Physical Machine Resource Management in Clouds: A Mechanism Design Approach

Lena Mashayekhy, *Student Member, IEEE,* Mahyar Movahed Nejad, *Student Member, IEEE, and* Daniel Grosu, *Senior Member, IEEE*

Abstract—We address the problem of physical machine resource management in clouds considering multiple types of physical machines and resources. We formulate this problem in an auction-based setting and design optimal and approximate strategy-proof mechanisms that solve it. Our proposed mechanisms consist of a winner determination algorithm that selects the users, provisions the virtual machines (VMs) to physical machines (PM), and allocates them to the selected users; and a payment function that determines the amount that each selected user needs to pay to the cloud provider. We prove that our proposed approximate winner determination algorithm satisfies the loser-independent property, making the approximate mechanism robust against strategic users who try to manipulate the system by changing other users' allocations. We show that our proposed mechanisms are strategy-proof, that is, the users do not have incentives to lie about their requested bundles of VM instances and their valuations. In addition, our proposed mechanisms are in alignment with green cloud computing strategies in which physical machines can be powered on or off to save energy. Our theoretical analysis shows that the proposed approximation mechanism has an approximation ratio of 3. We perform extensive experiments in order to investigate the performance of our proposed approximation mechanism compared to that of the optimal mechanism.

Index Terms—cloud computing, mechanism design, energy efficient resource management.

1 Introduction

The ever-growing demand for cloud resources from businesses and individuals places the cloud resource management at the heart of the cloud providers' decision-making process. A cloud provider offers infrastructure as a service (IaaS) by selling low level resources of its physical machines (PMs) in the form of virtual machines (VMs). These services are made available to users as utilities in a pay-as-you-go model, reducing the operational costs for the users. The cloud auction market follows the pay-as-you-go model, and it has proven to be beneficial for both users and cloud providers. This is due to the fact that in such market, cloud providers can attract more customers and better utilize their resources, while users can obtain services at a lower price than in the on-demand market.

We consider the physical machine resource management problem in the presence of multiple PMs and multiple types of resources (e.g., cores, memory, storage) in an auction-based setting, where each user bids for a bundle of heterogeneous VM instances. Bundles of heterogeneous VM instances are required by several types of applications, such as social game applications composed of three layers: front-end web server, load balancing, and back-end data storage. These types of applications require a bundle of heterogeneous VMs composed of communication-intensive VMs, computation-intensive VMs, and storage-intensive VMs, respectively [1]. The requests of the selected users are assigned to PMs, where a PM can be a server, a rack of servers, or a group of racks, and it may host several VMs. Each user has some private information about her requested bundle of VM instances and a private valuation for the bundle. This information is not publicly known by the cloud provider and the other users. The users are self-interested in a sense that they want to maximize their own utility. The cloud auction market could be vulnerable to such self-interested users' behaviors. It may be beneficial for the cloud users to manipulate the auction outcomes and gain unfair advantages by untruthfully revealing their requests (i.e., different VM bundles or bids from their actual request). Strategic behaviors of any user may hinder other qualified users, significantly reducing the auction efficiency, and discouraging users from participation. One of the goals in such settings is to design strategy-proof mechanisms, that is, mechanisms that induce the users to report their true requests and valuations.

Our proposed PM resource management mechanisms consist of three phases: winner determination, provisioning and allocation, and pricing. In the winner determination phase, the cloud provider decides which users receive their requested bundles. In the provisioning and allocation phase, the cloud provider provisions the amount of resources in the form of VM instances onto the PMs, and then allocates the requested bundles of VMs to the winning users. In the pricing phase, the cloud provider dynamically determines the price that the winning users should pay for their requests.

The winner determination phase of the PM resource management problem (PMRM) can be reduced to the

• *L. Mashayekhy, M. M. Nejad, and D. Grosu are with the Department of Computer Science, Wayne State University, Detroit, MI, 48202. E-mail: mlena@wayne.edu, mahyar@wayne.edu, dgrosu@wayne.edu*

multiple multidimensional knapsack problem (MMKP). In this setting, each PM is considered to be one multidimensional knapsack. The bundle of VMs from a user request is considered as an item. The aim is to select a subset of items for each knapsack maximizing the value. Chekuri and Khanna [2] showed that the multiple knapsack problem (MKP) is strongly NP-hard (even in the case of two knapsacks) using a reduction from the Partition problem. The MMKP problem is much harder than the MKP problem and is thus also strongly NP-hard. Sophisticated algorithms for solving MMKP do not necessarily satisfy the properties required to achieve strategy-proofness, and they need to be specifically designed to satisfy those properties. On the other hand, another desirable property of such algorithms in cloud auction settings is to have a very small execution time.

A major factor that a cloud provider needs to take into account when offering VM instances to users is pricing the VMs based on the market demand. Such pricing functions should consider the incentives of both cloud providers and users. Amazon reported that most users have saved between 50% and 66% by bidding in its spot market (Amazon auction market) compared to standard on demand market [3]. Dynamic pricing is an efficient way to improve cloud providers revenue [4]. Instead of exclusively selling VMs employing a fixed-price model, cloud providers such as Amazon EC2 employ auction-based models, where users submit bids for their requested VM instances. The auction-based model allows the cloud providers to sell their VM instances at a price determined according to the real market demand. Note that in an untruthful auction, users may declare bids lower than their actual valuations which may hurt other users and indirectly lead to profit losses for the cloud provider. Thus, unless strategy-proofness is enforced, maximizing the revenue may not be effective. In the pricing phase, the cloud provider dynamically determines the price that users should pay for their requests.

We design an optimal and an approximation mechanism that motivates cloud users to reveal their requests truthfully. Our proposed mechanisms take the strategic behavior of individual users into account and simultaneously maximize the global performance objective of the system. In addition, both mechanisms place VMs in as few PMs as possible. Such approach has been recognized as an efficient way of reducing cost [5]. This is also in alignment with green cloud computing objectives [6], where the cloud provider determines which PMs to power on/off in order to save energy. The mechanisms allow a cloud provider to choose PMs configurations that are aligned with its power consumption policies. In addition, our proposed approximation mechanism iteratively provisions VMs on each PM. This iterative mechanism allows the cloud provider to power on/off PMs based on the user demands.

1.1 Our Contribution

We address the problem of cloud resource management in the presence of multiple PMs with multiple types of resources. We design a strategy-proof greedy mechanism, called G-PMRM. G-PMRM not only provisions and allocates resources, but also dynamically determines the price that users should pay for their requests. In order to guarantee strategy-proofness of G-PMRM, we design the winner determination algorithm, such that it determines loser-independent allocations on each PM. This property makes the G-PMRM mechanism robust against strategic users who try to manipulate the system by changing the allocations of other users. We prove that G-PMRM mechanism is a polynomial time 3-approximation mechanism. We also design an optimal strategy-proof mechanism, VCG-PMRM, that we use as a benchmark when we investigate the performance of the G-PMRM mechanism. We perform extensive experiments in order to investigate the performance of the G-PMRM mechanism. The G-PMRM mechanism is fast and finds near optimal solutions, being very suitable for deployment in real cloud settings.

1.2 Related Work

Researchers approached the problem of VM placement in clouds considering different objectives and points of view. Dong et al. [7] proposed a method for VM placement considering multiple resource constraints using hierarchical clustering with best fit. Their goal is to improve resource utilization and reduce energy consumption by minimizing both the number of active physical servers and network elements. Ghribi et al. [8] proposed an allocation algorithm with a consolidation algorithm for VM placement in clouds in order to minimize overall energy consumption and migration cost. Maurer et al. [9] proposed a dynamic resource configuration to achieve high resource utilization and low service level agreement violation rates using knowledge management: case-based reasoning and a rule-based approach. Kesavan et al. [10] proposed a set of low-overhead management methods for managing the cloud infrastructure capacity to achieve a scalable capacity allocation for thousands of machines. Hu et al. [11] studied two time-cost optimization problems for provisioning resources and scheduling divisible loads with reserved instances in clouds. They formulated the problems as mixed integer programs. Tsai et al. [12] proposed a hyper-heuristic scheduling algorithm with the aim of reducing the makespan of task scheduling in clouds. Their approach uses two detection operators to determine when to change the low-level heuristic algorithm and a perturbation operator. Doyle et al. [13] proposed an algorithm to determine which data center requests should be routed, based on the relative priorities of the cloud operator. Such routing will reduce the latency, carbon emissions, and operational cost. Srikantaiah et al. [14] modeled the mapping of VMs to PMs as a multidimensional bin packing problem in

which PMs are represented by bins, and each resource is considered as a dimension of the bin. They studied energy consumption and resource utilization and proposed a heuristic algorithm based on the minimization of the sum of the Euclidean distances of the current allocations to the optimal point at each PM. Rodriguez and Buyya [15] proposed a meta-heuristic algorithm based on Particle Swarm Optimization for VM provisioning and scheduling strategies on IaaS that minimizes the overall workflow execution cost while meeting deadline constraints. Their approach considers dynamic provisioning, heterogeneity of unlimited computing resources, and VM performance variation. However, none of the above works proposed strategy-proof mechanisms for resource management in clouds.

Many researchers focused on reducing the operational costs of cloud providers through reducing energy consumption. Mastroianni et al. [5] proposed an approach for the consolidation of VMs on two resources, CPU and RAM, so that both resources are exploited efficiently. Their goal is to consolidate the VMs on as few PMs as possible and switch the other PMs off in order to minimize power consumption and carbon emissions, while ensuring a good level of QoS. In their proposed approach, decisions on the assignment and migration of VMs are driven by probabilistic processes and are based exclusively on local information. Mazzucco et al. [16] proposed policies based on dynamic estimates of users demand and models of system behavior in order to determine the minimum number of PMs that should be switched on to satisfy the demand with two objectives, reducing energy consumption and maximizing revenues. Polverini et al. [17] studied the problem of scheduling batch jobs on multiple geographically-distributed data centers. They considered the benefit of electricity price variations across time and locations. Their proposed algorithm schedules jobs when electricity prices are sufficiently low and to places where the energy cost per unit work is low. Mashayekhy et al. [18] proposed energy-aware scheduling algorithms for detailed task placement of MapReduce jobs. Their scheduling algorithms account for significant energy efficiency differences of different machines in a data center. Khosravi et al. [19] proposed a VM placement algorithm that increases the environmental sustainability by taking into account distributed data centers with different carbon footprint rates. Beloglazov et al. [20] investigated the challenges and architectural principles for energy-efficient management of cloud computing. They proposed energy-aware allocation heuristics that provision resources to improve energy efficiency of the data center considering QoS requirements. For a survey on energy-efficient cloud computing systems, and their taxonomy, the reader is referred to [21]. For a survey of green computing performance metrics for data centers, such as power metrics, thermal metrics, and extended performance metrics, the reader is referred to [22]. However, in this study we focus on cloud auction markets which necessitate designing game theory based mechanisms to reach market equilibria.

Motivated by the recent introduction of cloud auctions by Amazon, resource allocation and pricing in clouds have been increasingly considered by several researchers. Yi et al. [23] proposed a resource provisioning approach to reduce the monetary cost of computation using Amazon spot instances. Their results show that by using an appropriate checkpointing scheme, the cost and task completion time can be reduced. Prasad et al. [24] proposed a cloud resource procurement and dynamic pricing approach in a reverse auction setting with several cloud providers. They proposed several strategy-proof mechanisms for resource procurement and pricing. Fu et al. [1] proposed a core-based pricing method using a coalitional game. However, they did not consider strategy-proofness. Iyer and Veeravalli [25] studied the problem of resource allocation and pricing strategies in cloud computing. They considered the Nash Bargaining Solution and Raiffa Bargaining Solution, and proposed optimal solutions for allocating virtual CPU instances for both independent tasks and workflow tasks. Kang and Wang [26] proposed an auction-based cloud resource allocation algorithm that considers the fitness between resources and services. Mihailescu and Teo [27] proposed a reverse auction-based mechanism for dynamic pricing of resources. A revenue sharing mechanism for multiple cloud providers using cooperative games was proposed by Niyato et al. [28]. Teng and Magoules [29] employed game theoretical techniques to solve the multi-user equilibrium allocation problem, and proposed a resource pricing and allocation policy where users can predict the future resource price. In our previous studies, we proposed truthful mechanisms for VM allocation in clouds without considering their placement onto PMs [30], [31].

Our work is different from all the previous works, since we address the cloud resource management problem through an economic model by performing VM provisioning, placement, pricing, and considering possible energy savings. Moreover, we consider a realistic cloud setting with multiple heterogeneous PMs providing multiple types of resources, and users requesting different types of VM instances. We also provide worst case performance guarantee for our proposed mechanism.

1.3 Organization

The rest of the paper is organized as follows. In Section 2, we describe the PM resource management problem in clouds. In Section 3, we present our proposed optimal mechanism. In Section 4, we describe our proposed approximation mechanism, and in Section 5, we characterize its properties. In Section 6, we evaluate the performance of the mechanisms by extensive experiments. In Section 7, we summarize our results and present possible directions for future research.

TABLE 1: General purpose (M3) VM instance types offered by Amazon EC2.

	m3.medium $m=1$	m3.large $m=2$	m3.xlarge $m=3$	m3.2xlarge $m=4$
CPU	1	2	4	8
Memory (GB)	3.75	7.5	15	30
Storage (GB)	4	32	80	160

2 Physical Machine Resource Management Problem

In this section, we present the system model and the problem of Physical Machine Resource Management (PMRM) in clouds.

We consider a cloud provider managing a public cloud consisting of P PMs, $\mathcal{PM} = \{1, \ldots, P\}$, offering a set $\mathcal{R} = \{1, \ldots, R\}$ of R types of resources such as cores, memory, and storage, to users in the form of VM instances. The information about cloud's physical resources is not known to the users. A PM can be a server, a rack of servers, or a group of racks. A key characteristic of our model is that it enables cloud providers to define PMs based on their resource configurations and user demands. This allows the cloud provider to treat the heterogeneous resources in a flexible way. Each PM p has restricted capacity, C_{pr}, for a resource $r \in \mathcal{R}$ available for allocation. We denote by $\mathcal{C}_p$ the vector of available capacities on each PM p. The cloud provider offers its heterogeneous resources to users in the form of M types of VMs. The set of VM types is denoted by $\mathcal{VM}$. Each VM of type $m \in \mathcal{VM}$ consists of a specific amount of each type of resource $r \in \mathcal{R}$. In addition, w_{mr} represents the amount of resources of type r that one VM instance of type m provides. Table 1 presents the four types of general purpose (M3) VM instances offered by Amazon EC2. By considering CPU, memory, and storage as type 1, type 2, and type 3 resources, respectively, for example, the m3.medium instance ($m = 1$) is characterized by: $w_{11} = 1$, $w_{12} = 3.75$ GB, and $w_{13} = 4$ GB.

We consider a set $\mathcal{N}$ of N users requesting a set of VM instances. Each user has a private valuation for obtaining her request consisting of VM instances. Bidding is the process of expressing user's valuation for a heterogeneous set of VMs and communicating it to the system. In general, it does not matter how the valuation is being encoded, as long as the system can understand the bid submitted by the user (bidder). Users use a bidding language to express their requests. We define a bidding language that can be used to express a user's request (which may or may not be their true request) and to report it to the system. Each user i, $i \in \mathcal{N}$, can submit a pair (ρ_i, b_i), where ρ_i is her requested bundle of VMs and b_i is the price that she is willing to pay for ρ_i. As a result, her valuation is defined as $v_i(\hat{\rho}_i) = b_i$ if $\rho_i \subseteq \hat{\rho}_i$ and $v_i(\hat{\rho}_i) = 0$, otherwise. Such a bid $\beta_i = (\rho_i, b_i)$ is called an atomic bid. Users with atomic bids are called single-minded bidders. User i's requested bundle is represented by $\rho_i = \langle k_{i1}, k_{i2}, \ldots, k_{iM} \rangle$, where k_{im} is the number of requested VM instances of type $m \in \mathcal{VM}$. It is worth noting that ρ_i can consist of one type of VM, while the request for the remaining types of VMs are zero. For example, request $(\langle 10, 0, 0, 5 \rangle, \$25)$ represents a user requesting 10 m3.medium VM instances, 0 m3.large VM instance, 0 m3.xlarge VM instance, and 5 m3.2xlarge VM instances; and her bid is \$25. Table 2 summarizes the notation used throughout the paper.

TABLE 2: Notation

$\mathcal{PM}$	Set of physical machines $\{1, \ldots, P\}$
$\mathcal{VM}$	Set of virtual machine types $\{1, \ldots, M\}$
$\mathcal{R}$	Set of resources $\{1, \ldots, R\}$
w_{mr}	Amount of resources of type $r \in \mathcal{R}$ provided by a VM instance of type $m \in \mathcal{VM}$
C_{pr}	Capacity of $p \in \mathcal{PM}$ for a resource of type $r \in \mathcal{R}$
$\mathcal{N}$	Set of users $\{1, \ldots, N\}$
ρ_i	Bundle requested by user $i \in \mathcal{N}$
k_{im}	Number of requested VM instances of type $m \in \mathcal{VM}$ by user $i \in \mathcal{N}$
b_i	Bid of user $i \in \mathcal{N}$
v_i	Valuation function of user $i \in \mathcal{N}$
u_i	Utility function of user $i \in \mathcal{N}$
Π_i	Payment of user $i \in \mathcal{N}$

Given the above setting the problem of *Physical Machine Resource Management (PMRM)* in clouds is to determine the allocation of VM to PM simultaneously with the allocation of VM to users and the prices for the VM bundles such that the sum of users' valuations is maximized. A mechanism for solving the PMRM problem consists of three phases: winner determination, provisioning and allocation, and pricing. In the winner determination phase, the cloud provider determines which users receive their requested bundles. Based on the results of the winner determination phase, the cloud provider provisions the amount of resources in the form of VM instances onto the PMs, and then allocates the requested bundles of VMs to the winning users. Then, the cloud provider determines the unique amount that each winning user must pay based on the winner determination results. Note that the payment of a user is not greater than its submitted bid. The main building blocks of a PMRM mechanism include: a winner determination function $\mathcal{W}$ and a payment function Π.

Fig. 1 shows a high-level view of PMRM. For simplicity, we consider that only one type of resource is available. Four users submit their bids to the cloud provider, where two PMs are available to fulfill the users' requests. As an example, user 1 requests two VM_1 and one VM_2 as her bundle, and she submits a bid of \$0.50. The mechanism employed by the cloud provider collects the bids and then selects the users whose bundle would be provisioned. After it provisions the VMs on the PMs based on the selected users, it allocates the bundles to those users. The selected users pay the amount determined by the mechanism to the cloud provider.

User i has a *quasi-linear utility function* defined as the difference between her valuation and payment, $u_i = v_i(\mathcal{W}_i) - \Pi_i$, where $\mathcal{W}_i$ is the allocated bundle to user i, and Π_i is the payment for user i. The users are self-interested, that is, they want to maximize their own

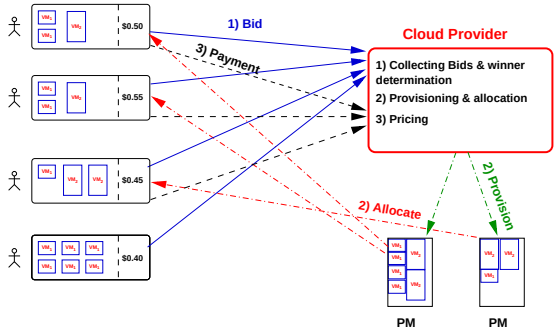


Fig. 1: A high-level view of PMRM.

utility. It may be beneficial for cloud users to manipulate the auction outcomes and gain unfair advantages via untruthfully revealing their requests. Since the request of a user is a pair of bundle and value, the user can declare a higher value in the hope to increase the likelihood of obtaining her requested bundle, or declare a different VM bundle from her actual request. Strategic behaviors of such users may hinder other qualified users, leading to reduced revenue and reputation of the cloud provider. Our goal is to design strategy-proof mechanisms that solve the PMRM problem and discourage users from gaming the system by untruthful reporting. The mechanism maximizes social welfare, the sum of users' valuations for the requested bundles of VMs.

3 Optimal Mechanism for PMRM

In this section, we propose an optimal strategy-proof mechanism for PMRM. For a detailed introduction on mechanism design the reader is referred to [32].

Cloud users may submit different requests from their true requests hoping to gain more utility. We denote by $\hat{\beta}_i = (\hat{\rho}_i, \hat{b}_i)$ user i's declared request. Note that $\beta_i = (\rho_i, b_i)$ is user i's true request. We denote by $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_N)$ the vector of requests of all users, and by $\boldsymbol{\beta}_{-i}$ the vector of all requests except user i's request (i.e., $\boldsymbol{\beta}_{-i} = (\beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_N)$). Users are rational in a sense that they do not want to pay more than their valuation for their requested bundles. A well-designed mechanism should give incentives to users to participate. Such a property of a mechanism is called individual rationality and is defined as follows:

Definition 1 (Individual rationality): A mechanism is *individually-rational* if for every user i with true request β_i and the set of other requests, we have $u_i(\beta_i) \geq 0$.

In other words, a mechanism is individually-rational if a user can always achieve as much utility from participation as without participation. However, such mechanisms do not always give incentives to users to report their requests truthfully. Our goal is to design a mechanism that is strategy-proof, i.e., a mechanism that gives incentives to users to reveal their true requests.

Definition 2 (Strategy-proofness): A mechanism is *strategy-proof* (or incentive compatible) if $\forall i \in \mathcal{N}$ with a true request declaration β_i and any other declaration $\hat{\beta}_i$, and $\forall \hat{\boldsymbol{\beta}}_{-i}$, we have that $u_i(\beta_i, \hat{\boldsymbol{\beta}}_{-i}) \geq u_i(\hat{\beta}_i, \hat{\boldsymbol{\beta}}_{-i})$.

The strategy-proofness property implies that truthful reporting is a dominant strategy for the users. As a result, it never pays off for any user to deviate from reporting her true request, irrespective of what the other users report as their requests.

Our first proposed strategy-proof mechanism is an optimal one and it is based on the Vickrey-Clarke-Groves (VCG) mechanism. An optimal winner determination function with VCG payments provides a strategy-proof mechanism [33], [34], [35]. We define our proposed optimal VCG-based mechanism for PMRM as follows:

Definition 3 (VCG-PMRM mechanism): The VCG-PMRM mechanism consists of winner determination function $\mathcal{W}$, and payment function Π, where

i) $\mathcal{W}$ is an optimal winner determination function maximizing the social welfare, and

ii) $\Pi_i(\hat{\boldsymbol{\beta}}) = \sum_{j \in \mathcal{N} \setminus \{i\}} v_j(\mathcal{W}_j(\hat{\boldsymbol{\beta}}_{-i})) - \sum_{j \in \mathcal{N} \setminus \{i\}} v_j(\mathcal{W}_j(\hat{\boldsymbol{\beta}}))$,

such that $\sum_{j \in \mathcal{N} \setminus \{i\}} v_j(\mathcal{W}_j(\hat{\boldsymbol{\beta}}_{-i}))$ is the optimal social welfare obtained when user i is excluded from participation, and $\sum_{j \in \mathcal{N} \setminus \{i\}} v_j(\mathcal{W}_j(\hat{\boldsymbol{\beta}}))$ is the sum of all users valuations in the optimal solution except user i's value.

The problem that needs to be solved in the winner determination phase of PMRM can be formulated as an integer program (called IP-PMRM), as follows:

$$\text{Maximize } V = \sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{PM}} b_i \cdot X_{ip} \tag{1}$$

Subject to:

$$\sum_{p \in \mathcal{PM}} X_{ip} \leq 1, \forall i \in \mathcal{N} \tag{2}$$

$$\sum_{i \in \mathcal{N}} \sum_{m \in \mathcal{VM}} k_{im} w_{mr} X_{ip} \leq C_{pr}, \quad \forall p \in \mathcal{PM}, \forall r \in \mathcal{R} \tag{3}$$

$$X_{ip} = \{0, 1\} \tag{4}$$

The decision variables X_{ip} are defined as follows: $X_{ip} = 1$, if ρ_i is allocated to i on machine p; and 0 otherwise. The objective function is to maximize social welfare V. Constraints (2) ensure that the request of each user is fulfilled at most once. Constraints (3) guarantee that the allocation of each resource type does not exceed the available capacity of that resource for any PM. Constraints (4) represent the integrality requirements for the decision variables.

The winner determination phase of VCG-PMRM (implementing $\mathcal{W}$) consists of solving the IP-PMRM. The execution time of VCG-PMRM becomes prohibitive for large instances of the PMRM problem. As a result, we resort to designing a fast mechanism providing an approximate solution for the PMRM problem. The VCG-PMRM mechanism will be used in our experiments as a benchmark for the performance of the proposed approximation mechanism.

4 A Strategy-proof Approximation Mechanism for PMRM

In this section, we introduce our proposed strategy-proof greedy mechanism, G-PMRM. Greedy algorithms to solve PMRM do not necessarily satisfy the strategy-proofness property. To obtain a strategy-proof mechanism, the winner determination function $\mathcal{W}$ must be monotone, and the payment function Π must be based on the critical payment [36]. In addition, we design an iterative winner determination algorithm in the sense that, in each iteration, it determines the assignment of winning requests to their associated PM. This way the mechanism utilizes PMs one by one until all winning requests are assigned. This approach allows the cloud provider to power off unutilized PMs to save energy. In the following, we define the properties that our proposed mechanism needs to satisfy in order to guarantee strategy-proofness.

Definition 4 (Monotonicity): A winner determination function $\mathcal{W}$ is *monotone* if it selects user i with $\hat{\beta}_i$ as her declared request, then it also selects user i with a more preferred request $\hat{\beta}'_i$, i.e., $\hat{\beta}'_i \succeq \hat{\beta}_i$.

That means, any winning user who receives her requested bundle by declaring a request $\hat{\beta}_i$ will still be a winner if she requests a more preferred request (i.e., smaller bundle and/or a higher bid). Formally, $\hat{\beta}'_i \succeq \hat{\beta}_i$ if $\hat{b}'_i \geq \hat{b}_i$ and $\hat{\rho}_i =< \hat{k}_{i1}, \hat{k}_{i2}, \ldots, \hat{k}_{iM} >$, $\hat{\rho}'_i =< \hat{k}'_{i1}, \hat{k}'_{i2}, \ldots, \hat{k}'_{iM} >$ such that $\sum_{m \in \mathcal{VM}} \hat{k}'_{im} w_{mr} \leq \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr}, \forall r \in \mathcal{R}$.

Definition 5 (Critical payment): Let $\mathcal{W}$ be a monotone winner determination function, then for every β_i, there exists a unique value v_i^c, called *critical payment*, such that $\forall \hat{\beta}_i \geq (\rho_i, v_i^c)$, $\hat{\beta}_i$ is a winning declaration, and $\forall \hat{\beta}_i < (\rho_i, v_i^c)$ is a losing declaration. $\Pi_i(\hat{\boldsymbol{\beta}}) = v_i^c$ if user i wins, and $\Pi_i(\hat{\boldsymbol{\beta}}) = 0$, otherwise.

However, a key challenge in the design of our greedy mechanism in order to satisfy monotonicity is the presence of multiple PMs with multiple types of resources. Lucier and Borodin [37] and Chekuri and Gamzu [38] showed that loser-independent algorithms can be employed as sub-procedures in a greedy iterative approach to obtain monotonicity for the overall winner determination algorithm.

Definition 6 (Loser-independence): An algorithm $\mathcal{W}$ is *loser-independent* with respect to user i's request $\hat{\beta}_i$ in which $\mathcal{W}_i(\hat{\beta}_i, \hat{\boldsymbol{\beta}}_{-i}) = \emptyset$, if user i declares a request $\hat{\beta}'_i$, then either $\mathcal{W}_j(\hat{\beta}'_i, \hat{\boldsymbol{\beta}}_{-i}) = \mathcal{W}_j(\hat{\beta}_i, \hat{\boldsymbol{\beta}}_{-i})$, for all $j \neq i$, or $\mathcal{W}_i(\hat{\beta}_i, \hat{\boldsymbol{\beta}}_{-i}) \neq \emptyset$, where $\hat{\beta}'_i \succeq \hat{\beta}_i$.

In other words, if user i was not selected by algorithm $\mathcal{W}$ when she declared request $\hat{\beta}_i$ and now she declares a new request $\hat{\beta}'_i$ while the declared requests of the rest of the users do not change, then the outcome of algorithm $\mathcal{W}$ changes only if user i becomes a winner by declaring a better request $\hat{\beta}'_i$.

If the bid of a not-selected user changes but her allocation stays the same then the allocations to all other users do not change. A key property of loser-independent algorithms is that if a user is not a winner, it guarantees the same output no matter her declaration. This property makes the algorithm robust against strategic users who try to manipulate the system by changing other users allocations. Note that if such a user tries to change the allocation determined by the algorithm, she should declare a request that will make her a winner.

Algorithm 1 G-PMRM Mechanism

1: {Collect user requests.}
2: **for all** $i \in \mathcal{N}$ **do**
3: Collect user request $\hat{\beta}_i = (\hat{\rho}_i, \hat{b}_i)$ from user i
4: {Allocation.}
5: (V, $\mathbf{x}$) = G-PMRM-WIN($\hat{\boldsymbol{\beta}}$)
6: Provisions and allocates VM instances according to $\mathbf{x}$.
7: {Payment.}
8: Π =G-PMRM-PAY($\hat{\boldsymbol{\beta}}, \mathbf{x}$)

Algorithm 2 IS-FEASIBLE($\rho_i, \mathcal{C}_p$)

1: **for all** $r \in \mathcal{R}$ **do**
2: $\sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$
3: $flag \leftarrow$ TRUE
4: **for all** $r \in \mathcal{R}$ **do**
5: **if** $\sigma_{ir} > C_{pr}$ **then**
6: $flag \leftarrow$ FALSE
7: **break;**
8: **Output:** $flag$

Obtaining strategy-proofness requires the design of a loser-independent winner determination algorithm that allocates the resources of each PM individually. If the winner determination algorithm is loser-independent for each PM, then when the solutions for each individual machines are combined in an iterative fashion it will lead to a monotone overall winner determination algorithm. Having a monotone winner determination algorithm along with a critical value payment, makes the mechanism strategy-proof.

We define our proposed G-PMRM mechanism as follows:

Definition 7 (G-PMRM mechanism): The G-PMRM mechanism consists of the winner determination algorithm G-PMRM-WIN and the payment function G-PMRM-PAY.

The G-PMRM mechanism is given in Algorithm 1. The mechanism is run periodically by the cloud provider. It collects the requests from all users, and then it determines the winning users by calling the G-PMRM-WIN algorithm. Once the users are selected the mechanism provisions the required number and types of VM instances on the selected PMs, and then it determines the payments by calling the G-PMRM-PAY function. The users are then charged the payment determined by the mechanism. G-PMRM-WIN and G-PMRM-PAY are presented in the following.

Before describing the winner determination algorithm, we need to define a function, called IS-FEASIBLE(), that we call in our proposed winner determination algorithm. IS-FEASIBLE() is given in Algorithm 2. It checks the feasibility of allocating the requested bundle of VMs of a user on a specific PM, that is, it checks whether PM p

Algorithm 3 G-PMRM-WIN($\hat{\boldsymbol{\beta}}$)

1: $\mathcal{U} = \emptyset$
2: **for all** $i \in \mathcal{N}$ **do**
3: **for all** $r \in \mathcal{R}$ **do**
4: $\sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$
5: $rp_i = \sum_{m \in \mathcal{VM}} k_{im} o_m$
6: **if** $\hat{b}_i \geq rp_i$ **then**
7: $\mathcal{U} = \mathcal{U} \cup \{i\}$
8: **for all** $p \in \mathcal{PM}$ **do**
9: {*First phase*}
10: $\mathcal{C}_p = (C_{p1}, \ldots, C_{pR})$
11: $\hat{\mathcal{U}} \leftarrow \emptyset$; $\hat{\mathbf{x}} = \mathbf{0}$
12: **for all** $i \in \mathcal{U}$ **do**
13: **if** IS-FEASIBLE($\rho_i, \mathcal{C}_p$) **then**
14: $\hat{\mathcal{U}} \leftarrow \hat{\mathcal{U}} \cup \{i\}$
15: $\hat{V} = \max_{i \in \hat{\mathcal{U}}} \hat{b}_i$
16: $j = \text{argmax}_{i \in \hat{\mathcal{U}}} \hat{b}_i$
17: $\hat{x}_j = 1$
18: {*Second phase*}
19: $\tilde{\mathcal{U}} \leftarrow \emptyset$; $\tilde{\mathbf{x}} = \mathbf{0}$
20: **for all** $i \in \mathcal{U}$ **do**
21: **if** IS-FEASIBLE($\rho_i, \mathcal{C}_p/2$) **then**
22: $\tilde{\mathcal{U}} \leftarrow \tilde{\mathcal{U}} \cup \{i\}$
23: **for all** $i \in \tilde{\mathcal{U}}$ **do**
24: $d_i = \hat{b}_i / \sqrt{\sum_{r=1}^{R} \frac{\sigma_{ir}}{C_{pr}}}$
25: Sort $\tilde{\mathcal{U}}$ in decreasing order of d_i
26: $\bar{\mathcal{U}} \leftarrow \emptyset$; $\bar{\mathcal{C}}_p = 0$; $flag \leftarrow$ TRUE
27: **while** $\tilde{\mathcal{U}} \neq \emptyset$ and $flag$ **do**
28: **for all** $r \in \mathcal{R}$ **do**
29: **if** $\bar{C}_{pr} > C_{pr}/2$ **then**
30: $flag \leftarrow$ FALSE
31: **if** $flag$ **then**
32: $i \leftarrow \text{argmax}_{i \in \tilde{\mathcal{U}}} d_i$
33: $\tilde{\mathcal{U}} = \tilde{\mathcal{U}} \setminus \{i\}$
34: $\tilde{x}_i = 1$
35: $\bar{\mathcal{U}} \leftarrow \bar{\mathcal{U}} \cup \{i\}$
36: **for all** $r \in \mathcal{R}$ **do**
37: $\bar{C}_{pr} = \bar{C}_{pr} + \sigma_{ir}$
38: $\tilde{V} = 0$; $\bar{\mathcal{C}}_p = 0$
39: **if** $\bar{\mathcal{U}} \neq \emptyset$ **then**
40: **for all** $i \in \bar{\mathcal{U}}$ except the last user j added to $\bar{\mathcal{U}}$ **do**
41: $\tilde{V} = \tilde{V} + \hat{b}_i$
42: **for all** $r \in \mathcal{R}$ **do**
43: $\bar{C}_{pr} = \bar{C}_{pr} + \sigma_{ir}$
44: **for all** $r \in \mathcal{R}$ **do**
45: $\bar{\sigma}_{jr} = C_{pr}/2 - \bar{C}_{pr}$
46: $flag \leftarrow$ TRUE
47: **for all** $r \in \mathcal{R}$ **do**
48: **if** $\sigma_{jr} > \bar{\sigma}_{jr}$ **then**
49: $flag \leftarrow$ FALSE
50: **if** $flag$ **then**
51: **for all** $r \in \mathcal{R}$ **do**
52: $\bar{\sigma}_{jr} = \sigma_{jr}$
53: $\bar{b}_j = d_j \sqrt{\sum_{r=1}^{R} \frac{\bar{\sigma}_{jr}}{C_{pr}}}$
54: $\tilde{V} = \tilde{V} + \bar{b}_j$
55: {*Third phase*}
56: **if** $\hat{V} \geq \tilde{V}$ **then**
57: $V_p = \hat{V}$; $\mathbf{x}_p = \hat{\mathbf{x}}$
58: **else**
59: $V_p = \tilde{V}$; $\mathbf{x}_p = \tilde{\mathbf{x}}$
60: Update $\mathcal{U}$ to the unallocated users based on $\mathbf{x}_p$
61: $V = \sum_{p \in \mathcal{PM}} V_p$; $\mathbf{x} = \sum_{p \in \mathcal{PM}} \mathbf{x}_p$
62: **Output:** V, $\mathbf{x}$

has enough resources to fulfill a requested bundle of VMs. For user i with $\rho_i = \langle k_{i1}, k_{i2}, \ldots, k_{iM} \rangle$, and PM p, the function computes σ_{ir}, the amount of resources of type r requested by user i, and then checks it against the available resource capacity C_{pr} for all types $r \in R$ of resources on PM p.

G-PMRM-WIN is given in Algorithm 3. G-PMRM-WIN has one input parameter, the vector of users declared requests $\hat{\boldsymbol{\beta}}$, and two output parameters: V, the total social welfare, and $\mathbf{x}$, the set of winning users.

The algorithm finds the total amount of each resource type requested by each user in $\mathcal{N}$ (lines 2-4). It also calculates the reserve price for each request (line 5). We consider reservation prices for the VMs to avoid non-profitable trade. The reserve price is often a reflection of the VM cost. The cloud provider sets a reserve price o_m for each type of VM $m \in \mathcal{VM}$. These prices are denoted by a vector $\mathcal{O} =< o_1, \ldots, o_M >$. The reserve price (bundle-specific) of a user is calculated based on her requested bundle as follows: $rp_i = \sum_{m \in \mathcal{VM}} k_{im} o_m$, which is the weighted sum of reserve prices for all the requested VMs in the bundle of user i. Users whose bids are above the reserve prices are included in $\mathcal{U}$, the set of users who are not yet selected to receive their requested bundles (lines 6-7). In the following, we call $\mathcal{U}$, the set of not-selected users. Then, the algorithm iterates over all PMs to find a subset of users whose requests should be assigned to each PM p, where the vector of resource capacities of PM p is $\mathcal{C}_p = (C_{p1}, \ldots, C_{pR})$ (lines 8-60). Each iteration of the algorithm (lines 8-60) consists of three phases. In the first phase, the algorithm finds the user with the maximum bid (lines 9-17). In the second phase, the algorithm finds the set of users based on their bid densities whose overall requested amount of resources is at least half of the capacities of the PM (lines 18-54). In the third phase, the algorithm finds the maximum social welfare V_p between the obtained social welfare of the first and second phase (lines 55-60). Based on the phase providing the maximum social welfare, the algorithm chooses the set of winning users $\mathbf{x}_p$ for PM p. The requested bundle of VMs of these users will be provisioned using the resources of PM p. Then, the algorithm updates the set of not-selected users $\mathcal{U}$ (line 60). After finding the set of winning users to be allocated to each PM $p, \forall p \in \mathcal{PM}$, the algorithm calculates the obtained social welfare V and the final set of winning users specified by vector $\mathbf{x}$ (line 61).

In the following, we discuss each phase in detail. In phase one, the algorithm first finds the set of users $\hat{\mathcal{U}}$ whose requests can be fulfilled by PM p (lines 9-14) by calling the IS-FEASIBLE() function that checks the feasibility of allocating the requested bundle of VMs of each user i on PM p. Then, it finds the maximum bid among the users in $\hat{\mathcal{U}}$ (line 15). It also finds the user associated with it as a winning user and updates $\hat{\mathbf{x}}$ (lines 16-17).

In the second phase, the algorithm finds the set of users $\tilde{\mathcal{U}}$, where each user's request is not greater than half of the available capacity of the PM p, for each resource by calling IS-FEASIBLE (lines 18-22). Then, the algorithm calculates the bid densities of users in $\tilde{\mathcal{U}}$ (lines 23-24) according to a *density* metric defined as

$d_i = \frac{\hat{b}_i}{\sqrt{\sum_{r=1}^{R} \frac{\sigma_{ir}}{C_{pr}}}}, \forall i \in \mathcal{U}$, where $\sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$ is the amount of each resource of type r requested by user i, and $\frac{1}{C_{pr}}$ is the *relevance factor* characterizing the scarcity of resources of type r. Next, the algorithm sorts the users in $\tilde{\mathcal{U}}$ based on the decreasing order of their bid densities (line 25). Then, the algorithm selects users based on their bid densities (lines 26-37). To do that it checks whether the total amount of requests of the current selected users are less than half of the available capacity of each resource in the PM (lines 28-30). If the total amount of requests do not reach the half, the algorithm selects a new user according to the bid densities, and updates the set $\bar{\mathcal{U}}$, $\bar{\mathbf{x}}$, and the total amount of requests assigned to PM p (lines 31-37). Then, G-PMRM-WIN finds the social welfare of the selected users in the second phase if $\bar{\mathcal{U}}$ is not an empty set (lines 38-54). It first, finds the social welfare $\tilde{V}$ of all the selected users except the last user (i.e., user j) added to the set $\bar{\mathcal{U}}$ (lines 40-43). Then, the algorithm finds the remaining capacities of each resource $\bar{\sigma}_{jr}$ from half of the capacities of the PM (lines 44-45). It also checks if the actual request of user j is less than the remaining capacities (lines 46-52). Next, the algorithm calculates the value $\bar{b}_j$ (line 53) based on either the remaining capacities (lines 44-45) or the actual user j's request if her request is less than the remaining capacities (lines 51-52). Finally, the algorithm updates the social welfare $\tilde{V}$ by adding $\bar{b}_j$ (line 54).

In phase three, the algorithm selects the maximum social welfare and the associated selected users from the two phases as the social welfare and the set of winning users whose requests will be provisioned on PM p (lines 55-60). The obtained social welfare on PM p is V_p, the maximum social welfare between the social welfare of the two phases. The set of winning users whose requests will be provisioned on PM p, $\mathbf{x}_p$, is the solution that gives V_p. Then, the algorithm updates the set of not-selected users $\mathcal{U}$ based on $\mathbf{x}_p$ to guarantee that each user is selected at most once (line 60). After finding the set of users to be allocated to each PM, the algorithm calculates the obtained social welfare V and the final set of winning users specified by vector $\mathbf{x}$ (line 61).

The G-PMRM-PAY function is given in Algorithm 4. The G-PMRM-PAY function has two input parameters, the vector of users declared requests ($\hat{\boldsymbol{\beta}}$), and the set of winning users given by $\mathbf{x}$. The payment of winning user i is v_i^c, where v_i^c is the critical payment of user i, if i wins and zero if i loses. Finding the critical payment is done by a binary search over values less than the declared value and above the reserve price. We consider reserve prices for the VMs in order to avoid non-profitable trade.

G-PMRM-PAY initializes the payment of each user to zero (line 4). Then for each winning user i (i.e., $x_i = 1$) it initializes the lower and upper bounds of the payment to the reserve price and the declared value of the user, respectively (lines 6-9). The reserve price of a user is calculated based on her requested bundle as follows: $l = \sum_{m \in \mathcal{VM}} k_{im} o_m$, which is the weighted sum of reserve prices for all the requested VMs in the bundle of user i.

Algorithm 4 G-PMRM-PAY: Critical Payment Function

1: **Input:** $\hat{\boldsymbol{\beta}} = (\hat{\beta}_1, \ldots, \hat{\beta}_N)$; vector of requests (bundle, bid)
2: **Input:** $\mathbf{x}$; winning users
3: **for all** $i \in \mathcal{U}$ **do**
4: $\quad \Pi_i = 0$
5: $\quad$ **if** x_i **then**
6: $\quad\quad l = 0$
7: $\quad\quad$ **for all** $m \in \mathcal{VM}$ **do**
8: $\quad\quad\quad l = l + k_{im} o_m$
9: $\quad\quad h = \hat{b}_i$
10: $\quad\quad$ **while** $(h - l) \geq 1$ **do**
11: $\quad\quad\quad v_i^c = (h + l)/2$
12: $\quad\quad\quad \hat{\beta}_i^c = (\hat{\rho}_i, v_i^c)$
13: $\quad\quad\quad (V', \mathbf{x}')$ = G-PMRM-WIN $((\hat{\beta}_1, \ldots, \hat{\beta}_i^c, \ldots, \hat{\beta}_N))$
14: $\quad\quad\quad$ **if** x_i' **then**
15: $\quad\quad\quad\quad h = v_i^c$ {user i is winning by declaring v_i^c}
16: $\quad\quad\quad$ **else**
17: $\quad\quad\quad\quad l = v_i^c$
18: $\quad\quad \Pi_i = h$
19: **Output:** $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_N)$

G-PMRM-PAY sets the critical payment, v_i^c, as the average of the lower and upper bounds, and checks if user i would have won if she have had declared her bid as v_i^c (lines 11-13). If user i would have won by declaring v_i^c, G-PMRM-PAY sets the value as a new upper bound; otherwise, the value is a new lower bound (lines 14-17). G-PMRM-PAY tries to reduce the gap between upper and lower bound to 1. Then, the payment of user i, Π_i is set to the upper bound value (line 18). G-PMRM-PAY returns an output parameter, Π, the payment vector for the users.

Example. A key idea in the design of G-PMRM-WIN, is finding a partial allocation in the second phase in order to not only guarantee approximation ratio but also strategy-proofness. It is not possible to guarantee the strategy-proofness of the mechanism, if we allow G-PMRM-WIN to find an allocation considering the full capacity in the second phase. According to the Definition 4 (Monotonicity), any winning user who receives her requested bundle by declaring a request $\hat{\beta}_i$ will still be a winner if she requests a more preferred request (i.e., smaller bundle and/or a higher bid). In the following, we provide an example showing that considering the full capacity in the second phase will not satisfy the monotonicity property, that is, showing that if a winning user submits a higher bid, she becomes a loser.

We consider six users, where their requests, bids, and bid densities are shown in Table 3. We consider a cloud provider with only one type of resource (storage) with two PMs each with capacity of 1024 GB. The first phase of the mechanism selects users I and II for PM_1, and users III and IV for PM_2, where the solution has a total value of \$102+\$102+\$150+\$50=\$404. The second phase of the modified mechanism (i.e., considering the full capacity) selects user V for PM_1, and user VI for PM_2, where the solution has a total value of \$198+\$198=\$396. As a result, the mechanism selects the result of the

TABLE 3: Example - Users requests.

User	requested storage	bid	bid density
I	512 GB	\$102	204
II	512 GB	\$102	204
III	768 GB	\$150	200
IV	256 GB	\$50	200
V	1024 GB	\$198	198
VI	1024 GB	\$198	198

first phase as the solution. Now, we consider that user IV submits a higher bid of \$52 instead of her actual value (\$50). Her bid density would change to 208, which is the highest density among all six users. Therefore, the first phase selects users IV and I for PM_1, and user II for PM_2, where the solution has a total value of \$102+\$102+\$52=\$256. The second phase (with full capacity) selects user V for PM_1, and user VI for PM_2, where the solution has a total value of \$198+\$198=\$396. As a result, the modified mechanism selects the results of the second phase as the solution. This solution does not include user IV anymore, and the winner determination is not monotone, and thus the mechanism is not strategy-proof. This example shows that a user can lose by declaring a higher value which should not be a case.

In the next section, we prove that our proposed mechanism is strategy-proof and that its worst-case performance is well-bounded.

5 Properties of G-PMRM

In this section, we investigate the properties of G-PMRM. We first show that the mechanism is individually rational (i.e., truthful users will never incur a loss).

Theorem 1: G-PMRM mechanism is individually rational.

Proof: We consider two cases. In case one, we consider a truthful user i who does not win. Such user is not incurring a loss since she pays 0 (line 4 of Algorithm 4), and her utility is 0. In case two, we consider user i as a winning user. We need to prove that if user i reports her true request then her utility is non-negative. In line 18 of Algorithm 4, the payment for user i is set to h, where h is initially set to $\hat{b}_i$ as an upper bound. The determined payment of user i is less than the initial value of h due to binary search. As a result, G-PMRM-PAY always computes a payment $\Pi_i \leq \hat{b}_i$. The utility of user i (i.e., $u_i = \hat{b}_i - \Pi_i \geq 0$) is non-negative if she report truthfully (i.e., $\hat{b}_i = b_i$), and she never incurs a loss. This proves the individual-rationality of G-PMRM. □

We now prove that the allocation on each PM (obtained in each iteration of the G-PMRM-WIN algorithm) is loser-independent.

Theorem 2: The allocation obtained by each iteration of G-PMRM-WIN is loser-independent.

Proof: To prove the loser-independency property of each iteration of G-PMRM-WIN, we need to analyze the results of two scenarios, a current declaration and a new declaration. In the current declaration, user i submits a request $\hat{\beta}_i$, and G-PMRM-WIN on PM p finds $\hat{V}$ and $\tilde{V}$ as the social welfare obtained by the first and the second phase, respectively. In the new declaration, user i submits a request $\hat{\beta}'_i \succeq \hat{\beta}_i$, and the rest of the users declare the same request as they declared in their current declarations. G-PMRM-WIN on PM p finds $\hat{V}'$ and $\tilde{V}'$ as the social welfare obtained in this scenario by the first and the second phase, respectively.

In order to prove that the allocation obtained by each iteration of G-PMRM-WIN on each machine p is loser-independent, we need to show that $\hat{V}' \geq \hat{V}$ and $\tilde{V}' \geq \tilde{V}$. In addition, if either $\hat{V}' > \hat{V}$ or $\tilde{V}' > \tilde{V}$, then user i has been selected, and if the obtained social welfare does not change, then either user i has been selected or the allocation of the users does not change. We separate the proof into two cases as follows.

i) $\hat{V}' \geq \hat{V}$, if one user increases her bid. If $\hat{V}' > \hat{V}$, then user i must be the user with the maximum bid, and thus, user i is in the solution (i.e., $\hat{x}'_i = 1$). If $\hat{V}' = \hat{V}$, then the allocation of the users does not change unless user i declares a bid $\hat{b}'_i = \hat{V}$, and she is selected by the algorithm.

ii) $\tilde{V}' \geq \tilde{V}$, we consider two subcases (a) and (b). In subcase (a), we consider that the overall amount of resource requests is less than half of the capacities of a PM. Then all users must be selected (i.e., $\tilde{x}'_i = 1, \forall i \in \tilde{\mathcal{U}}'$), where $\tilde{\mathcal{U}} \subseteq \tilde{\mathcal{U}}'$ since user i may declare a smaller bundle. As a result, $\tilde{V}' \geq \tilde{V}$. In subcase (b), we consider that the overall amount of resource requests is at least half of the capacities of a PM. Note that user i has a better bid density by declaring $\hat{\beta}'_i$. If $\tilde{x}'_i = 0$, then $\tilde{x}_i = 0$. If $\tilde{V}' = \tilde{V}$ and $\tilde{x}'_i = 0$, then the allocation would be the same. In addition, if $\tilde{V}' > \tilde{V}$, then $\tilde{x}'_i = 1$.

This proves that the allocation obtained by each iteration of G-PMRM-WIN on each machine p is loser-independent. □

Theorem 3: G-PMRM-WIN is a monotone winner determination algorithm.

Proof: The allocation obtained by each iteration of G-PMRM-WIN on each machine p is loser-independent with respect to the request of each user. If loser-independent algorithms are employed as sub-procedures in a greedy iterative approach, then the overall winner determination algorithm is monotone [37], [38]. Therefore, the overall allocation on all PMs obtained by G-PMRM-WIN is monotone. □

Theorem 4: G-PMRM-PAY implements the critical payment.

Proof: We need to prove that Π_i is the critical payment for user $i, \forall i \in \mathcal{N}$. We assume that user i is selected by G-PMRM-WIN (i.e., $x_i = 1$). If user i declares a higher value than Π_i, (i.e., $\hat{b}'_i > \Pi_i$), she wins and pays the same amount Π_i. This is due to the fact that if user i was selected in phase one, then declaring a higher value makes her a winner. In addition, if user i was selected in phase two, declaring a higher value increases her bid density, and thus, user i becomes a winner. If user i declares a lower value than Π_i, (i.e., $\hat{b}'_i < \Pi_i$), this leads to a lower bid density and a lower value. If user i was chosen based on either phase one or two, a lower bid

density and a lower value for the user makes user i a non-winning user. These show that the payment Π_i is the minimum valuation that user i must bid to obtain her required bundle. This payment is between the reserve price and the declared value of the user. The critical payment property holds considering the reserve prices. In the case in which user i is not a winner, she pays 0, thus, satisfying the properties of the critical payment. As a result, the payment determined by G-PMRM-PAY is the critical payment. □

We now show that our proposed mechanism, G-PMRM, is strategy-proof.

Theorem 5: G-PMRM mechanism is strategy-proof.

Proof: The winner determination is monotone (Theorem 3) and the payment is the critical value payment (Theorem 4), therefore, according to [32], our proposed mechanism, G-PMRM, is strategy-proof. □

Theorem 6: The time complexity of G-PMRM is polynomial.

Proof: The time complexity of G-PMRM-WIN is $O(PN(\log N+MR))$. This is because sorting the requests requires $O(N \log N)$, while checking the feasibility of the allocation for each user on each PM requires $O(MR)$. The time complexity of G-PMRM-PAY is polynomial for similar reasons. As a result, the time complexity of G-PMRM is polynomial. □

We now prove that in the case of only one PM (i.e., $P = 1$), G-PMRM-WIN is a 2-approximation algorithm.

Theorem 7: The approximation ratio of G-PMRM-WIN in the case of only one PM is 2.

Proof: Let X^* be set of users in the optimal solution, and V^* be the optimal social welfare. Let X and V be the set of users and the social welfare in the obtained solution by G-PMRM, respectively. We need to prove that $V^* \leq V\alpha$, where α is the approximation ratio.

We consider two cases:

i) V is obtained by the second phase of the winner determination algorithm (i.e., $V = \tilde{V}$). If the amount of overall requests is less than half of the capacities of the physical machine, then all such users must be selected (i.e., $X^* = X$ and $V^* = V$). We now consider that the amount of overall allocated requests is at least one half the resource capacities of a physical machine.

In the optimal solution, for the remaining capacity of that resource, the social welfare is less than V since the second phase is based on bid densities. Thus, the first half contains the most valuable requests. Therefore, $V > V^*/2$.

ii) V is obtained by the first phase of the winner determination algorithm (i.e., $V = \hat{V}$). That means $\hat{V} \geq \tilde{V}$. In the optimal solution, the first half cannot have a better social welfare and the second half cannot have a better social welfare than the first half. As a result, $V > V^*/2$. □

Theorem 8: The approximation ratio of G-PMRM in the case of multiple PMs is 3.

Proof: To prove this theorem we use a result from [38], that states that if the winner determination algorithm is α-approximation on a bin, then the overall approximation ratio of the winner determination algorithm applied iteratively on multiple bins is $\alpha + 1$. Since we proved that the approximation ratio for G-PMRM-WIN on only one PM is 2, then it follows from [38] that the overall approximation ratio of G-PMRM on multiple PMs is 3. □

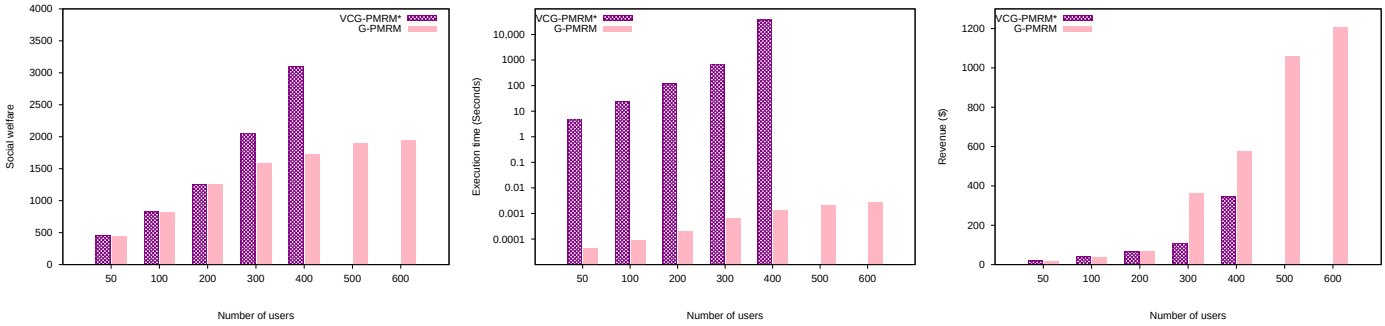
6 Experimental Results

We perform extensive experiments in order to investigate the performance of the proposed mechanism G-PMRM against the performance of the optimal VCG-PMRM mechanism. While it is desirable to compare G-PMRM with several other mechanisms, we found out that the existing mechanisms and approaches are not directly comparable to ours and decided to compare it with the optimal mechanism, VCG-PMRM. Therefore, we rely on the optimal results obtained by VCG-PMRM as a benchmark for our experiments. For the VCG-PMRM mechanism, we use the IBM ILOG CPLEX Optimization Studio Multiplatform Multilingual eAssembly to solve the PMRM problem optimally. The mechanisms are implemented in C++ and the experiments are conducted on AMD 2.4GHz Dual Proc Dual Core nodes with 16GB RAM which are part of the WSU Grid System. In this section, we describe the experimental setup and analyze the experimental results.

6.1 Experimental Setup

The generated requests are based on realistic data combining publicly available information provided by Amazon EC2 and Microsoft Azure as follows. We consider the same types of VM instances available to users as those offered by Amazon EC2. Each of these VM instances has specific resource demands with respect to two available resource types: cores and memory. We also set the amount of each resource type provided by a VM instance to be the same as in the specifications provided by Amazon Web Services for its Spot Instances and Elastic Compute Cloud (EC2) (See Table 1). Users can request a bundle of VMs, where for each VM type, they can request between 0 and 20 VM instances.

We generate bids based on Amazon Spot market report on users bidding strategies [3]. Amazon regularly updates its spot price history based on the past 90 days of activity. Amazon reported that most users bid between the price of reserved instances and on-demand prices. By doing so, these users saved between 50% and 66% compared to the on demand prices. The lowest price of the reserved instances is for the *Heavy Utilization Reserved Instances* which is $\$0.018$ per hour for a medium VM instance of General Purpose - Current Generation. However, the trade-off is that the user's requested bundles can be reclaimed by a cloud provider if the spot price exceeds their submitted bid prices. Thus, some users bid above on-demand prices and up to twice the on-demand prices in some cases. To generate bids, we



(a) (b) (c)

Fig. 2: G-PMRM vs. VCG-PMRM: (a) Social welfare; (b) Execution time; (c) Revenue. (*VCG-PMRM was not able to determine the allocation for 500 and 600 users in feasible time, and thus, there are no bars in the plots for those cases)

TABLE 4: Simulation Parameters

Param.	Description	Value(s)
N	Number of users	[50-600]
M	Number of VM instances	4 (M,L,XL,2XL)
R	Number of resource types	2 (Core, Memory)
PM	Number of PMs	100
C_1	Core capacity	512 cores
C_2	Memory capacity	1244.9 GB
w_{mr}	Amount of resource r provided by a VM instance m	as in Table 1
k_{im}	Number of requested VM of type m by user i	[0, 20]
b_i^0	bid of user i for a medium VM	$[0.018, 0.34]$
b_i	value of user i	$b_i^0 \sum_{m=1}^{M} 2^{m-1} k_{im}$

generate a random number, b_i^0, for each user i from the range $[0.018, 0.34]$ for a medium VM instance, where the range is given by the lowest price for the reserved Amazon EC2 medium instance and the on-demand price for the medium instance. Then, we multiply the random number by the total weights of VMs in the user's requested bundle. The total weight of a VM instance for user i is $\sum_{m=1}^{M} 2^{m-1} k_{im}$. For both mechanisms G-PMRM and VCG-PMRM, we set the reserve prices of the VMs to the half of the posted prices for the Heavy Utilization Reserved Instances. Such reservation prices are reasonable considering the VM costs and a low profit margin. The parameters and their values used in the experiments are listed in Table 4.

We setup the PM configurations based on the specification of the Titan system [39] at Oak Ridge National Laboratory (No. 2 in Top500 [40]). Titan currently contains 299,008 CPU cores and 710 terabytes of memory. We consider 100 PMs available from Titan, where each PM has 512 cores and 1244.9 gigabytes of memory.

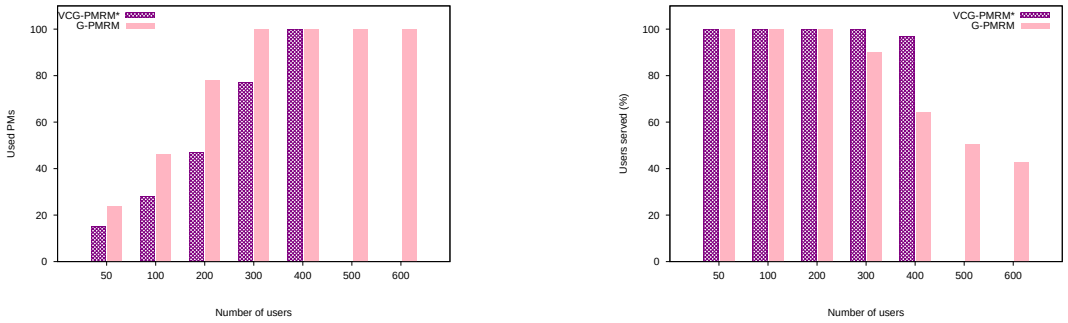
6.2 Analysis of Results

We compare the performance of G-PMRM and VCG-PMRM for different numbers of users, ranging from 50 to 600. For 500 and 600 users, the optimal mechanism, VCG-PMRM, could not find the solutions even after 24 hours. This is due to the fact that the problem is strongly NP-hard, and it is infeasible to solve for large instances.

Fig. 2a shows the social welfare obtained by the mechanisms for 50 to 600 users. The results show that G-PMRM obtains the optimal social welfare when user demand is low compared to the available capacity. For example, for 200 users, G-PMRM and VCG-PMRM obtain the same social welfare of 1259.19. However, the results show that with the increase of the user demand, optimality gap increases. For example, for 300 users, the social welfare obtained by G-PMRM and VCG-PMRM is 1582.25 and 2056.26, respectively. For the first five groups of users for which the optimal mechanism could find solutions, the optimality gap is 23%. This gap is reasonable given the fact that the execution time of the VCG-PMRM is very large as we show in the next figure.

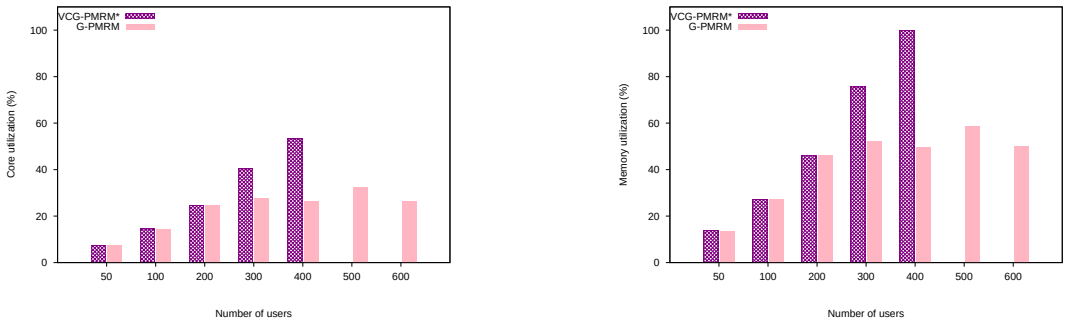
Fig. 2b shows the execution times of the mechanisms on a logarithmic scale. The execution time of VCG-PMRM is more than five orders of magnitude greater than that of G-PMRM. G-PMRM is very fast being suitable for real cloud settings, where the number of PMs and users are large. In addition, in auction-based mechanisms the response time should be very small. For example, Amazon runs its Spot Market auction-based mechanism every hour, and needs to find the winning users and their payments as soon as possible. In particular, the optimal VCG-PMRM mechanism is not feasible when the problem scales. VCG-PMRM was not able to determine the allocation for 500 and 600 users in feasible time, and thus, there are no bars in the plots for those cases. The results of Fig. 2a and 2b show that when the user demand is low, VCG-PMRM obtains the results in reasonable time. However, the execution time of VCG-PMRM is prohibitive when the demand is high. On the other hand, G-PMRM not only obtains the optimal results when the user demand is low, but also it obtains reasonable solutions very fast when the demand is high.

Fig. 2c shows the revenue obtained by the cloud provider using both mechanisms. The results show that G-PMRM and VCG-PMRM obtains the same revenue for the cloud provider when user demand is low compared to the available capacity. However, when the demand is high, G-PMRM obtains a higher revenue than VCG-PMRM. This is due to the fact that VCG-PMRM fulfills



(a) (b)

Fig. 3: G-PMRM vs. VCG-PMRM: (a) Used PMs; (b) Users served. (* see note in Fig. 2)



(a) (b)

Fig. 4: G-PMRM vs. VCG-PMRM: (a) Core utilization; (b) Memory utilization. (* see note in Fig. 2)

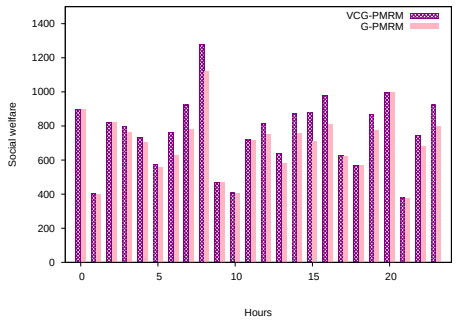


Fig. 5: G-PMRM vs. VCG-PMRM: Social welfare over time.

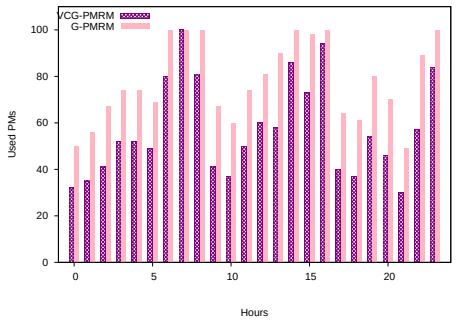


Fig. 6: G-PMRM vs. VCG-PMRM: Used PMs over time.

more requests, which in turn, leads to accepting more bids, and thus, reducing the price. Note that both mechanisms charge users bellow their submitted bids.

Fig. 3a shows the percentage of used PMs for the mechanisms. With an increase in the number of users, both mechanisms activate more PMs. For example for 100 users, G-PMRM powers on about 46 percent of all the PMs, while VCG-PMRM powers about 28 percent of all the PMs. By increasing the number of users, all PMs are activated by both mechanisms. This can be seen for 400 users.

Fig. 3b shows the percentage of served users by the mechanisms. When the demand is low, all users are served by both mechanisms. However, by increasing in the demand, the percentage of served users decreases. Note that a higher percentage of served users does not necessarily result in a higher revenue.

Figs. 4a and 4b show the percentage of resource utilization for both mechanisms. The percentage of core and memory utilization increases with the increase in the number of users. This is due to the fact that with more demand, the cloud provider can better utilize its resources.

We now analyze the performance of the mechanisms over an extended period of time of 24 hours, where users dynamically arrive and submit their requests. We consider that between 50 and 150 users arrive every

TABLE 5: Different scenarios for User A's request declaration

Case	$\hat{\rho}_A$	$\hat{b}_A$	Scenario
I	$< 0, 0, 0, 15 >$	\$35.14	$\hat{\rho}_A = \rho_A, \hat{b}_A = b_A$
II	$< 0, 0, 0, 15 >$	\$50	$\hat{\rho}_A = \rho_A, \hat{b}_A > b_A$
III	$< 0, 0, 0, 15 >$	\$25	$\hat{\rho}_A = \rho_A, \hat{b}_A < b_A$
IV	$< 0, 0, 0, 15 >$	\$10	$\hat{\rho}_A = \rho_A, \hat{b}_A < b_A$
V	$< 0, 0, 0, 20 >$	\$35.14	$\hat{\rho}_A > \rho_A, \hat{b}_A = b_A$
VI	$< 0, 0, 0, 100 >$	\$35.14	$\hat{\rho}_A > \rho_A, \hat{b}_A = b_A$

hour and that each user requests resources for 1 to 4 hours. Fig. 5 shows the social welfare obtained by the mechanisms for each hour. The results show that G-PMRM obtains the optimal social welfare in several cases. When the number of users' requests accumulates, G-PMRM cannot always find the optimal solution. However, the optimality gap is small as guaranteed by the results of Theorem 8. Fig. 6 shows the percentage of PMs used in each hour. When the number of users' requests accumulates, both mechanisms use all the available PMs. However, with a decrease in the number of requests, both mechanisms can turn off several PMs to reduce the energy costs.

We also perform experiments to investigate the effect of untruthful reporting on the utility of the users. In these experiments, we consider the case with 300 users, and select one of the winning users (i.e., User A) with true request of 15 VMs of type m3.2xlarge and true valuation of 35.14. We consider User A's declarations that are different from her true request as shown in Table 5, where Case I represents User A true request. Fig. 7 shows the payment and the utility of User A for all these cases.

In case II, User A submits a request with a higher bid and G-PMRM selects User A as a winner determining the same payment for her as in case I. In case III, User A submits a request with a lower bid, where the bid is not less than her payment determined by G-PMRM. In this case also, the user is selected as a winner, and her payment remains the same. In case IV, User A reports a lower bid than her bid in case I (the true valuation). G-PMRM does not select the user as a winner leading to zero utility for her. Therefore, User A did not increase her utility by this untruthful reporting. If User A requests a larger bundle as shown in case V, she obtains the bundle due to available capacities. However, her payment increases while her utility decreases due to requesting more VMs. If User A requests a larger bundle as in case VI, G-PMRM does not select the user as a winner leading to zero utility for her. These cases show that if any user submits an untruthful request, she can not increase her utility, that is the mechanism is strategy-proof.

From all the above results we can conclude that G-PMRM decides the allocation much faster than VCG-PMRM, achieves a social welfare close to the optimal, and obtains a higher revenue than VCG-PMRM. The performance of G-PMRM scales very well with the number of users.

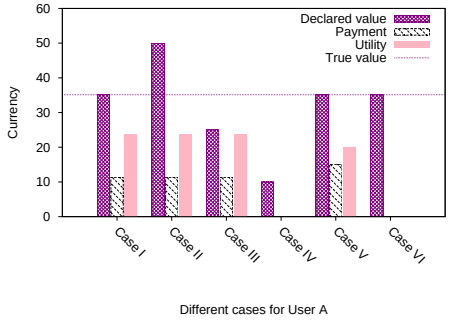


Fig. 7: G-PMRM: Effect of untruthful declarations on the user.

7 Conclusion

We proposed optimal and approximate strategy-proof mechanisms for resource management in clouds in the presence of multiple PMs and multiple types of resources that give incentives to the users to reveal their true valuations for the requested bundles of VM instances. Therefore, our mechanisms do not put the burden on users to compute complex strategies of how to best interact with the mechanisms. We investigated the properties of our proposed mechanisms by performing extensive experiments. The results showed that the performance of our proposed approximation mechanism scales very well with the number of users. We plan to implement the mechanism as part of an integrated solution for dynamic resource management in an experimental cloud computing system.

Acknowledgment

This research was supported in part by NSF grants DGE-0654014 and CNS-1116787.

References

[1] H. Fu, Z. Li, C. Wu, and X. Chu, "Core-selecting auctions for dynamically allocating heterogeneous vms in cloud computing," in *Proc. 7th IEEE Intl. Conf. Cloud Comput.*, 2014, pp. 152–159.

[2] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 713–728, 2005.

[3] "Amazon EC2 Spot Instance Curriculum," [Online]. Available: http://aws.amazon.com/ec2/spot-tutorials/.

[4] H. Xu and B. Li, "Dynamic cloud pricing for revenue maximization," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 158–171, 2013.

[5] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 215–228, 2013.

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

[7] J. Dong, X. Jin, H. Wang, Y. Li, P. Zhang, and S. Cheng, "Energy-saving virtual machine placement in cloud data centers," in *Proc. 13th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Comput.*, 2013, pp. 618–624.

[8] C. Ghribi, M. Hadji, and D. Zeghlache, "Energy efficient VM scheduling for cloud data centers: Exact allocation and migration algorithms," in *Proc. 13th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Comput.*, 2013, pp. 671–678.

[9] M. Maurer, I. Brandic, and R. Sakellariou, "Adaptive resource configuration for cloud infrastructure management," *Future Generation Comput. Syst.*, vol. 29, no. 2, pp. 472–487, 2013.

[10] M. Kesavan, R. Soundararajan, A. Gavrilovska, I. Ahmad, O. Krieger, and K. Schwan, "Practical compute capacity management for virtualized datacenters," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 88–100, 2013.

[11] M. Hu, J. Luo, and B. Veeravalli, "Optimal provisioning for scheduling divisible loads with reserved cloud resources," in *Proc. 18th IEEE Int. Conf. on Networks*, 2012, pp. 204–209.

[12] C.-W. Tsai, W.-C. Huang, M.-C. Chiang, C.-S. Yang, and M.-H. Chiang, "A hyper-heuristic scheduling algorithm for cloud," *IEEE Trans. Cloud Comput.*, vol. 99, no. PrePrints, p. 1, 2014.

[13] J. Doyle, R. Shorten, and D. O'Mahony, "Stratus: Load balancing the cloud for carbon emissions control," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 116–128, 2013.

[14] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proc. USENIX Workshop on Power Aware Comput. and Syst.*, vol. 10, 2008.

[15] M. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 99, no. PrePrints, p. 1, 2014.

[16] M. Mazzucco, D. Dyachuk, and R. Deters, "Maximizing cloud providers' revenues via energy aware allocation policies," in *Proc. 3rd IEEE Int. Conf. on Cloud Comput.*, 2010, pp. 131–138.

[17] M. Polverini, A. Cianfrani, S. Ren, and A. Vasilakos, "Thermal-aware scheduling of batch jobs in geographically distributed data centers," *IEEE Trans. Cloud Comput.*, no. PrePrints, p. 1, 2013.

[18] L. Mashayekhy, M. Nejad, D. Grosu, Q. Zhang, and W. Shi, "Energy-aware scheduling of mapreduce jobs for big data applications," *IEEE Trans. Parallel Distrib. Syst. (forthcoming)*.

[19] A. Khosravi, S. K. Garg, and R. Buyya, "Energy and carbon-efficient placement of virtual machines in distributed cloud data centers," in *Proc. Int. European Conf. Parallel and Distrib. Comput.*, 2013, pp. 317–328.

[20] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[21] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya *et al.*, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in Computers*, vol. 82, no. 2, pp. 47–111, 2011.

[22] L. Wang and S. U. Khan, "Review of performance metrics for green data centers: a taxonomy study," *The Journal of Supercomputing*, vol. 63, no. 3, pp. 639–656, 2013.

[23] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud," in *Proc. 3rd IEEE Int. Conf. Cloud Comput.*, 2010, pp. 236–243.

[24] A. Prasad and S. Rao, "A mechanism design approach to resource procurement in cloud computing," *IEEE Trans. Comput.*, 2014.

[25] G. N. Iyer and B. Veeravalli, "On the resource allocation and pricing strategies in compute clouds using bargaining approaches," in *Proc. 17th IEEE Int. Conf. on Networks*, 2011, pp. 147–152.

[26] Z. Kang and H. Wang, "A novel approach to allocate cloud resource with different performance traits," in *Proc. 10th IEEE Intl. Conf. on Services Computing*, 2013, pp. 128–135.

[27] M. Mihailescu and Y. M. Teo, "On economic and computational-efficient resource pricing in large distributed systems," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud and Grid Comput.*, 2010, pp. 838–843.

[28] D. Niyato, A. Vasilakos, and Z. Kun, "Resource and revenue sharing with coalition formation of cloud providers: Game theoretic approach," in *Proc. IEEE/ACM Int. Symp. Cluster, Cloud & Grid Comput.*, 2011, pp. 215–224.

[29] F. Teng and F. Magoules, "Resource pricing and equilibrium allocation policy in cloud computing," in *Proc. 10th IEEE Int. Conf. Computer and Inf. Tech.*, 2010, pp. 195–202.

[30] L. Mashayekhy, M. M. Nejad, and D. Grosu, "A truthful approximation mechanism for autonomic virtual machine provisioning and allocation in clouds," in *Proc. ACM Cloud and Autonomic Comput. Conf.*, 2013, pp. 1–10.

[31] M. M. Nejad, L. Mashayekhy, and D. Grosu, "A family of truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," in *Proc. of the 6th IEEE Intl. Conf. on Cloud Comput.*, 2013, pp. 188–195.

[32] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic game theory*. Cambridge University Press, 2007.

[33] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *The Journal of Finance*, vol. 16, no. 1, pp. 8–37, 1961.

[34] E. H. Clarke, "Multipart pricing of public goods," *Public choice*, vol. 11, no. 1, pp. 17–33, 1971.

[35] T. Groves, "Incentives in teams," *Econometrica: Journal of the Econometric Society*, vol. 41, no. 4, pp. 617–631, 1973.

[36] A. Mu'Alem and N. Nisan, "Truthful approximation mechanisms for restricted combinatorial auctions," *Games and Economic Behavior*, vol. 64, no. 2, pp. 612–631, 2008.

[37] B. Lucier and A. Borodin, "Price of anarchy for greedy auctions," in *Proc. 21st ACM-SIAM Symp. on Discrete Algo.*, 2010, pp. 537–553.

[38] C. Chekuri and I. Gamzu, "Truthful mechanisms via greedy iterative packing," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2009, pp. 56–69.

[39] "Titan," [Online]. Available: http://www.olcf.ornl.gov/titan/.

[40] "Top 500 Supercomputers," [Online]. Available: http://www.top500.org.

Lena Mashayekhy received her BSc degree in computer engineering-software from Iran University of Science and Technology, and her MSc degree from the University of Isfahan. She is currently a PhD candidate in computer science at Wayne State University, Detroit, Michigan. She has published more than twenty peer-reviewed papers in venues such as IEEE Transactions on Parallel and Distributed Systems, IEEE BigData, IEEE CLOUD, ICPP, etc. Her research interests include distributed systems, cloud computing, big data analytics, game theory and optimization. She is a student member of the ACM, the IEEE, and the IEEE Computer Society.

Mahyar Movahed Nejad received his BSc degree in mathematics from Iran University of Science and Technology. He received his MSc degree in socio-economic systems engineering from Mazandaran University of Science and Technology. He is currently a MSc student in computer science, and a PhD candidate in industrial and systems engineering at Wayne State University, Detroit. His research interests include cloud computing, big data analytics, game theory, network optimization, and integer programming. His publications appeared in journals such as IEEE Transactions on Parallel and Distributed Systems. He is a student member of the IEEE and the INFORMS.

Daniel Grosu received the Diploma in engineering (automatic control and industrial informatics) from the Technical University of Iaşi, Romania, in 1994 and the MSc and PhD degrees in computer science from the University of Texas at San Antonio in 2002 and 2003, respectively. Currently, he is an associate professor in the Department of Computer Science, Wayne State University, Detroit. His research interests include parallel and distributed systems, cloud computing, parallel algorithms, resource allocation, computer security, and topics at the border of computer science, game theory and economics. He has published more than ninety peer-reviewed papers in the above areas. He has served on the program and steering committees of several international meetings in parallel and distributed computing. He is a senior member of the ACM, the IEEE, and the IEEE Computer Society.