned to no more than one computing node. Constraint (iii) is CPU capacity restriction, which specifies that the total provisioned CPU cores should not exceed its available number. Constraint (iv) is memory capacity restriction, which is similar to constraint (iii). The last constraint indicates that if VM request V_i is allocated to computing node CN_j, then computing node CN_j must satisfy VM request V_i's deadline and budget restrictions. In Eq. (6), the term wt_i is the waiting time of VM request V_i; the term $\max_{1 \leq k \leq K_i}(w_{ik}/s_j)$ is the execution time of VM request V_i on computing node CN_j, which is equal to the maximum execution time among VM request V_i's K_i subtasks. Therefore, Eq. (6) constrains that the sum

of VM request V_i's waiting time and execution time must be less than its deadline. Equation (7) constrains that the execution cost of VM request V_i on computing node CN_j must be less than its budget.

This paper formulates the incentive-aware VM scheduling in cloud computing as a multi-objective optimization problem, i.e., *maximizing the SERoV* [objective (I)], *minimizing the combined cost* [objective (II)] and *minimizing the FDoP* [objective (III)]. It can be noted that, when substituting Eq. (3) and Eq. (4) into Eq. (5), the last optimization objective, (III), is not linear concerning the decision variable x_{ij}. Therefore, the proposed optimization model is a nonlinear optimization problem. Moreover, from the first two constraints we can see that the problem is a kind of combinatorial optimization problem, which has been proved to be a NP-hard problem [30]. The intractability of the problem increases exponentially with the number of variables if being handled with deterministic algorithms, such as exhaustive search. It becomes more challenging with the increase in the proliferation and complexity of cloud data centers [8]. Using an heuristic-based algorithm to tackle VM scheduling problem in cloud computing has received increasing attentions in recent years, as such an algorithm offers an NP-hard problem global solution acceptable in a time frame proportional to the number of variables [17, 39, 40]. Therefore, this paper develops a heuristic-based algorithm, called *cost-greedy dynamic price scheduling* (*CGDPS*) *algorithm*, to tackle the incentive-aware VM scheduling problem.

4.2 Proposed algorithm

Before giving the detailed description of the developed CGDPS algorithm, two definitions should be explicitly understood.

Definition 1 Candidate Node Set (CNS): For an arbitrary VM request $V_i \in \mathbf{V}$, if the resource capacity of a computing node $\mathrm{CN}_j \in \mathbf{CN}$ can satisfy VM request V_i's resource demands and, at the same time, the computing node CN_j can successfully complete VM request V_i's workloads before its deadline with the cost less than its budget, then the computing node CN_j is a candidate node for VM request V_i. All the candidate nodes of VM request V_i compose the candidate node set of VM request V_i, which can be denoted by CNS_i. If there is more than one candidate node for a VM request, then the developed algorithm will allocate the VM request to the candidate node with the lowest cost.

Definition 2 Scheduling Matrix (SM): Let $\mathbf{X} = (x_{ij})_{n\times m}$ be the scheduling matrix, in which x_{ij} is the decision variable of the proposed multi-objective optimization model (shown in Sect. 4.1.3). If VM request V_i is allocated to computing node CN_j, then $x_{ij} = 1$; otherwise, $x_{ij} = 0$. Actually, the scheduling matrix is the scheduling result obtained by a certain algorithm.

The pseudo-code of the developed CGDPS algorithm is shown in Algorithm 1. The inputs of CGDPS algorithm are the set of VM requests, $\mathbf{V} = \{V_1, V_2, \ldots, V_n\}$, and the set of computing nodes, $\mathbf{CN} = \{\mathrm{CN}_1, \mathrm{CN}_2, \ldots, \mathrm{CN}_m\}$. The output is the obtained scheduling matrix $\mathbf{X} = (x_{ij})_{n\times m}$. The detailed scheduling processes of CGDPS algorithm are as follows.

Algorithm 1: Cost-Greedy Dynamic Price Scheduling (CGDPS) algorithm

Input: the set of VM requests $\mathbf{V}=\{V_1,V_2,\ldots,V_n\}$; the set of computing nodes $\mathbf{CN}=\{CN_1, CN_2,\ldots, CN_m\}$.

Output: the scheduling matric $\mathbf{X}=(x_{ij})_{n\times m}$.

```
1  Initialization: mark all the VM requests' states in V as unscheduled and set X = 0 ;
2  foreach unscheduled VM request V_i(1≤i≤n) do
3     foreach computing node CN_j(1≤j≤m) do
4        if CN_j can satisfy V_i's resource demands and QoS requirements then
5           CNS_i = {CN_j} ∪ CNS_i ;
6        end if
7     end foreach
8     if  CNS_i = Φ  then
9        add VM request V_i to unsuccessful scheduled set U;
10       change VM require V_i's state to failed scheduling;
11    else if | CNS_i |=1 then
12          allocate VM request V_i to the candidate node in CNS_i;
13          change the resource capacity (CPU cores and memory size) of computing node CN_j;
14          set x_ij=1;
15          change VM request V_i's state to scheduled;
16       end if
17    end if
18 end foreach
//The following steps will process the VM requires who have more than one candidate nodes.
19 while there exists unscheduled VM require do
20    find the candidate node set CNS_i (|CNS_i |>1) which contains minimal elements;
21    find the computing node CN_j (from CNS_i) which can minimize the execution cost of V_i;
22    allocate VM require V_i to computing node CN_j;
23    change the resource capacity of computing node CN_j;
24    set x_ij=1;
25    change VM require V_i's state to successful allocation;
26    AdjustComputingNodePrice(CNS_i);
27 end while
```

Initially, all VM requests are marked with *unscheduled* state, and the values of all the elements of scheduling matrix **X** are set as 0 (line 1, Algorithm 1). Then, CGDPS algorithm iteratively executes the following processes (lines 2–27, Algorithm 1):

Firstly, for each unscheduled VM request V_i, CGDPS algorithm orderly checks m computing nodes, $\{\mathrm{CN}_1, \mathrm{CN}_2, \ldots, \mathrm{CN}_m\}$ whose subscripts are randomly generated, to find its candidate node set $\mathrm{CN}S_i$ (lines 3–7, Algorithm 1). During this process, if a certain candidate node set $\mathrm{CN}S_i$ is an empty set, which means that there is not any candidate node for VM request V_i and thus V_i cannot be successfully scheduled, then CGDPS algorithm changes VM request V_i's state to *failed scheduling* (lines 8–10, Algorithm 1); otherwise, if a certain candidate node set $\mathrm{CN}S_i$ contains only one candidate node CN_j, then CGDPS algorithm allocates VM request V_i to the computing node CN_j and changes the relevant parameters, namely VM request V_i's state, the value of the decision variable x_{ij} and computing node CN_j's available resource capacity, such as CPU cores and memory size (lines 11–16, Algorithm 1).

Secondly, for VM requests whose candidate node sets contain more than one candidate node, CGDPS algorithm orderly selects the VM request V_i whose can-

didate node set CNS_i contains the minimal number of candidate nodes [as shown in Eq. (8)] and allocates V_i to the candidate node CN_j (in CNS_i) which can minimize V_i's execution cost (lines 19–25, Algorithm 1). Then, CGDPS algorithm invokes the *AdjustComputingNodePrice*( ) function, as shown in Algorithm 2, to adjust the prices of corresponding computing nodes related to VM request V_i (line 26, Algorithm 1).

Continue the two steps mentioned above until there is no VM request in *unscheduled* state.

$$|\mathrm{CNS}_i| = \min_{1 \le k \le n} \{|\mathrm{CNS}_k|\}, \text{ s.t. } |\mathrm{CNS}_k| > 1. \tag{8}$$

The developed CGDPS algorithm contains a price adjusting function *AdjustComputingNodePrice*( ) whose aim is to adjust the prices of some related computing nodes to ensure the fairness of obtaining profits for all computing nodes. The pseudo-code of *AdjustComputingNodePrice*( ) function is shown in Algorithm 2. The inputs are the set of computing nodes $\mathbf{CN} = \{\mathrm{CN}_1, \mathrm{CN}_2, \ldots, \mathrm{CN}_m\}$, a certain successfully scheduled VM request V_i, the coefficient of increasing price α and the coefficient of decreasing price β. This function is invoked once a certain VM request V_i is successfully scheduled.

The detailed executing processes of the price adjusting function are as follows:

Firstly, the function needs to find all the qualified computing nodes, which can satisfy both the resource demands and deadline restriction of VM request V_i (lines 1–5, Algorithm 2). These qualified computing nodes compose VM require V_i's qualified computing node set, which can be denoted by QCS_i.

Algorithm 2: *AdjustComputingNodePrice*(V_i)

Input: the set of computing nodes **CN**={CN_1, CN_2,…, CN_m}; the scheduled VM require V_i; the coefficients of increasing and decreasing price α and β, respectively.

Output: calculated new price.

```
1  foreach computing node CN_j(1≤j≤m) do
2    if CN_j can satisfy V_i's resource requirement and deadline restriction then
3      QCS_i = {CN_j} ∪ QCS_i ;
4    end if
5  end foreach
6  foreach computing node CN_j ∈ QCS_i do
7    if CN_j is the computing node that VM require V_i is allocated to then
8      CN_j.price = α×CN_j.price;
9    else
10     CN_j.price = β×CN_j.price;
11   end if
12 end foreach
```

Secondly, the function changes the prices of the computing nodes in QCS_i. For each computing node CN_j ($\mathrm{CN}_j \in \mathrm{QCS}_i$), if CN_j is the computing node that executes VM request V_i, then the price of CN_j increases by a increasing coefficient α, which is a decimal slightly greater than 1, to avoid CN_j always being selected in the following steps (lines 7 and 8, Algorithm 2); otherwise, the price of CN_j decreases by a decreasing coefficient β, which is a decimal slightly less than 1, to avoid CN_j never being selected in the following steps (lines 9–11, Algorithm 2).

5 Performance evaluation

In this section, we discuss the performance evaluation of CGDPS algorithm by a series of experiments. All the experiments share the following configurations.

5.1 Simulation configurations

The simulated cloud data center totally consists of 100 computing nodes, and each node is characterized by four parameters, i.e., number of CPU cores, memory size, processing speed and price. The number of CPU cores of each computing node is an integer uniformly distributed in the range of {2, 4, 8, 16}. Each computing node also has different memory sizes whose value is generated by a uniform distribution in the range of {4 GB, 8 GB, 16 GB, 32 GB}. The processing speed of each computing node is uniformly distributed within the range [100, 200] with the average speed of 150 MIPS. The initial prices of all computing nodes are generated by a uniform distribution within the range [0.35, 1]. Generally, the prices of computing nodes have roughly linear relationship with their processing speed, which makes sure that a faster node needs more execution cost than a slower one for executing a same task.

The parameters of each VM request includes arrival time, required memory size, number of subtasks, workloads of each subtask, deadline and budget. In the experiments, we assume all the VM requests needed to be scheduled are submitted within a scheduling interval ($T = 100$ s) and their arrival times are generated by a uniform distribution within the range (0, 100]. The required memory sizes are randomly distributed within the range of {1 GB,2 GB,3 GB,4 GB}. Each VM request consists of 1–7 subtasks whose value is uniformly generated with the average number of four subtasks. The workload of each subtask is considered as a random integer uniformly distributed within the range of {100,000, 120,000, 140,000, 160,000,…, 500,000} MI. The deadline restriction allocated to each VM request is set as the sum of average estimated runtime with a 10% variation. The budget allocated to each VM request is set as the value of the maximum workload of all its subtasks divided by the average processing speed of all computing nodes, multiplying by the number of subtask and the average price with a 10% variation. Most of the configurations mentioned above are similar to those adopted in [11, 15, 17, 18]. In order to avoid the influence of causal factors, we run each experiment 1000 times, and the results presented in this paper are the mean value of the results obtained by the 1000 experiments.

It is to be noted that the resource attributes of CNs can be obtained according to their configurations and the attribute information (such as CPU core and memory requirements, task length, etc.) of VM requests can also be achieved by some prediction techniques, such as [41, 42]. The two twofold inputs of the proposed algorithm can be both obtained by existing techniques and thus the simulation configurations are feasible.

5.2 Performance metrics

In order to verify the efficiency of the developed CGDPS algorithm, we compare it with the other three related algorithms, i.e., PSO [11], FCFS [15] and MinCTT [18], with the following five performance metrics. The first performance metric is the *successful execution ratio of VM requests* (SERoV), which can be calculated by Eq. (1). The second metric is the *average execution cost* (AEC), which is the total combined cost [calculated by Eq. (2)] divided by the number of successfully executed VM requests. The third metric is the *average makespan* (AMS), which is the average value of all the successfully executed VM requests' makespans [11]. The makespan of a successfully executed VM request is defined as the time span between its submitted time and completed time. The fourth metric is the *fairness deviation of profit* (FDoP) among the computing nodes, which can be calculated by Eq. (5). The last performance metric is the *overall user satisfaction* (OUS), which is the sum of the satisfactions of all cloud users according to the obtained scheduling result. Denote by us_i the satisfaction of the cloud user, who submits the VM require V_i, according to the scheduling result, so we have

$$\mathrm{US} = \sum_{i=1}^{n} \mathrm{us}_i . \tag{9}$$

In Eq. (9), us_i is determined by three factors, i.e., whether VM request V_i is successfully executed or not, the makespan and the execution cost of VM request V_i. Denote by $makespan_i$ and $cost_i$ the makespan and the execution cost of VM request V_i, respectively, so we have

$$\mathrm{us}_i = \varphi_i \times \left[(d_i - \mathrm{makespan}_i)/d_i + (b_i - \mathrm{cost}_i)/b_i \right]. \tag{10}$$

5.3 Simulation results

5.3.1 Experiment 1

To evaluate the performance of the proposed CGDPS algorithm, we randomly generate six groups of VM requests in this experiment by the methods mentioned above. In the six VM request groups, the number of computing nodes is the same, i.e., 100 computing nodes, and the numbers of VM requests are 75, 100, 125, 150, 175 and 200, respectively. The six groups simulate the cloud environments with general system workloads, which can be used to evaluate the performance of the proposed algorithm under general system workloads.

Successful execution ratio of VM requests (SERoV) Successful execution of a VM request means that the VM request is completed before its deadline with the cost less than its budget. The higher SERoV a cloud data center provides, the more interest users have in performing their businesses through the data center. The results of SERoV obtained by the four compared algorithms are listed in Table 1.

Table 1 Results of SERoV obtained by different algorithms with varied system workloads

No. of VM requires	Algorithm			
	FCFS (%)	MinCTT (%)	PSO (%)	GDPS (%)
75	93.26	99.77	99.31	**99.91**
100	89.04	98.52	98.31	**99.74**
125	84.12	95.93	94.69	**99.48**
150	78.57	88.42	87.3	**96.17**
175	72.38	82.54	81.61	**89.7**
200	61.89	74.63	73.65	**80.64**

Bold values reflects the optimal value of each line

From the results listed in Table 1, it can be observed that the proposed CGDPS algorithm can obtain the highest successful execution rate. Specifically, when the numbers of VM requests are 100, 125, 150, 175 and 200, the SERoV obtained by CGDPS algorithm is 10.7, 15.36, 17.6, 17.32 and 18.75%, respectively, higher than that of FCFS algorithm, 1.22, 3.55, 7.75, 7.16 and 6.01%, respectively, higher than that of MinCTT algorithm and 1.43, 4.79, 8.87, 8.09 and 6.99%, respectively, higher than that of PSO algorithm.

The reason is as follows: For the VM requests with tight QoS constraints, few computing nodes can satisfy their requirements. Assigning high priority to the VM requests with tight QoS constraints can increase the possibility of these VM requests to be scheduled. The proposed CGDPS algorithm exactly adopts the idea mentioned above and preferentially schedules the VM requests with tight QoS constraints, as presented in lines 11–16 and 19–22 of Algorithm 1. Therefore, CGDPS algorithm can obtain the highest successful execution ration of VM requests among the compared algorithms.

Also, from Table 1, it can be seen that the SERoV obtained by the four algorithms all decreases with the increasing of the number of VM requests. This is because with the increasing of VM requests, the system workload increases, but the resource capacity of the data center is fixed (i.e., 100 computing nodes), and thus, it leads to the decrease in the SERoV.

Furthermore, Fig. 1 shows the results of the number of successfully executed VM requests (SEVR) obtained by different algorithms. As shown in Fig. 1, the proposed CGDPS algorithm can successfully execute more VM requests than the other compared algorithms. Specifically, when the numbers of VM requests are 125, 150, 175 and 200, the SEVR obtained by CGDPS algorithm are 12.97, 15.6, 14.65 and 14.98%, respectively, higher than that of FCFS algorithm, 5.17, 8.38, 7.98 and 7.18%, respectively, higher than that of MinCTT algorithm and 6.39, 9.36, 8.94 and 7.73%, respectively, higher than that of PSO algorithm. The trend of the metric of SEVR is the same with that of the metric of SERoV. The reason lies in that the proposed CGDPS algorithm preferentially schedules the VM requests with tight QoS constraints.

Average execution cost (AEC) Average execution cost of all VM requests can be calculated by the total combined cost [Eq. (2)] divided by the number of successfully

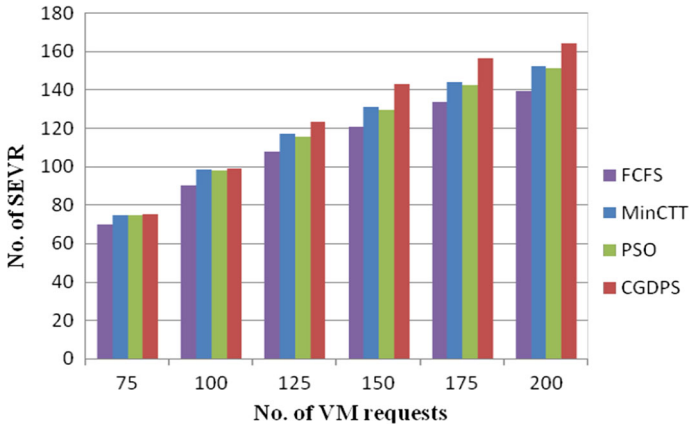

Fig. 1 Number of SEVR obtained by different algorithms with varied system workloads

Table 2 Results of AEC obtained by different algorithms with varied system workloads

No. of VM requires	Algorithm			
	FCFS	MinCTT	PSO	GDPS
75	2.502	2.329	2.238	**2.041**
100	2.519	2.234	2.176	**1.994**
125	2.467	2.157	2.089	**1.972**
150	2.395	2.136	2.057	**2.043**
175	2.124	2.033	1.92	**1.9**
200	**1.826**	1.936	1.88	1.944

Bold values reflects the optimal value of each line

executed VM requests. The smaller the average execution cost is, the lower the cost (users need to pay for their VM requests) is. The results of AEC obtained by different algorithms are listed in Table 2. It can be seen that when the numbers of VM requests are 75, 100, 125, 150 and 175, the average execution cost obtained by CGDPS algorithm is 18.42, 20.84, 20.06, 14.7 and 10.54%, respectively, smaller than that obtained by FCFS algorithm, 12.36, 10.74, 8.58, 4.35 and 6.54%, respectively, smaller than that obtained by MinCTT algorithm, and 8.8, 8.36, 5.6, 0.68 and 1.04%, respectively, smaller than that obtained by PSO algorithm.

The proposed CGDPS algorithm can obtain the lowest average execution cost among the compared algorithms in almost all the varied system workload situations. This is because the proposed CGDPS algorithm tries to schedule a VM request to the candidate computing node with the lowest cost (shown in line 21 of Algorithm 1), which can lead to a lower execution cost.

Average makespan (AMS) This performance metric indicates the average time that an algorithm needs to complete a VM request. The smaller the value of average makespan is, the shorter the time that the corresponding algorithm needs to complete users' VM requests is. Table 3 shows the results of average makespan of all successfully executed

Table 3 Results of AMS obtained by different algorithms with varied system workloads

No. of VM requires	Algorithm			
	FCFS	MinCTT	PSO	GDPS
75	0.692	**0.648**	0.724	0.772
100	0.704	**0.661**	0.716	0.752
125	0.69	**0.666**	0.702	0.741
150	0.69	**0.677**	0.697	0.747
175	0.683	**0.68**	0.69	0.741
200	**0.668**	0.677	0.671	0.747

Bold values reflects the optimal value of each line

VM requests obtained by different algorithms. It can be seen that MinCTT algorithm can obtain the shortest average makespan and the proposed CGDPS algorithm does not perform well on this metric. The reason for this is that makespan is one of the optimization objectives of MinCTT algorithm and the algorithm always schedules a VM request to the computing node which can minimize the weighting sum of execution cost and makespan. However, when CGDPS algorithm schedules a VM request, makespan is just as one of the restrictions [shown in Eq. (6)], which is used to determine whether a computing node is the candidate node for the VM request (shown in Definition 1). Thus, the proposed CGDPS algorithm cannot perform well on the performance metric of average makespan.

Moreover, although FCFS algorithm can obtain relatively low average execution cost and short average makespan, the successful execution rate of VM requests obtained by the algorithm is much lower than other algorithms. The VM requests rejected by FCFS algorithm are those with tight budget or deadline constriction. In other words, these rejected VM requests are either the ones who contain too many subtasks or the ones whose workloads of subtasks are too long. Most of its successfully executed VM requests are the ones with a few subtasks or with short subtask workload. Thus, FCFS algorithm performs relatively well on the two metrics.

Fairness deviation of profits (FDoP) The fairness deviation of profits means that all computing nodes provided by the cloud provider have equal opportunities to offer their resources and can obtain fair profits according to their resource capacities. FDoP, shown in Eq. (5), indicates the dispersion of all computing nodes' profit rates. The smaller the value of FDoP is, the fairer all the computing nodes' profits are. Table 4 shows the results of FDoP obtained by different algorithms with varied system workloads. As listed in Table 4, when the numbers of VM requests are 75, 100, 125, 150, 175 and 200, the FDoP obtained by CGDPS algorithm is 45.7, 41.7, 38.2, 30.5, 30.3 and 29.9%, respectively, smaller than that of FCFS algorithm, 32.1%, 28.9.4, 25.6, 18.3, 18.7 and 20.5%, respectively, smaller than that of MinCTT algorithm, and 27.5, 24.7, 23.1, 16.4, 18.7 and 20.8%, respectively, smaller than that of PSO algorithm.

It can be seen that the proposed CGDPS algorithm can obtain the lowest fairness deviation of profits. The reason is that once CGDPS algorithm successfully schedules a VM request, the price adjusting function (shown in Algorithm 2) will be invoked to

Table 4 Results of FDoP obtained by different algorithms with varied system workloads

No. of VM requires	Algorithm			
	FCFS	MinCTT	PSO	GDPS
75	0.315	0.252	0.236	**0.171**
100	0.324	0.266	0.251	**0.189**
125	0.319	0.265	0.256	**0.197**
150	0.308	0.262	0.256	**0.214**
175	0.293	0.251	0.251	**0.204**
200	0.277	0.244	0.245	**0.194**

Bold values reflects the optimal value of each line

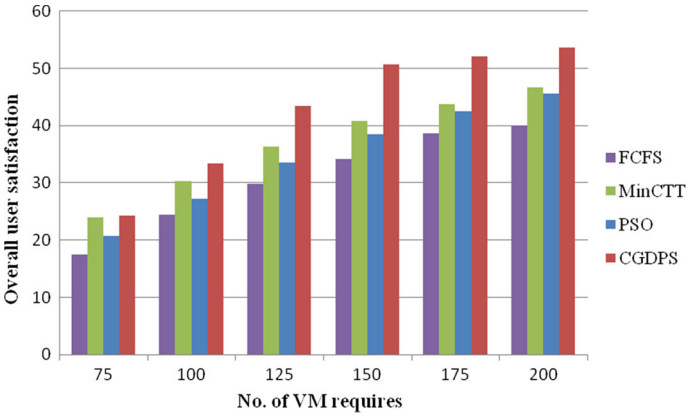

Fig. 2 Overall user satisfaction obtained by different algorithms with varied system workloads

execute to dynamically adjust the price of all computing nodes in its candidate node set. If a candidate computing node obtains the VM request, then CGDPS algorithm slightly increases its price to prevent it from always being selected in following scheduling; otherwise, CGDPS algorithm slightly decreases its price to avoid the computing node never being selected in following scheduling. Thus, the proposed algorithm can ensure FDoP better than the other compared algorithms.

Overall user satisfaction (OUS) The OUS represents the degree of users' satisfaction based on the obtained scheduling result (i.e., the scheduling matrix). Higher OUS means meeting user satisfaction better. The comparison results of OUS obtained by different algorithms are presented in Fig. 2. It can be seen that the proposed CGDPS algorithm can obtain the highest degree of user satisfaction. Specifically, when the numbers of VM requests are 75, 100, 125, 150, 175 and 200, the OUS of CGDPS algorithm are 38.4, 37.1, 45.6, 48.5, 34.6 and 34.2%, respectively, higher than that of FCFS algorithm, 1.2, 10, 19.3, 24.2, 18.8 and 15.1%, respectively, higher than that of MinCTT algorithm, and 18.6, 22.4, 29.7, 31.7, 22.5 and 17.8%, respectively, higher than that of PSO algorithm. It is evident that the proposed CGDPS algorithm can meet users' satisfaction best.

Table 5 Results of all metrics obtained by different algorithms under a large-scale data center

Metrics	Algorithm			
	FCFS	MinCTT	PSO	GDPS
SERoV	78.59%	85.63%	86.12%	**91.92%**
AEC	2.41	2.07	1.99	**1.89**
AMS	**0.69**	0.691	0.697	0.727
FDoP	0.243	0.218	0.196	**0.139**
OUS	249.1	303.2	289.7	**359.6**

Bold values reflects the optimal value of each line

5.3.2 Experiment 2

To evaluate the performance of the proposed algorithm in a large-scale cloud environment, this experiment randomly generates a cloud data center with 600 computing nodes and 1000 VM requests submitted by cloud users, using the methods discussed in Sect. 5.1.

Table 5 shows the results of the five performance metrics obtained by the compared algorithms under a large-scale cloud data center. It can be seen that, in most cases, the comparison results are similar to the results presented in Sect. 5.3.1. In particular, the overall user satisfaction (OUS) obtained by CGDPS algorithm is comparatively 44.3% higher than FCFS algorithm, 18.6% higher than MinCTT algorithm and 24.1% higher than PSO algorithm. It means that CGDPS algorithm can guarantee cloud users' QoS requirements better. Therefore, we can conclude that the proposed CGDPS algorithm is effective to be adopted in large-scale cloud data centers.

5.3.3 Experiment 3

In this section, we verify the practicability of the proposed CGDPS algorithm under real-life workload traces [43, 44], which are also widely adopted in related research. We use *DAS2 fs1* trace as the simulated workload because it records the similar resource attributes required by a VM request. There are total 40,315 records in the workload, from which we choose 196 ones as VM requests in the experiment to limit the simulation time. The 196 records are selected by a principle that the number of subtasks contained in a record must be in the range of 1–7. Each record has detailed descriptions of the submitted time, runtime, required CPU cores and memory size, and some other descriptions. However, some QoS requirements, such as deadline and budget constraints, still need to be randomly generated using the methods discussed in Sect. 5.1 since the real-life workload trace does not contain any information about them. Also, we also normalize the submitted times of the selected records within the range of [0, 100] so that we can schedule all of them in a scheduling interval.

Table 6 shows the results of the five performance metrics obtained by different algorithms under the real-life workload trace of *DAS2 fs1*. As listed in Table 6, the proposed CGDPS algorithm can achieve the highest successful execution rate of VM requests, relatively lower average execution cost, the lowest fairness deviation of profit, and, most important, the highest degree of user satisfaction.

Table 6 Results of all metrics obtained by different algorithms under real-life workload trace

Metrics	Algorithm			
	FCFS	MinCTT	PSO	GDPS
SERoV	82.94%	88.55%	90.42%	**98.43%**
AEC	1.13	1.27	**1.05**	1.09
AMS	0.58	**0.52**	**0.52**	0.63
FDoP	0.72	0.68	0.65	**0.54**
OUS	48.65	52.68	50.5	**59.32**

Bold values reflects the optimal value of each line

This section conducts experiment under the real workload trace. From the results and analysis presented above, we can find that the proposed algorithm can also perform pretty well under the real-life workload trace. Therefore, it can be concluded that the proposed CGDPS algorithm can be practically applied to real-life cloud environments.

6 Conclusions

Cloud users and cloud provider are two important entities in a cloud system, and they make autonomous scheduling decisions with different incentives of interest, which makes the problem of VM scheduling in cloud computing more complex than ever before.

This paper formulates VM scheduling in cloud computing as a multi-objective optimization problem and develops a heuristic-based CGDPS algorithm. In each iterative step, CGDPS algorithm sets high priority to VM requests with few candidate nodes and schedules a VM request to the candidate computing node with the lowest cost. After successfully scheduling a VM request, CGDPS algorithm adopts the price adjusting function to dynamically adjust the price of all qualified computing nodes to avoid the fairness deviation becoming worse. The simulation results clearly illustrate that the developed method can achieve the highest degree of users' satisfaction.

Although this paper considers the scenario of multi-core physical machines, there is a constraint that each of PM's CPU cores cannot be allocated to more than one VM. However, in modern cloud environments, multi-core physical machines are more promising and each of PM's CPU cores can be shared by cores of multiple multi-core VMs. So as a future work, we will conduct research on the more real-life scenario of multi-core PMs, which may lead to a more complex scheduling issue.

Acknowledgements This work was supported by the National Natural and Science Foundation of China (Nos. 61472460, 61702162 and U1504607), Foundations for Science and Technology of Henan province (Nos. 162102210044, 172102110013), Key Science and Technology Research Project of Education Department Henan Province (No. 17A520004), Program for Innovative Research Team (in Science and Technology) in University of Henan Province (No. 17IRTSTHN011) and the Research Foundation for Advanced Talents of Henan University of Technology.

References

1. Armbrust M, Fox A, Griffith R et al (2010) A view of cloud computing. Commun ACM 53(4):50–58
2. Mell P, Grance T (2011) NIST Special Publication 800-145: The NIST definition of cloud computing
3. Chaisiri S, Lee BS, Niyato D (2012) Optimization of resource provisioning cost in cloud computing. IEEE Trans Serv Comput 5(2):164–177
4. Mann ZÁ (2015) Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. ACM Comput Surv 48(1):11
5. Beloglazov A, Buyya R (2012) Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concur Comput Pract Exp 24(13):1397–1420
6. Madni SHH, Latiff MSA, Coulibaly Y (2016) Resource scheduling for infrastructure as a service (IaaS) in cloud computing: challenges and opportunities. J Netw Comput Appl 68:173–200
7. Zhang R, Wu K, Li M et al (2016) Online resource scheduling under concave pricing for cloud computing. IEEE Trans Parallel Distrib Syst 27(4):1131–1145
8. Zhan ZH, Liu XF, Gong YJ et al (2015) Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Comput Surv 47(4):63
9. Wei L, He B, Foh CH (2014) Towards multi-resource physical machine provisioning for IaaS clouds. In: Proceeding of IEEE International Conference on Communications (ICC), pp 3469–3472
10. Wei L, Foh CH, He B et al (2015) Towards efficient resource allocation for heterogeneous workloads in IaaS clouds. In: IEEE Transactions on Cloud Computing, vol 99, p 1
11. Somasundaram TS, Govindarajan K (2014) CLOUDRB: a framework for scheduling and managing high-performance computing (HPC) applications in science cloud. Future Gener Comput Syst 34:47–65
12. Mashayekhy L, Nejad MM, Grosu D et al (2014) Incentive-compatible online mechanisms for resource provisioning and allocation in clouds. In: Proceedings of IEEE 7th International Conference on Cloud Computing (CLOUD), pp 312–319
13. Yang SJ, Chen YR (2015) Design adaptive task allocation scheduler to improve MapReduce performance in heterogeneous clouds. J Netw Comput Appl 57:61–70
14. Li S, Zhou Y, Jiao L et al (2015) Towards operational cost minimization in hybrid clouds for dynamic resource provisioning with delay-aware optimization. IEEE Trans Serv Comput 8(3):398–409
15. Singh S, Chana I (2015) QRSF: QoS-aware resource scheduling framework in cloud computing. J Supercomput 71(1):241–292
16. Xiao L, Zhu Y, Ni LM, Xu Z (2008) Incentive-based scheduling for market-like computational grids. IEEE Trans Parallel Distrib Syst 19(7):903–913
17. Xu H, Yang B (2015) An incentive-based heuristic job scheduling algorithm for utility grids. Future Gener Comput Syst 49:1–7
18. Garg SK, Buyya R, Siegel HJ (2010) Time and cost trade-off management for scheduling parallel applications on utility grids. Future Gener Comput Syst 26(8):1344–1355
19. Singh S, Chana I (2014) Metrics based workload analysis technique for IaaS cloud. In: Proceedings of International Conference on Next Generation Computing and Communication Technologies, pp 1–7
20. Singh S, Chana I (2015) Q-aware: quality of service based cloud resource provisioning. Comput Electr Eng 47:138–160
21. Ran Y, Yang J, Zhang S et al (2017) Dynamic IaaS computing resource provisioning strategy with QoS constraint. IEEE Trans Serv Comput 10(2):190–202
22. Wang Z, Hayat MM, Ghani N et al (2017) Optimizing cloud-service performance: efficient resource provisioning via optimal workload allocation. IEEE Trans Parallel Distrib Syst 28(6):1689–1702
23. Rodriguez MA, Buyya R (2014) Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans Cloud Comput 2(2):222–235
24. Xu H, Yang B, Qi W, Ahene E (2016) A multi-objective optimization approach to workflow scheduling in clouds considering fault recovery. KSII Trans Internet Inf 10:976–995
25. Zheng Z, Li M, Xiao X et al (2013) Coordinated resource provisioning and maintenance scheduling in cloud data centers. In: Proceedings of IEEE conference on computer communications, pp 601–609
26. Albagli-Kim S, Shachnai H, Tamir T (2014) Scheduling jobs with dwindling resource requirements in clouds. In: Proceedings of IEEE Conference on Computer Communications, pp 601–609
27. Yu L, Chen L, Cai Z et al (2017) Stochastic load balancing for virtual resource management in datacenters. In: IEEE Transactions on Cloud Computing, 2017, published online. https://doi.org/10.1109/tcc.2016.2525984

28. Xu H, Yang B (2017) Energy-aware resource management in cloud computing considering load balance. J Inf Sci Eng 33(1):1–16
29. Mishra SK, Puthal D, Sahoo B et al (2018) An adaptive task allocation technique for green cloud computing. J Supercomput 74(1):370–385
30. Tian C, Jiang H, Iyengar A et al (2011) Improving application placement for cluster-based web applications. IEEE Trans Netw Serv Manage 8(2):104–115
31. Liu J, Shen H (2016) Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds. In: Proceedings of 8th IEEE International Conference on Cloud Computing Technology and Science, pp 110–117
32. Gregory A, Majumdar S (2016) A configurable energy aware resource management technique for optimization of performance and energy consumption on clouds. In: Proceedings of 8th IEEE International Conference on Cloud Computing Technology and Science, pp 184–192
33. Van BR, Vanmechelen K, Broeckhove J (2010) Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In: Proceedings of the IEEE 3rd International Conference on Cloud Computing, pp 228–235
34. Takouna I, Dawoud W, Meinel C (2012) Energy efficient scheduling of hpc-jobs on virtualize clusters using host and vm dynamic configuration. ACM SIGOPS Oper Syst Rev 46(2):19–27
35. Abdullahi M, Ngadi MA (2016) Symbiotic organism search optimization based task scheduling in cloud computing environment. Future Gener Comput Syst 56:640–650
36. Gutierrez-Garcia JO, Ramirez-Nafarrate A (2015) Collaborative agents for distributed load management in cloud data centers using live migration of virtual machines. IEEE Trans Serv Comput 8(6):916–929
37. Xu B, Zhao C, Hu E et al (2011) Job scheduling algorithm based on Berger model in cloud environment. Adv Eng Softw 42(7):419–425
38. Wang W, Li B, Liang B (2014) Dominant resource fairness in cloud computing systems with heterogeneous servers. In: Proceedings of the IEEE Conference on Computer Communications, pp 583–591
39. Zhang J, Zhan Z, Lin Y et al (2011) Evolutionary computation meets machine learning: a survey. IEEE Comput Intell Mag 6(4):68–75
40. Chen N, Chen WN, Gong YJ et al (2015) An evolutionary algorithm with double-level archives for multiobjective optimization. IEEE Trans Cybern 45(9):1851–1863
41. Jang S, Taylor V, Wu X et al (2005) Performance prediction-based versus load-based site selection: quantifying the difference. In: Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems, pp 148–153
42. De Felice M, Yao X (2011) Short-term load forecasting with neural network ensembles: a comparative study. IEEE Comput Intell Mag 6(3):47–56
43. Feitelson DG, Tsafrir D, Krakov D (2014) Experience with using the parallel workloads archive. J Parallel Distrib Comput 74(10):2967–2982
44. http://www.cs.huji.ac.il/labs/parallel/workload/ (2014)