A PTAS Mechanism for Provisioning and Allocation of Heterogeneous Cloud Resources

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

Abstract—Cloud providers provision their heterogeneous resources such as CPUs, memory, and storage in the form of virtual machine (VM) instances which are then allocated to the users. One of the major challenges faced by the cloud providers is to allocate and provision these resources such that their profit is maximized, and the resources are utilized efficiently. Recently, cloud providers have introduced auction-based models which allow users to submit bids for their requested VMs. We address the problem of autonomic VM provisioning and allocation for the auction-based model considering multiple types of resources by designing an approximation mechanism. In addition, the mechanism determines the payment the users have to pay for using the allocated resources. This problem is computationally intractable, and our proposed mechanism is by far the strongest approximation result that can be achieved for this problem. We show that the proposed approximation mechanism is a polynomial-time approximation scheme (PTAS). Furthermore, our proposed mechanism drives the system into an equilibrium in which the users do not have incentives to manipulate the system by untruthfully reporting their VM bundle requests and valuations. We perform extensive experiments using real workload traces in order to investigate the performance of the proposed mechanism.

Index Terms—Cloud computing, strategy-proof mechanism, virtual machine provisioning, resource allocation, polynomial time approximation scheme

1 Introduction

Cloud computing systems provide a large pool of abstracted, virtualized, and dynamically scalable resources to users as Infrastructure as a Service (IaaS). More specifically, the resources are offered to users as different types of virtual machine (VM) instances based on a pay-as-you-go model. For example, Amazon Elastic Compute Cloud (Amazon EC2) [1] is currently offering four types of VM instances: Medium (M), Large (L), Extra-Large (XL), and 2Extra-Large (2XL), charging a fixed price per hour for each type of VM instance.

The ever-growing complexity of IaaS makes human administration and management inefficient and, in most of the cases, unfeasible. Therefore, avoiding direct management actions in resource allocation, VM provisioning, VM pricing, and monitoring, requires self-management and self-optimizing mechanisms. The aim of this paper is to design such mechanisms that facilitate autonomic provisioning of cloud resources based on the user demand and the availability of resources. The proposed mechanisms can be incorporated in system tools for self-managing the cloud infrastructure [2].

Recently, cloud providers have introduced auction-based models when offering IaaS, which allow users to name their own price for their requested VM instances. Auctions have been proven to be effective market-based mechanisms for trading cloud services which not only benefit users by allowing them to obtain their requested resources at lower prices, but also allow cloud providers to utilize more resources and increase their profits. Mainstream cloud provider powerhouses such as Amazon have been offering cloud services in an auction market, the Amazon spot market, for several years. Based on Amazon's report [3], many of its users such as Scribd, So-net, Numerate, Backtype, and Fliptop saved more than 50 percent using its auction-based cloud market over its fixed-price market. In addition, a new initiative by Deutsche Börse Cloud Exchange, will launch in 2014 a vendor-neutral marketplace for cloud resources. This market will be a platform for offering, buying and deploying IaaS in an auction setting [4].

We consider an auction market with a set of users and a set of heterogeneous VM instances, where each user can bid for any arbitrary combinations of VMs (bundle of VMs). Our setup and mechanisms are different from the Amazon spot market. The Amazon spot market allows requests only for individual VM instances and not for bundles of VM instances of different types. Therefore, users have to request each VM for their bundle individually. However, in such an auction-based setting, there is no guarantee of receiving the requested VMs all together. In our setting, we allow users to request bundles of heterogeneous VM instances such as requesting communication-intensive VMs and computation-intensive VMs together. In practice, there are many applications that require such heterogeneous bundles of VMs. For example, a social game application composed of three layers:

• *The authors are with the Department of Computer Science, Wayne State University, Detroit, MI 48202. E-mail: {mlena, mahyar, dgrosu}@wayne.edu.*

Manuscript received 11 Apr. 2014; revised 21 July 2014; accepted 1 Sept. 2014. Date of publication 4 Sept. 2014; date of current version 7 Aug. 2015. Recommended for acceptance by K. Li.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2355228

front-end web server, load balancing, and back-end data storage, requires a bundle of heterogeneous VMs composed of communication-intensive VMs, computation-intensive VMs, and storage-intensive VMs, respectively [5].

Each user has a private value for her requested bundle. In our model, each user is interested in a single bundle of VM instances, and bid only for that bundle. Such a user obtains the specified value if she is allocated the whole bundle of VM instances (or any superset of it) and zero value, if she is allocated any other bundle. The users are also selfish in the sense that they want to maximize their own utilities. It may be beneficial for them to manipulate the system by declaring different bundles or bids from their actual requests. In an untruthful auction, users may bid much lower (than their actual valuations) which not only may hurt other users, but also may indirectly lead to profit losses for the cloud provider. Thus, unless strategy-proofness is enforced, maximizing the revenue may not be effective. In strategy-proof auctions, the dominating strategy for users is to bid truthfully, thereby eliminating the fear of market manipulations and the overhead of strategizing over others. When users report their true valuations, the cloud provider can allocate its resources efficiently to users who value them the most. However, allowing users to bid on bundles of VMs makes the design of strategy-proof mechanisms more challenging. This is due to the fact that by allowing bids on bundles, the dimensionality of the problem increases. A major problem in such settings is determining the optimal allocation and protecting against manipulations by the users. Because finding the optimal allocation is computationally intractable [6], designing strategy-proof approximation mechanisms for solving the problem is of major interest.

In this paper, we design a strategy-proof polynomial time approximation scheme (PTAS) mechanism that solves the VM instance provisioning and allocation problem. The goal is to find an allocation of resources to the users maximizing the social welfare, where the social welfare is the sum of users' valuations. We also design an exponential time strategy-proof optimal mechanism that will serve as a benchmark for the performance of the proposed PTAS mechanism. Our proposed PTAS mechanism is strategy-proof that is, the users do not have incentives to lie about their requested bundles of VM instances and their valuations. The proposed mechanism is designed to adapt to changing conditions (i.e., users requests) and to lead the system into an equilibrium in which users do not have incentives to manipulate the system by untruthfully reporting their resource requests and valuations.

1.1 Our Contributions

We address the problem of VM provisioning and allocation in clouds in the presence of multiple types of heterogeneous resources. First, we design a strategy-proof optimal mechanism that uses a dynamic programming algorithm to optimally select the winning users. We then design a strategy-proof approximation mechanism in spite the fact that approximation algorithms, in general, do not necessarily satisfy the properties required to guarantee strategy-proofness. In doing so, the allocation and payment determination of the proposed mechanisms are designed to satisfy the strategy-proofness property. Strategy-proof mechanisms drive the system into an equilibrium in which the users do not have incentives to manipulate the system by untruthfully reporting their VM bundle requests and valuations. We also show that the proposed approximation mechanism is a polynomial-time approximation scheme which is by far the strongest approximation result that can be achieved for this problem, unless $P = NP$. To the best of our knowledge, this is the first study proposing a strategy-proof PTAS mechanism for solving the VM provisioning and allocation problem in clouds. The proposed mechanism allows dynamic provisioning of VMs, and does not require pre-provisioning the VMs. As a result, cloud providers can fulfill dynamic market demands efficiently. A key property of our proposed mechanism is the consideration of multiple types of heterogeneous resources for VMs which is the case in real cloud settings. We analyze the properties of the proposed mechanism and perform extensive experiments. The results show that the proposed PTAS mechanism determines near optimal allocations while satisfying the strategy-proofness property.

1.2 Related Work

The primary objective of mechanism design is to obtain system wide solutions for problems where multiple self-interested users with private information interact. Mechanism design provides a framework for designing mechanisms that align the system's incentives with those of the participants. For a comprehensive introduction to mechanism design, the reader is referred to [7].

Researchers approached the problem of VM provisioning in clouds from different points of view. Wood et al. [8] proposed an approach for dynamic provisioning of VMs by defining a unique metric based on the consumption of the three resources: CPU, network and memory. Ferrer et al. [9] proposed a toolkit for the cloud service and infrastructure providers. The toolkit aims to provide a foundation for a reliable cloud computing industry, by addressing the whole service life cycle. Casalicchio et al. [10] proposed a heuristic (hill-climbing) algorithm to maximize revenue in VM allocation problems satisfying capacity, availability, SLA, and VM migration constraints. Bjorkqvist et al. [11] proposed an opportunistic service provisioning strategy minimizing the service provisioning costs by provisioning a small number of faster VMs, while maintaining the target system utilization. Mashayekhy et al. [12] proposed a federation formation mechanism for resource provisioning and allocation in clouds considering several heterogeneous resources. Xiong et al. [13] considered an economical provisioning, where VMs are allocated to achieve a balanced resource allocation and a better overall performance. Sharma et al. [14] considered minimizing the provisioning cost by reducing the time to transit to new configurations and optimizing the selection of virtual server configurations. Kokkinos et al. [15] proposed an algorithm to minimize the usage cost of Amazon EC2 instances and to maximize the utilization. The algorithm collects information regarding the current instances and then proposes a new set of instances that could be used for serving the same load. However, our main focus is on dynamic resource provisioning that achieves strategy-proofness and leads the system into an equilibrium. We

also propose an approach for determining the prices of the bundles of VM instances.

Pricing and modeling of spot instances have been recently addressed by applying a variety of methodologies. Zhao et al. [20] developed two resource rental planning models, deterministic and stochastic, to minimize the operational cost of cloud applications on spot instances. Leslie et al. [21] proposed a resource allocation and job scheduling framework based on checkpointing. Huang et al. [22] proposed a tool for users to minimize their expenses of running applications in clouds while satisfying the deadlines. The tool automatically determines whether to choose on-demand or spot instances, and also the number of VMs. However, those studies have focused on users/SaaS sides to better use spot instances, while we focus on modeling the problem from IaaS' perspective. Toosi et al. [23] proposed pricing policies for federated clouds that increase utilization and profit. Lampe et al. [24] formulated the equilibrium price auction allocation problem for pricing and distribution of VMs across physical machines as a binary integer program. Based on this mathematical formulation, they proposed an optimal solution approach as well as a fast heuristic approach. However, none of the above-mentioned studies guarantee strategy-proofness.

Designing mechanisms for auction-based cloud markets has attracted a great deal of attention. Zaman and Grosu [16] proposed a truthful-in-expectation auction-based mechanism to allocate VM instances that are statically provisioned. However, their mechanism does not consider heterogeneous VMs. In addition, truthfulness-in-expectation is a weaker notion of truthfulness (strategy-proofness). Zhang et al. [18] proposed a truthful auction-based mechanism for resource allocation in clouds in the presence of only one type of resources. Zhang et al. [17] proposed a randomized mechanism for VM allocation in clouds in an auction market considering heterogeneous VMs. Their proposed mechanism is truthful in expectation and is based on a pair of primal and dual LPs. Recently, Fu et al. [5] modeled the heterogeneous VM provisioning problem as a coalitional game, and proposed a core-based pricing method that obtains the optimal solution. Their method guarantees the optimal social welfare, at the expense of not obtaining strategy-proofness.

System heterogeneity plays an important role in determining the dynamics of strategy-proof mechanisms [25]. Our proposed PTAS mechanism takes into account the heterogeneity of the systems and that of user requests when making allocation decisions. In our previous work [19], [26], we proposed a family of strategy-proof greedy mechanisms for solving the VM instance provisioning and allocation problem considering multiple types of resources. These existing greedy mechanisms cannot guarantee near optimal solutions. We also proved that the approximation ratios of the greedy mechanisms is $\sqrt{NRC_{max}}$, where N is the number of users, R is the number of types of resources, and C_{max} is the highest capacity among all resources' capacities. We present a stylized example to show how far the solution obtained by a greedy mechanism can be from the optimal solution for the problem. Let us consider a cloud provider with one type of resource with capacity of 100. We consider two users submitting their requests, where the first one bids \$2 for 1 unit of the resource, while the second user bids \$100 for 100 units of the resources. All greedy mechanisms proposed in [19], choose the first user since it has the highest bid density with a value of 2, where the bid density is the ratio of bid and the amount of resources requested. However, the optimal mechanism chooses the second user with the value of 100. This leads to \$98 loss in revenue for the cloud provider if it chooses the greedy mechanism over the optimal mechanism. However, none of the above-mentioned studies proposed a mechanism guaranteeing a near optimal solution which is the case for our PTAS mechanism proposed in this paper. Our proposed PTAS mechanism is by far the strongest approximation result that can be achieved for the VM provisioning and allocation problem, unless $P = NP$. Our proposed PTAS mechanism represents a big departure from the existing designs of VM allocation mechanisms that only provided a constant approximation guarantee. In Table 1, we summarize the differences between the existing auction-based VM allocation methods.

TABLE 1
Comparison of Auction-Based VM Allocation Methods

Reference	heterogeneous VMs	strategy-proofness	optimality
[16]	x	in expectation	approx.
[17]	✓	in expectation	approx.
[18]	x	✓	approx.
[5]	✓	x	optimal
[19]	✓	✓	approx.
this paper	✓	✓	optimal & PTAS approx.

1.3 Organization

The rest of the paper is organized as follows. In Section 2, we describe the VM provisioning and allocation problem in clouds, and introduce the basic concepts of mechanism design. In Section 3, we present the design of an optimal mechanism for VM provisioning and allocation. In Section 4, we present the proposed PTAS mechanism and characterize its properties. In Section 5, we evaluate the proposed mechanism by extensive experiments. In Section 6, we summarize our results and present possible directions for future research.

2 Problem Statement and Preliminaries

We define the problem of VM provisioning and allocation in clouds in the presence of multiple types of resources as follows. We consider a cloud provider offering a set of M types of VMs, $\mathcal{VM} = \{1, \ldots, M\}$, to users. Each VM consists of heterogeneous resources such as cores, memory, storage, etc. There are a set of R types of resources, $\mathcal{R} = \{1, \ldots, R\}$, where each VM of type $m \in \mathcal{VM}$ has a specific amount of resource of type r denoted by w_{mr}. The cloud provider has restricted capacity, C_r, on each resource $r \in \mathcal{R}$ available for allocation.

Table 2 shows the four types of VM instances offered by Amazon EC2 US West (Northern California) Region at the time of writing this paper. If we consider that CPU represents the type 1 resource, memory, the type 2 resource, and storage, the type 3 resource, we can characterize, for example, the XLarge instance ($m = 3$) by: $w_{31} = 4$, $w_{32} = 15$ GB, and $w_{33} = 80$ GB.

TABLE 2
VM Instance Types Offered by Amazon EC2—US West (Northern California) Region

	Medium $m=1$	Large $m=2$	XLarge $m=3$	2XLarge $m=4$
CPU	1	2	4	8
Memory (GB)	3.75	7.5	15	30
Storage (GB)	4	32	80	160

We consider that the cloud provider receives requests for bundles of VM instances from a set $\mathcal{U}$ of N users. User $i \in \mathcal{U}$, $i = 1, \ldots, N$ submits a request (S_i, b_i), composed of a bundle of VM instances denoted by $S_i = \langle k_{i1}, k_{i2}, \ldots, k_{iM} \rangle$, where k_{im} is the number of requested VM instances of type $m \in \mathcal{VM}$, and a bid, denoted by b_i representing the maximum price the user is willing to pay for using the requested bundle if it is allocated. User i has a true valuation $v_i(S_i)$ for her requested bundle S_i. Note, that for user i, $v_i(S_i) = b_i$ is her true valuation for S_i. An example of a user request is $(< 2, 1, 4, 3 >, \$10)$, which means that the user is requesting two Medium VM instances, one Large VM instance, four XLarge VM instances, and three 2XLarge VM instances, and her bid is \$10. We denote by $a_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$, the total amount of each resource of type r that user i has requested. Note that $a_{ir} > 0$, due to the fact that each VM instance includes all resources.

The goal of the cloud provider is to allocate VMs to users who value the VM instances the most, which can be achieved by maximizing the social welfare. We denote by V the *social welfare*, which is defined as the aggregation of users' valuations, i.e.,

$$V = \sum_{i \in \mathcal{U}} v_i(S_i) x_i, \tag{1}$$

where x_i, $i \in \mathcal{U}$, are decision variables defined as follows:

$$x_i = \begin{cases} 1, & \text{if bundle } S_i \text{ is allocated to user } i, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The problem of *VM provisioning and allocation in clouds* (VMPAC) is to find a subset of users who value the VM instances the most, such that the cloud provider fulfills their requested bundle of VMs along with determining their payments. In doing so, the cloud provider first finds such subset of users and then provisions the requested VMs for the selected users. Finally, it bills the selected users based on all the submitted bids. We denote by $\mathcal{A}$ and $\mathcal{P}$, the allocation function and the payment function, respectively. The allocation function $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_N)$ determines which users receive their requested bundles, and the payment function $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$ determines the amount that each user must pay to the cloud provider.

The request of a user i is denoted by $\theta_i = (S_i, b_i)$. We denote by $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_N)$ the vector of requests of all N users. User i preferences are characterized by a *quasi-linear utility function* defined as the difference between her valuation and payment, $u_i(\boldsymbol{\theta}) = v_i(\mathcal{A}_i(\boldsymbol{\theta})) - \mathcal{P}_i(\boldsymbol{\theta})$, where $\mathcal{A}_i(\boldsymbol{\theta})$ is the allocated bundle to user i, and $\mathcal{P}_i(\boldsymbol{\theta})$ is the determined payment for user i. Each user's bundle and her valuation is private knowledge. However, users are selfish, and each user's goal is to maximize her utility, thus, she may manipulate the mechanism by submitting a different request from her true request to increase her utility. Since the request of a user is a pair of bundle and value, the user can lie about the value by submitting a higher bid in the hope to increase the likelihood of obtaining her requested bundle, or she can lie about her requested bundle. Such manipulations will lead to an inefficient allocation of VMs and will reduce the revenue obtained by the cloud provider if we do not prevent them by design. As a result, we resort to designing strategy-proof mechanisms that determine allocation and payment of users.

To distinguish user i's truthful request $\theta_i = (S_i, b_i)$ from the actual submitted request (that can be untruthful), we denote the actual request by $\hat{\theta}_i = (\hat{S}_i, \hat{b}_i)$. We denote by $\boldsymbol{\theta}_{-i}$ the vector of all requests except user i's request (i.e., $\boldsymbol{\theta}_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_N)$). A mechanism composed of an allocation and a payment function is *strategy-proof* if all users have incentives to report their true requests.

Definition 1 (Strategy-proofness). *A mechanism consisting of an allocation function $\mathcal{A}$ and a payment function $\mathcal{P}$ is* strategy-proof *(or truthful) if for every user i with a true request θ_i and any other request $\hat{\theta}_i$, and for every request of the other users $\hat{\boldsymbol{\theta}}_{-i}$, it satisfies $u_i(\theta_i, \hat{\boldsymbol{\theta}}_{-i}) \geq u_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$.*

The definition implies that a mechanism is strategy-proof if truthful reporting of requests is a dominant strategy. As a result, the users maximize their utilities by truthful reporting of their requests irrespective of the requests of the other users. To design a strategy-proof mechanism the allocation function $\mathcal{A}$ must be monotone and the payment determination function $\mathcal{P}$ must be based on the critical payment [27].

To define monotonicity, we need to introduce a preference relation $\succeq$ on the set of requests as follows: $\hat{\theta}'_i \succeq \hat{\theta}_i$ if $\hat{b}'_i \geq \hat{b}_i$ and $\hat{S}_i =< \hat{k}_{i1}, \hat{k}_{i2}, \ldots, \hat{k}_{iM} >$, $\hat{S}'_i =< \hat{k}'_{i1}, \hat{k}'_{i2}, \ldots, \hat{k}'_{iM} >$ such that $\sum_{m \in \mathcal{VM}} \hat{k}'_{im} w_{mr} \leq \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr}, \forall r \in \mathcal{R}$. That means request $\hat{\theta}'_i$ is more preferred than $\hat{\theta}_i$ if user i requests fewer resources of each type in her current bundle and/or submits a higher bid.

Definition 2 (Monotonicity). *An allocation function $\mathcal{A}$ is* monotone *if it allocates the resources to user i with $\hat{\theta}_i$ as her declared request, then it also allocates the resources to user i with $\hat{\theta}'_i$, where $\hat{\theta}'_i \succeq \hat{\theta}_i$.*

The definition implies that any winning user who receives her requested bundle by submitting a request $\hat{\theta}_i$ will still be a winner if she requests a smaller bundle or submits a higher bid.

Definition 3 (Critical payment). *If $\mathcal{A}$ is a monotone allocation function, for every θ_i, there exist a unique value v_i^c, called* critical payment, *such that $\forall \hat{\theta}_i \succeq (S_i, v_i^c)$, $\hat{\theta}_i$ is a winning declaration and otherwise, is a losing declaration.*

A mechanism having the critical payment as a payment function will charge user i, $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = v_i^c$ if user i wins, and $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = 0$, otherwise.

In the design of our PTAS mechanism for solving VMPAC the allocation function needs to satisfy an additional property, called bitonicity.

Definition 4 (Bitonicity). *A monotone allocation function $\mathcal{A}$ is* bitonic *if for any user i:*

(i) if $\mathcal{A}$ allocates the resources to the user i with $\hat{\theta}_i$ as her declared request, then $v_i(\mathcal{A}(\hat{\theta}'_i, \hat{\boldsymbol{\theta}}_{-i})) \geq v_i(\mathcal{A}(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}))$, where $\hat{\theta}'_i \succeq \hat{\theta}_i$.

(ii) if $\mathcal{A}$ does not allocate the resources to the user i with $\hat{\theta}_i$ as her declared request, then $v_i(\mathcal{A}(\hat{\theta}'_i, \hat{\boldsymbol{\theta}}_{-i})) \geq v_i(\mathcal{A}(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}))$, where $\hat{\theta}_i \succeq \hat{\theta}'_i$.

The allocation function $\mathcal{A}$ is bitonic with respect to $v_i()$. This requires that the welfare does not increase with $v_i()$ when user i loses ($\hat{b_i} < v_i^c$), and it does increase with $v_i()$ when user i wins ($\hat{b_i} > v_i^c$).

In the next sections, we will design an optimal and a PTAS mechanism that solve the VMPAC problem. These mechanisms work as follows. They first receive the declared requests (bundles and bids) from each participating user and then based on the received requests determine the allocation, using their specific allocation function $\mathcal{A}$, and the payments, using their specific payment function $\mathcal{P}$.

3 Strategy-Proof Optimal Mechanism for VM Provisioning and Allocation

In this section, we propose a Vickrey-Clarke-Groves (VCG)-based optimal mechanism that solves VMPAC. A VCG mechanism [28], [29], [30] is defined as follows:

Definition 5 (VCG mechanism). *A mechanism is a Vickrey-Clarke-Groves mechanism if*

(i) $\mathcal{A}$ is an optimal allocation function, and

(ii) $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = \sum_{j \in \mathcal{U} \setminus \{i\}} v_j(\mathcal{A}_j(\hat{\boldsymbol{\theta}}_{-i})) - \sum_{j \in \mathcal{U} \setminus \{i\}} v_j(\mathcal{A}_j(\hat{\boldsymbol{\theta}}))$,

where $\sum_{j \in \mathcal{U} \setminus \{i\}} v_j(\mathcal{A}_j(\hat{\boldsymbol{\theta}}_{-i}))$ is the optimal social welfare that would have been obtained had user i not participated, and $\sum_{j \in \mathcal{U} \setminus \{i\}} v_j(\mathcal{A}_j(\hat{\boldsymbol{\theta}}))$ is the sum of all users valuations except user i's.

We define our proposed VCG-based mechanism that solves the VMPAC problem as follows:

Definition 6. *The VCG-VMPAC mechanism consists of the allocation algorithm DP-VMPAC and the payment function VCG-PAY defined by the VCG payment function (given in Definition 5 (ii)).*

Our proposed VCG-VMPAC mechanism is given in Algorithm 1. The mechanism is run periodically by the cloud provider. It collects the requests from the users, and it determines the allocation by calling the DP-VMPAC allocation algorithm. Once the allocation is determined the mechanism provisions the required number and types of VM instances and determines the payments by calling the VCG-PAY function. The users are then charged the amount determined by the mechanism.

Algorithm 1. VCG-VMPAC Mechanism

1: {Collect user requests.}
2: **for all** $i \in \mathcal{U}$ **do**
3: Collect request $\hat{\theta}_i = (\hat{S}_i, \hat{b}_i)$ from user i
4: {Allocation.}
5: $(V^*, \mathbf{x}^*) =$ DP-VMPAC$(\hat{\boldsymbol{\theta}}, \mathbf{C})$
6: Provisions and allocates VM instances according to $\mathbf{x}^*$.
7: {Payment.}
8: $\mathcal{P} =$VCG-PAY$(\hat{\boldsymbol{\theta}}, \mathbf{C}, V^*, \mathbf{x}^*)$

In order to design a VCG-based mechanism for VMPAC, we need to design an algorithm that provides the optimal solution to VMPAC. The algorithm, called DP-VMPAC, is based on a dynamic programming approach, and it is given in Algorithm 2. The DP-VMPAC algorithm has two input parameters, the vector of users declared requests ($\hat{\boldsymbol{\theta}}$) and the vector of resource capacities $\mathbf{C} = (C_1, \ldots, C_R)$. The algorithm has two output parameters: V^*, the optimal social welfare and $\mathbf{x}^*$, the optimal allocation of VM instances to the users.

Algorithm 2. DP-VMPAC: Optimal Allocation Algorithm

1: **Input:** $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of requests (bundle, bid)
2: **Input:** $\mathbf{C} = (C_1, \ldots, C_R)$; vector of resource capacities
3: **for all** $i \in \mathcal{U}$ **do**
4: **for all** $r \in \mathcal{R}$ **do**
5: $\hat{a}_{ir} = \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr}$, where $\hat{k}_{im} \in \hat{S}_i$
6: $\mathbf{A}_i = (\hat{a}_{i1}, \ldots, \hat{a}_{iR})$
7: $\hat{\mathbf{C}} = \mathbf{C}$
8: **if** $\hat{a}_{1r} \leq C_r, \forall r \in \mathcal{R}$ **then**
9: $V(1, \mathbf{C}) = \hat{b}_1$
10: $\hat{\mathbf{C}} = \mathbf{C} - \mathbf{A}_1$
11: **else**
12: $V(1, \mathbf{C}) = 0$
13: **for all** $j = 2, \ldots, N$ **do**
14: $V(j, \hat{\mathbf{C}}) = \max\{V(j-1, \hat{\mathbf{C}}), V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) + \hat{b}_j\}$
15: $V^* = V(N, \mathbf{C})$
16: Find $\mathbf{x}^*$ by looking backward at $V(j, \hat{\mathbf{C}})$
17: **Output:** $V^*, \mathbf{x}^*$

DP-VMPAC starts by determining $\hat{a}_{ir}$, the amount of each resource of type r requested by user i (lines 3-6). We denote by $\mathbf{A}_i$ the vector specifying the amount of all resource types requested by user i. We also denote by $V(j, \hat{\mathbf{C}})$ the optimal welfare for the subproblem that considers the first j users and the available capacity $\hat{\mathbf{C}}$. The algorithm calculates $V(1, \mathbf{C})$ (lines 8-12). Based on these values, it calculates $V(j, \hat{\mathbf{C}})$, where $j = 2, \ldots, N$ (lines 13-14) according to the following dynamic programming recurrence:

$$V(j, \hat{\mathbf{C}}) = \max\{V(j-1, \hat{\mathbf{C}}), V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) + \hat{b}_j\}. \quad (3)$$

The recurrence considers two cases, not allocating the bundle to j and allocating it to j. If allocating the requested bundle of the jth user increases the value $V(j-1, \hat{\mathbf{C}})$, the algorithm allocates the bundle to the jth user. The maximum between $V(j-1, \hat{\mathbf{C}})$ and $V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) + \hat{b}_j$ gives the optimal value of $V(j, \hat{\mathbf{C}})$. Once the final value $V(N, \mathbf{C})$ is determined, the algorithm finds $\mathbf{x}^*$, the optimal allocation of VM instances, by looking backward at $V(j, \hat{\mathbf{C}})$, as follows. If $V(N, \mathbf{C}) = V(N-1, \mathbf{C})$, then it means that we did not select the Nth user (i.e., $x_N = 0$), and thus, we just recursively work backwards from $V(N-1, \mathbf{C})$. Otherwise, we select that user (i.e., $x_N = 1$), output the Nth request, and recursively work backwards from $V(N-1, \mathbf{C} - \mathbf{A}_N)$.

Theorem 1. *The DP-VMPAC algorithm finds the optimal solution to the VMPAC problem.*

Proof. To prove that $V(N, \mathbf{C})$ computed by DP-VMPAC is optimal, we need to show that the subproblem $V(j, \hat{\mathbf{C}})$ is

optimal for every j and $\hat{\mathbf{C}}$, where j is the number of users ($j \leq N$), and $\hat{\mathbf{C}}$ is the vector representing the available capacities of the resources. We consider two cases, according to the dynamic programming recurrence given in equation (3). In case one, we assume that not allocating the bundle to j is in the optimal solution for the subproblem $V(j, \hat{\mathbf{C}})$. Then, according to equation (3), $V^*(j, \hat{\mathbf{C}}) = V(j-1, \hat{\mathbf{C}})$. The proof in this case is by contradiction. We assume that $V^*(j, \hat{\mathbf{C}})$ is the optimal solution for the subproblem. As a result, we need to check if $V(j-1, \hat{\mathbf{C}})$ is optimal. If $V(j-1, \hat{\mathbf{C}})$ is not optimal, then there would be a better solution $V'(j-1, \hat{\mathbf{C}}) > V(j-1, \hat{\mathbf{C}})$. Then, using the subproblem $V'(j-1, \hat{\mathbf{C}})$, we have: $V^*(j, \hat{\mathbf{C}}) < V'(j-1, \hat{\mathbf{C}})$, which contradicts the fact that $V^*(j, \hat{\mathbf{C}})$ is the optimal solution. Therefore, $V(j-1, \hat{\mathbf{C}})$ is optimal.

In case two, we consider that allocating the bundle to j is in the optimal solution for the subproblem $V(j, \hat{\mathbf{C}})$. Then, according to equation (3), $V^*(j, \hat{\mathbf{C}}) = V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) + \hat{b}_j$. The proof in this case is by contradiction. We assume that $V^*(j, \hat{\mathbf{C}})$ is the optimal solution for the subproblem. As a result, we need to check if $V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j)$ is optimal. If $V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j)$ is not optimal, then there would be a better solution $V'(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) > V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j)$. Then, using the subproblem $V'(j-1, \hat{\mathbf{C}} - \mathbf{A}_j)$, and the rest of the subproblems, $V^*(j, \hat{\mathbf{C}}) < V'(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) + \hat{b}_j$, which contradicts the fact that $V^*(j, \hat{\mathbf{C}})$ is the optimal solution. Therefore, $V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j)$ is optimal.

We conclude that $V^*(j, \hat{\mathbf{C}}) = V(j, \hat{\mathbf{C}})$, and that this property is maintained at all times thereafter. □

DP-VMPAC solves VMPAC optimally in time $O(N (C_{max})^R)$, where $C_{max} = \max_{r \in \mathcal{R}}\{C_r\}$. This is due to the fact that the dynamic programming builds a $(R+1)$-dimensional table, where the first dimension corresponds to the number of users and the other R dimensions correspond to the R types of resources.

Algorithm 3. VCG-PAY: Payment Function

1: **Input:** $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of requests (bundle, bid)
2: **Input:** $\mathbf{C}$; vector of resource capacities
3: **Input:** V^*; optimal welfare
4: **Input:** $\mathbf{x}^*$; optimal allocation
5: **for all** $i \in \mathcal{U}$ **do**
6: $(V'^*, \mathbf{x}'^*) = \text{DP-VMPAC}(\hat{\boldsymbol{\theta}}_{-i}, \mathbf{C})$
7: $sum_1 = 0$
8: $sum_2 = 0$
9: **for all** $j \in \mathcal{U}, j \neq i$ **do**
10: $sum_1 = sum_1 + \hat{b}_j x'^*_j$
11: $sum_2 = sum_2 + \hat{b}_j x^*_j$
12: $\mathcal{P}_i = sum_1 - sum_2$
13: **Output:** $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

The VCG-PAY function is given in Algorithm 3. VCG-PAY has four input parameters, the vector of users declared requests ($\hat{\boldsymbol{\theta}}$), the vector of resource capacities $\mathbf{C}$, the optimal welfare V^*, and the optimal allocation given by $\mathbf{x}^*$. It has one output parameter: $\mathcal{P}$, the payment vector for the users. VCG-PAY calls DP-VMPAC to find the allocation and welfare obtained without user i's participation (line 6). Based on the optimal allocation to the users with and without user i's participation, VCG-PAY finds the payment for user i, where sum_1 is the sum of all values without user i's participation in the mechanism, and sum_2 is the sum of all except user i's value in the optimal case (lines 7-12).

The VCG-VMPAC mechanism is strategy-proof, and it determines the optimal allocation. However, its execution time becomes prohibitive for large instances of VMPAC. More than this, the problem is strongly NP-hard (by a trivial reduction from the multidimensional knapsack problem [31]), and there is no fully polynomial time approximation scheme (FPTAS) for solving it, unless $P = NP$. PTAS is by far the strongest approximation result that can be achieved for this problem, unless $P = NP$. In the next section, we design such a PTAS mechanism for VMPAC.

4 Strategy-Proof PTAS Mechanism for VM Provisioning and Allocation

We now introduce our proposed strategy-proof PTAS mechanism, PTAS-VMPAC. The definition of a PTAS is as follows:

Definition 7. *A maximization problem has a PTAS if for every instance I and for every $\epsilon > 0$, it finds a solution S for I in time polynomial in the size of I that satisfies $S(I) \geq (1-\epsilon)\ S^*(I)$, where $S^*(I)$ is the optimal value of a solution for I.*

We define the PTAS-VMPAC mechanism that solves the VMPAC problem as follows:

Definition 8. *The PTAS-VMPAC mechanism consists of the allocation algorithm PTAS-ALLOC and the payment function C-PAY.*

The PTAS-VMPAC mechanism is given in Algorithm 4. The mechanism is run periodically by the cloud provider. It collects the requests from the users and determines the allocation by calling the PTAS-ALLOC algorithm. Once the allocation is determined the mechanism provisions the required number and types of VM instances, and then it determines the payments by calling the C-PAY function. The users are then charged the payment determined by the mechanism. PTAS-ALLOC and C-PAY are presented in the following sections.

Algorithm 4. PTAS-VMPAC Mechanism

1: {Collect user requests.}
2: **for all** $i \in \mathcal{U}$ **do**
3: Collect request $\hat{\theta}_i = (\hat{S}_i, \hat{b}_i)$ from user i
4: {Allocation .}
5: $(V, \mathbf{x}) = \text{PTAS-ALLOC}(\hat{\boldsymbol{\theta}}, \mathbf{C}, q)$
6: Provisions and allocates VM instances according to $\mathbf{x}$.
7: {Payment.}
8: $\mathcal{P} = \text{C-PAY}(\hat{\boldsymbol{\theta}}, \mathbf{C}, q)$

Our proposed mechanisms, VCG-VMPAC and PTAS-VMPAC, provision the VMs based on the requests of the users. The mechanisms can handle dynamic changes of heterogeneous user demands by supporting dynamic provisioning of cloud resources. As a result, cloud providers can employ our proposed mechanisms to fulfill dynamic

market demands efficiently. For example, if a user releases a large VM, then the cloud provider can provision the released resources in the form of smaller VMs if there is a request for them.

4.1 PTAS-ALLOC: Allocation Algorithm of PTAS-VMPAC

Our proposed PTAS allocation algorithm, called PTAS-ALLOC, is given in Algorithm 5. PTAS-ALLOC has three input parameters: the vector of users declared requests $\hat{\boldsymbol{\theta}}$, the vector of resource capacities $\mathbf{C} = (C_1, \ldots, C_R)$, and an integer q, where $q \leq N$. The parameter q controls how close the solution determined by PTAS-ALLOC is to the optimal solution. The PTAS-ALLOC algorithm has two output parameters: V, the total social welfare and $\mathbf{x}$, the allocation of VM instances to the users. Our approximation technique is inspired from Briest et al. [32] who proposed a strategy-proof approximation algorithm for the generalized assignment problem where all bins have the same capacity.

Algorithm 5. PTAS-ALLOC: Allocation Algorithm for VMPAC

1: **Input:** $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of requests (bundle, bid)
2: **Input:** $\mathbf{C} = (C_1, \ldots, C_R)$; vector of resource capacities
3: **Input:** q;
4: $V = -\infty$
5: **for all** $\hat{\mathcal{U}} \subseteq \mathcal{U} : |\hat{\mathcal{U}}| \leq q$ **do**
6: $\hat{\mathbf{x}} = \mathbf{0}$
7: $\hat{V} = 0$
8: $sum_r = 0, \forall r \in \mathcal{R}$
9: **for all** $i \in \hat{\mathcal{U}}$ **do**
10: $\hat{x}_i = 1$
11: $\hat{V} = \hat{V} + \hat{b}_i$
12: **for all** $r \in \mathcal{R}$ **do**
13: $sum_r = sum_r + \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr} \hat{x}_i$
14: **if** $C_r \geq sum_r, \forall r \in \mathcal{R}$ **then**
15: $\tilde{\mathcal{U}} = \mathcal{U} \setminus \hat{\mathcal{U}}$
16: $\hat{q} = |\tilde{\mathcal{U}}|$
17: **for all** $r \in \mathcal{R}$ **do**
18: $d_r = C_r - \sum_{i \in \mathcal{U}} \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr} \hat{x}_i$
19: $\mathbf{d} = (d_1, \ldots, d_R)$
20: **for all** $i \in \tilde{\mathcal{U}}$ **do**
21: **for all** $r \in \mathcal{R}$ **do**
22: $\hat{a}_{ir} = \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr}$
23: $\tilde{a}_{ir} = \lceil \hat{a}_{ir} N^2 / d_r \rceil d_r / N^2$
24: $\tilde{\mathbf{A}}_i = (\tilde{a}_{i1}, \ldots, \tilde{a}_{iR})$
25: {DP to find $(\tilde{V}, \tilde{\mathbf{x}})$ for $(\tilde{\mathcal{U}}, \mathbf{d})$:}
26: $\hat{\mathbf{d}} = \mathbf{d}$
27: **if** $d_r \geq \tilde{a}_{1r}, \forall r \in \mathcal{R}$ **then**
28: $V(1, \mathbf{d}) = \hat{b}_1$
29: $\hat{\mathbf{d}} = \mathbf{d} - \tilde{\mathbf{A}}_1$
30: **else**
31: $V(1, \mathbf{d}) = 0$
32: **for all** $j = 2, \ldots, N - \hat{q}$ **do**
33: $V(j, \hat{\mathbf{d}}) = \max\{V(j-1, \hat{\mathbf{d}}), V(j-1, \hat{\mathbf{d}} - \tilde{\mathbf{A}}_j) + \hat{b}_j\}$
34: $\tilde{V} = V(N - \hat{q}, \mathbf{d})$
35: Find $\tilde{\mathbf{x}}$ by looking backward at $V(j, \mathbf{d})$
36: **if** $V < (\hat{V} + \tilde{V})$ **then**
37: $V = \hat{V} + \tilde{V}$
38: $\mathbf{x} = \hat{\mathbf{x}} + \tilde{\mathbf{x}}$
39: **Output:** V, $\mathbf{x}$

The main idea in the design of PTAS-ALLOC, is finding a partial allocation first and then allocating through dynamic programming the remaining resources based on the rounded requests of the unallocated users. The partial allocation of fewer users is used as a "seed" for the approximate solution and is also allowing us to control the solution error, ϵ. The more users we consider in the partial allocation (greater q), the better the error. In the most extreme case, when q is equal to the total number of users, the algorithm degenerates into an exhaustive search, producing the optimal allocation but making the algorithm computationally infeasible. In addition, as a consequence of dividing the allocation process into a partial allocation and an allocation of rounded requests of remaining users, the resulting overall allocation in each iteration of PTAS-ALLOC is bitonic and monotone. The algorithm chooses the maximum among the allocations obtained in each iteration, and thus, as we will show in Theorem 2, the allocation produced by PTAS-ALLOC is monotone.

The PTAS-ALLOC algorithm iterates over all subsets $\hat{\mathcal{U}}$ of at most q users (lines 5-38). For each such subset the algorithm finds a feasible partial allocation $\hat{\mathbf{x}}$ of at most q users (lines 5-14), determines the amount of partially allocated resources for each of the r types of resources (lines 17-19) and rounds the amount of requested resources by the unallocated users (set $\tilde{\mathcal{U}}$) for each of the r resources (lines 20-24). Then, it uses a dynamic programming approach to find an allocation of bundles based on the rounded requests $\tilde{a}_{ir}$, and the remaining unallocated capacities, d_r (lines 25-33). The algorithm determines the maximum welfare and the corresponding VM instance allocation $\mathbf{x}$ obtained over all iterations (lines 34-38).

We now describe the dynamic programming approach that finds the optimal allocation for the remaining users of the remaining capacities using the rounded requests of users (lines 25-33). In order to formulate the problem as a dynamic program, we consider the subproblem $V(j, \hat{\mathbf{d}})$ which includes the first j remaining users with the available capacity $\hat{\mathbf{d}}$ such that $V(j, \hat{\mathbf{d}})$ is the optimal value of the subproblem. The algorithm first calculates $V(1, \mathbf{d})$ (lines 26-31). Based on these values, it calculates $V(j, \hat{\mathbf{d}})$, where $j = 2, \ldots, N - \hat{q}$ (lines 32-33). The algorithm compares two cases, not allocating the bundle to j and allocating it to j. If allocating the requested bundle of the jth user increases the value $V(j-1, \hat{\mathbf{d}})$, the algorithm allocates the bundle to the jth user. The maximum between $V(j-1, \hat{\mathbf{d}})$ and $V(j-1, \hat{\mathbf{d}} - \tilde{\mathbf{A}}_j) + \hat{b}_j$ gives the optimal value of $V(j, \hat{\mathbf{d}})$, where $\tilde{\mathbf{A}}_j$ is the vector of the rounded sizes of requested resources by user j. We can formulate the dynamic programming recursion as follows:

$$V(j, \hat{\mathbf{d}}) = \max\{V(j-1, \hat{\mathbf{d}}), V(j-1, \hat{\mathbf{d}} - \tilde{\mathbf{A}}_j) + \hat{b}_j\}. \quad (4)$$

This dynamic programming formulation is the same as the one proposed for DP-VMPAC algorithm except that here the available capacity vector $\hat{\mathbf{C}}$ is replaced by the vector of remaining unallocated capacities $\hat{\mathbf{d}}$, and the vector specifying the amount of all resource types requested by user j, $\mathbf{A}_j$, is replaced by $\tilde{\mathbf{A}}_j$, the vector of the rounded sizes of requested resources by user j. Therefore, from Theorem 1, the dynamic programming approach finds the optimal

solution for the allocation of the unallocated resources to the remaining users with rounded requests (corresponding to lines 25-33 of Algorithm 5).

The dynamic programming builds a table of size $(N - \hat{q})$ rows and N^2 columns, where $(N - \hat{q})$ is the number of users and N^2 is the number of possible different sizes for the resource capacities due to rounding of the sizes. As a result, the time complexity of the dynamic programming is $O(N\ (N^2)^R)$, where R is the number of resource types. The algorithm stores $V(N - \hat{q}, \mathbf{d})$ to $\tilde{V}$ as the optimal welfare obtained by the dynamic programming for the selected $\tilde{\mathcal{U}}$, and the corresponding allocation to $\tilde{\mathbf{x}}$. Then, PTAS-ALLOC finds the maximum total social welfare, V across all iterations on the subsets of at most q users. It also finds the allocation $\mathbf{x}$ by $\hat{\mathbf{x}} + \tilde{\mathbf{x}}$ (lines 35-38).

Theorem 2. *PTAS-ALLOC is monotone.*

Proof. To prove that the PTAS-ALLOC is monotone, we need to show that each iteration of the main for loop provides a monotone and bitonic allocation. This is based on a result from [27] that states that if an algorithm A consists of applying the maximum operator among a set of allocation algorithms that are monotone and bitonic, then algorithm A is monotone. In our case, the allocation algorithms are basically the iterations of the main for-loop in PTAS-ALLOC.

We show that one iteration is producing a monotone allocation by considering two cases. First, we consider a user i with declared request $\hat{\theta}_i$ is allocated her requested bundle, and she is in the first q users selected by the algorithm. If user i declares a request $\hat{\theta}'_i \succeq \hat{\theta}_i$ (a smaller bundle or higher bid), the allocation will not change. This satisfies the definition of the monotonicity property, where the winning user is among the first q users. Second, we consider that a user i with declared request $\hat{\theta}_i$ is allocated her requested bundle, and she is not in the first q users. In this case, if user i declares a request $\hat{\theta}'_i \succeq \hat{\theta}_i$, her allocation by dynamic programming will not change. This is due to the fact that she declares a more profitable request. As a result, user i remains among winning users which satisfies the monotonicity property, where the winning user is not among the first q users. This proves the monotonicity of each iteration.

To prove that PTAS-ALLOC is bitonic in each iteration, we consider two cases. First, user i is not among the first q users. If user i is a winning user, then by declaring a better request (a smaller bundle or higher bid), the social welfare can only be increased. If user i is not a winning user, then by declaring a larger bundle or less bid, the social welfare can not be increased. Second, user i is among the first q users. Thus, she is a winning user. If she declares a higher bid, the social welfare will increase. If she declares a smaller bundle, then the remaining capacities of each resource will increase. As a result, the social welfare can only increase. Thus, each iteration is bitonic.

The monotonicity and bitonicity properties of PTAS-ALLOC in each iteration, combined with the fact that the PTAS-ALLOC keeps the allocation that gives the maximum welfare among these iterations, proves the overall monotonicity of PTAS-ALLOC. □

We now show that our proposed allocation algorithm is a PTAS, that is, for every fixed ϵ, its running time is polynomial in the size of the input.

Theorem 3. *The PTAS-ALLOC algorithm is a PTAS.*

Proof. To prove that the algorithm is PTAS, we need to show that the solution determined by the algorithm is in a $(1 - \epsilon)$ neighborhood of the optimal, and that the time complexity of the algorithm is polynomial in N.

We first show that the solution is within $(1 - \epsilon)$ of the optimal solution. Let $\mathbf{x}^*$ be the optimal allocation of the requested bundles, and V^* be the corresponding optimal value. Assume that PTAS-ALLOC determines an allocation $\mathbf{x}$ and a value V. Let $\hat{\mathbf{x}}$ be the optimal allocation when we consider only q users with the highest declared values in the first step. The second step of allocation is allocating the remaining resources given by $\mathbf{d}$ to the users who were not selected in the first step. The rounding procedure for the remaining users, in the second step, increases the size of the requested bundles of those users for each resource type. This may lead to an infeasible allocation of the bundles based on the new rounded sizes. Based on the rounding, the total increase in the size of the requested bundles for each resource is less than d_r/N. In order to make the allocation feasible, we can remove a requested bundle such that it satisfies the capacity constraints for each resource type while decreasing the least amount of value from the objective function. We find those allocated bundles in the second step where for all resource types their size is larger than d_r/N. Among those, we choose the bundle $\hat{S}_i$ with the smallest size. Since in the first step, we chose the q bundles with the highest values, the bundle $\hat{S}_i$ can be the $q + 1$ most valuable bundle. Therefore, user i valuation for this bundle is $v_i(\hat{S}_i) \leq 1/(q+1)V^*$. Removing bundle $\hat{S}_i$ makes the obtained objective function between $(1 - 1/(q+1))V^*$ and V^*. Therefore, we have $(1 - \epsilon)V^* \leq V \leq V^*$, where $\epsilon = 1/(q+1)$.

We now show that the time complexity of PTAS-ALLOC is polynomial in N. The running time depends on the partial allocation of q users and the dynamic programming. The time complexity of the dynamic programming is $O(N(N^2)^R)$, where N is the number of users and N^2 is the size of each resource based on the rounding. The exhaustive search to find a partial allocation is based on the total number of allocations of q users which is $\sum_{i=1}^{q} R\binom{N}{i} \leq qRN^q$. Thus, the time complexity of the algorithm is $O(qRN^{2R+q+1})$. This proves that the algorithm is PTAS. □

4.2 C-PAY: Payment Algorithm of PTAS-VMPAC

The C-PAY function is given in Algorithm 6. The C-PAY function has four input parameters, the vector of users declared requests ($\hat{\boldsymbol{\theta}}$), the vector of resource capacities $\mathbf{C}$, the obtained allocation given by $\mathbf{x}$, and the integer q. It has one output parameter: $\mathcal{P}$, the payment vector for the users. The payments are based on the critical payments of the winning

users. 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.

Algorithm 6. C-PAY: Critical Payment Function

1: **Input:** $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of requests (bundle, bid)
2: **Input: C**; vector of resource capacities
3: **Input:** q;
4: **Input: x**; winning users
5: **for all** $i \in \mathcal{U}$ **do**
6: $\mathcal{P}_i = 0$
7: **if** x_i **then**
8: $l = 0$
9: $h = \hat{b}_i$
10: **while** $(h - l) \geq 1$ **do**
11: $v_i^c = (h + l)/2$
12: $\hat{\theta}_i^c = (\hat{S}_i, v_i^c)$
13: $(V', \mathbf{x}') = \text{PTAS-ALLOC}\ ((\hat{\theta}_1, \ldots, \hat{\theta}_i^c, \ldots, \hat{\theta}_N), \mathbf{C}, q)$
14: **if** x_i' **then**
15: {user i is winning by declaring v_i^c}
16: $h = v_i^c$
17: **else**
18: $l = v_i^c$
19: $\mathcal{P}_i = h$
20: **Output:** $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

Theorem 4. *The payment algorithm, C-PAY, implements the critical payment.*

Proof. To prove that C-PAY determines the critical payment for the users, we need to show that $\mathcal{P}_i$ is the critical payment for user i, i.e., $\mathcal{P}_i$ is the minimum value that she can declare to obtain her requested bundle of VMs. The C-PAY algorithm finds the critical payment when the difference between upper and lower bound is less than 1 (line 10 of C-PAY algorithm). In addition, the algorithm always sets the winning value as an upper bound h, and a losing value as a lower bound, l (lines 16 and 18, respectively). We separate the proof into two parts depending on user i's declared value $\hat{b}_i'$, as follows:

i) User i declares a value $\hat{b}_i'$ greater than $\mathcal{P}_i$, (i.e., $\hat{b}_i' > \mathcal{P}_i$), and the algorithm finds a critical payment of $v_i^{c'}$. To prove that $\mathcal{P}_i$ is the critical payment for user i, we need to show that for every other critical payments (e.g., $v_i^{c'}$), the difference between $v_i^{c'}$ and $\mathcal{P}_i$ is at most one. We claim that $|v_i^{c'} - \mathcal{P}_i| \leq 1$. The proof is by contradiction, i.e., we assume $|v_i^{c'} - \mathcal{P}_i| > 1$. Therefore, there are two cases, either $v_i^{c'} - \mathcal{P}_i > 1$, or $\mathcal{P}_i - v_i^{c'} > 1$. In the first case, user i wins by declaring both $v_i^{c'}$ and $\mathcal{P}_i$, where $v_i^{c'} > \mathcal{P}_i$. Based on lines 16 and 19 of the C-PAY algorithm $v_i^{c'}$ is an upper bound on the payment, and thus, there exists a value l such that $v_i^{c'} - l < 1$ due to the convergence and termination of the binary search. Then, we have l as the lower bound and user i cannot win her bundle by declaring l based on line 18. As a result, $\mathcal{P}_i > l$, and thus $v_i^{c'} - \mathcal{P}_i < 1$, which contradicts the assumption. In the second case, user i wins by declaring both $v_i^{c'}$ and $\mathcal{P}_i$, where $\mathcal{P}_i > v_i^{c'}$. With the same argument, we have $\mathcal{P}_i$ as an upper bound for the payment, and thus, $\mathcal{P}_i - l < 1$ to terminate the binary search. Then, we have l as the lower bound and user i cannot win her bundle by declaring l. As a result, $v_i^{c'} > l$, and thus, $\mathcal{P}_i - v_i^{c'} < 1$, which contradicts the assumption.

ii) User i declares a value $\hat{b}_i'$ less than $\mathcal{P}_i$, (i.e., $\mathcal{P}_i - \hat{b}_i' > 1$). Since the algorithm converges when $\mathcal{P}_i - l < 1$, there exists l such that user i cannot win her bundle of VMs by declaring l. As a result, by declaring $\hat{b}_i'$, user i is not a winning user, and her payment is zero, thus, satisfying the properties of the critical payment.

These show that the payment $\mathcal{P}_i$ is the minimum valuation that user i must bid to obtain her requested bundle. As a result, the payment determined by C-PAY is the critical payment. □

We now show that the proposed mechanism is strategy-proof.

Theorem 5. *The PTAS-VMPAC mechanism is strategy-proof.*

Proof. The allocation algorithm PTAS-ALLOC is monotone (Theorem 2) and the payment function C-PAY determines the critical payment (Theorem 4). Therefore, according to [27], the PTAS-VMPAC mechanism is strategy-proof. □

TABLE 3
Users' Tue Requests

User	1	2	3	4	5	6	7	8
k_{i1}	0	1	2	0	2	1	2	3
k_{i2}	0	2	0	0	0	0	0	2
k_{i3}	2	0	2	2	0	0	0	1
k_{i4}	0	0	3	1	2	3	1	1
b_i	30	18	95	10	5	15	7	80

4.3 Example

We now analyze the effect of untruthful reporting on the utility of the users participating in the PTAS-VMPAC mechanism by considering an example. Our goal is to show that our proposed mechanism, PTAS-VMPAC, is robust against manipulation by a user. The true requests of the eight users are shown in Table 3. We consider the capacities of the two resources as follows: 100 cores, and 1,800 MB of storage. The PTAS-VMPAC ($\epsilon = 0.33$) allocates resources to user 1, 2, 3, 7, and 8 in the case where all users declare their true requests. The payments of the winning users based on C-PAY are 3, 3, 18, 0, and 10, respectively.

We assume that user 8 reports a different request, $\hat{\theta}_8$, from her true request $\theta_8 = (<3, 2, 1, 1>, 80)$, where $S_8 = < 3, 2, 1, 1 >$ and $b_8 = 80$. As shown in Table 4, we analyze different scenarios, where user 8 submits different requests. In addition, Fig. 1 shows the payment and utility of the user for all the cases. In Case I, user 8 submits her true request, that is, $\theta_8 = \hat{\theta}_8$. In this case, user 8 wins, and receives the requested bundle of VMs, S_8. The mechanism charges her \$10 for the bundle, and her utility is $80 - 10 = 70$. In case II, user 8 submits a request with a higher bid $\hat{b}_8 = 90$. In this case, user 8 is still a winner and the mechanism determines

TABLE 4
Different Scenarios for User 8's Request Declaration

Case	$\hat{S}_8$	$\hat{b}_8$	Scenario	Status
I	$<3,2,1,1>$	\$80	$\hat{b}_8 = b_8, \hat{S}_8 = S_8$	W
II	$<3,2,1,1>$	\$90	$\hat{b}_8 > b_8, \hat{S}_8 = S_8$	W
III	$<3,2,1,1>$	\$70	$\hat{b}_8 < b_8, \hat{S}_8 = S_8$	W
IV	$<3,2,1,1>$	\$9	$\hat{b}_8 < b_8, \hat{S}_8 = S_8$	L
V	$<3,2,1,3>$	\$80	$\hat{b}_8 = b_8, \hat{S}_8 > S_8$	W
VI	$<3,2,1,5>$	\$80	$\hat{b}_8 = b_8, \hat{S}_8 > S_8$	L

the same payment for her as in case I, leading to a utility of 70. In case III, she submits a request with a lower bid $\hat{b}_8 = 70$, which is not less than the price determined by our mechanism (i.e., \$10). Thus, user 8 is still winning, and the mechanism charges her the same amount as in case I. However, if user 8 submits a request with a lower bid below the payment, she will not obtain her requested bundle, leading to zero utility. This is shown in case IV, where user 8 submits a bid $\hat{b}_8 = 9$. We now investigate scenarios in which user 8 requests a different bundle than her true bundle. In case V, she submits a larger bundle $\hat{S}_8 = <3,2,1,3>$, where she requests three VM instances of type 2XL instead of 1 (the case of her true request, Case I). In this case, she obtains the bundle due to available capacities. However, she pays more than she pays in case I, II, and III. Thus, her utility decreases. In case VI, she submits a larger bundle $\hat{S}_8 = <3,2,1,5>$, where she requests five VM instances of type 2XL instead of one VM instance (the case of her true request, Case I). However, she becomes a loser since the cloud provider does not have enough resources to fulfill her request. We showed that if a user submits a request untruthfully, she can not increase her utility.

5 Experimental Results

We perform extensive experiments with real workload data in order to investigate the properties of the proposed mechanisms, PTAS-VMPAC, and the VCG-VMPAC. We also compare our proposed mechanisms with a greedy mechanism proposed in [19], called G-VMPAC-II. Here, we use a simpler name, G-VMPAC, to refer to G-VMPAC-II. G-VMPAC allocates the VM instances to users in decreasing order of their density metric (a metric based on the bid of a user and the scarcity of her requested resources). The auctions are generated using four workload logs from the Grid Workloads Archive [33] and the Parallel Workloads Archive [34]. VCG-VMPAC, PTAS-VMPAC, and G-VMPAC mechanisms are implemented in C++ and the experiments are

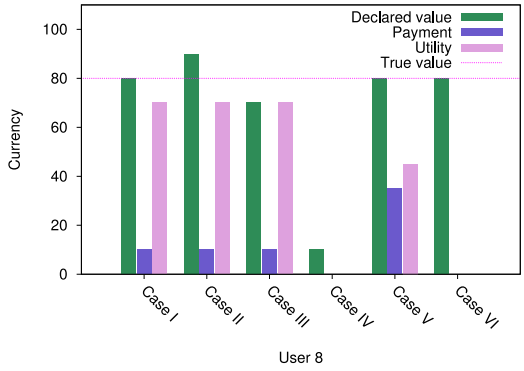

Fig. 1. PTAS-VMPAC: Effect of untruthful declarations.

conducted on Intel 2.93 GHz Quad Proc Hexa Core nodes with 90 GB RAM which are part of the Wayne State Grid System. In this section, we describe the experimental setup and analyze the experimental results.

5.1 Experimental Setup

In the absence of publicly available users requests data from cloud providers, we resort to the well studied and standardized workloads from both the Grid Workloads Archive and the Parallel Workloads Archive.

We selected three logs from the Grid Workloads Archive as follows: 1) NorduGrid traces from the NorduGrid system; 2) AuverGrid traces from the AuverGrid system; 3) SHARCNET traces from SHARCNET clusters installed at several academic institutions in Ontario, Canada. We also selected the following log from the Parallel Workloads Archive. 4) MetaCentrum from the national grid of the Czech republic. We selected these logs because of the availability of CPU and memory requests/usage recorded. We consider each hour of a log as one auction, where each job corresponds to a user request. For each log, except GWA-T-10 SHARCNET, we select 100 hours, while from GWA-T-10 SHARCNET, we select two 100 hours segments. This selection gives five 100-hour auctions for the experiments, and represents different input configurations such as available capacities and number of users. We present the statistics of the logs for the selected segments in Table 5. The number of users (jobs) for each log is given in the second column of Table 5. The total number of users (requests) is 12,613.

We consider each hour of a log as one auction to follow the standard practice in Amazon EC2. As a result, each log represents a series of auctions, where users submits their requests over time to a cloud provider. In each auction hour, the participants are the new users and the unserved users whose deadline has not been exceeded.

The following fields from the log files are selected in order to generate user requests: JobID, SubmitTime,

TABLE 5
Statistics of Workload Logs for the First 100 Hours

Logfile	Number of jobs	Range of CPU	Range of Storage (MB)	Available CPUs	Storage Capacity (MB)
GWA-T-3 NorduGrid	843	1	[1-630]	500	3,000
GWA-T-4 AuverGrid	1,524	1	[2-1,168]	500	10,000
METACENTRUM-2009-2	679	[1-32]	[1-26,755]	500	12,000
GWA-T-10 SHARCNET	2,938	[1-512]	[1-7,812]	10,000	20,000
GWA-T-10 SHARCNET (2)	6,629	[1-150]	[1-8,000]	10,000	20,000

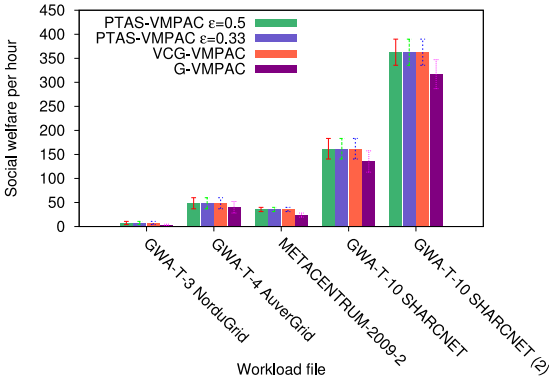

Fig. 2. PTAS-VMPAC versus VCG-VMPAC & G-VMPAC: Social welfare.

RunTime, ReqNProcs, and Used Memory. JobID is the index of the job. SubmitTime and RunTime give the submission time of the job, and the time the job needs to complete its execution, respectively. ReqNProcs and Used Memory give the requested number of processors and the average used memory per processor, respectively.

Each job from the logs corresponds to a user request, where the job's resource usage represents the resources requested by the user. Therefore, each user request includes the requested number of CPUs and the amount of storage. For the bid of each user, we generate a random number between 1 and 10. In addition, we consider a deadline for each user request which is between three to six times of RunTime of each job. Users leave the auctions after their deadline. A user starts bidding for her requested bundle from the SubmitTime until her deadline.

5.2 Analysis of Results

We compare the performance of PTAS-VMPAC (for $\epsilon = 0.5$ and $\epsilon = 0.33$), G-VMPAC and VCG-VMPAC for different workloads in Figs. 2 and 3. The selected ϵ values correspond to q equal to 1 and 2. For each workload, we record the execution time, the social welfare, and the percentage of served users per hour for each mechanism.

Fig. 2 shows the average social welfare per hour for the logs. The reason that we only choose PTAS-VMPAC with $\epsilon = 0.5$ and $\epsilon = 0.33$, and did not select smaller values for ϵ is that for these cases PTAS-VMPAC obtained optimal results equivalent to the one obtained by the optimal VCG-VMPAC for all logs. Note that PTAS-VMPAC guarantees worst case performance, and it does not necessarily produce non-optimal solutions. The rounding procedure of PTAS makes the size of the requests larger than their actual size. For most cases, the total size of the requests in the optimal solutions is not equal to the available capacities. That means, there is extra remaining capacities even in the optimal allocation. As a result, the rounding of the optimal solution still fits in the available capacity. Note that such cases occur irrespective of the amount of aggregated users demands, which can be much higher than the available capacities. The optimality of PTAS-VMPAC depends on the total size of the requests in the optimal solution and the available capacities. Our proposed PTAS-VMPAC mechanism can find the optimal solution in such cases. In fact, there should be a specific configuration in the requested bundles and the available capacities in order to create the worst case scenario (in terms of performance guarantee of ϵ) for the PTAS-VMPAC mechanism. Later in this section, we provide a discussion on the cases where PTAS-VMPAC cannot achieve optimal social welfare. Since G-VMPAC considers only the bid densities when making allocation decisions, it obtains the lowest social welfare in general, which is far from the optimal social welfare. For example, for GWA-T-10 SHARCNET (2) the optimal social welfare is 362.70, while G-VMPAC obtains a social welfare of 317.05 leading to a 12.58 percent gap from the optimal solution obtained by our proposed PTAS-VMPAC.

Fig. 3a shows the average execution time of PTAS-VMPAC ($\epsilon = 0.5$ and $\epsilon = 0.33$), VCG-VMPAC, and G-VMPAC for the logs. In this set of experiments, PTAS-VMPAC with $\epsilon = 0.5$ not only achieves optimal social welfare, but it also has the lowest execution time among all the mechanisms obtaining optimal solutions. For example, in the case of METACENTRUM-2009-2, the execution time of PTAS-VMPAC with $\epsilon = 0.5$ is more than three orders of magnitude greater than that of VCG-VMPAC. VCG-VMPAC has the highest execution time in the GWA-T-10 SHARCNET (2) log since it has the highest available capacity and the highest number of requests 6,629. Note that the complexity of VCG-VMPAC depends on the number of requests and the available capacities. The results show that the execution time of the PTAS-VMPAC is polynomial in the number of requests. One key observation is that since PTAS-VMPAC with $\epsilon = 0.5$ can obtain optimal solutions very fast, thus it is beneficial for the cloud providers to use this mechanism rather than PTAS-VMPAC with other

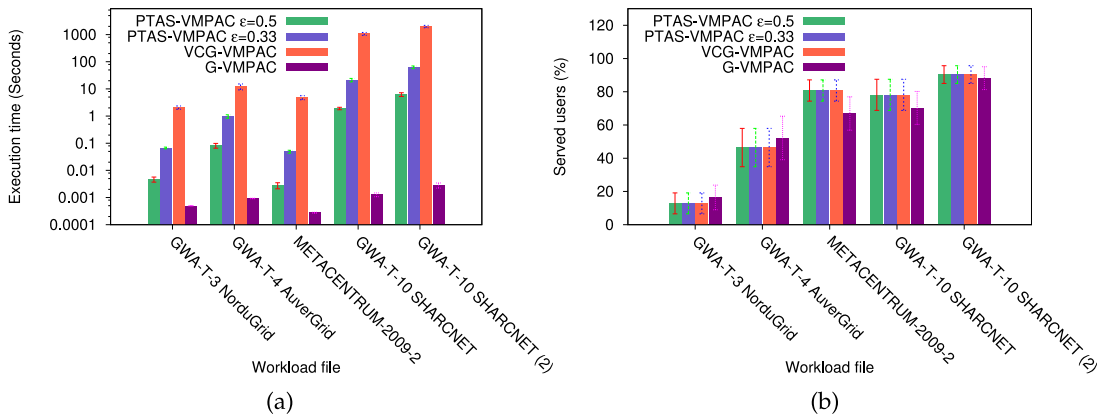

Fig. 3. PTAS-VMPAC versus VCG-VMPAC & G-VMPAC: (a) Execution time; (b) Users served.

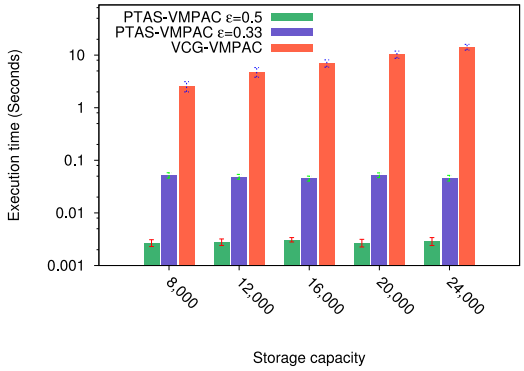

Fig. 4. PTAS-VMPAC versus VCG-VMPAC, sensitivity analysis on execution time with respect to storage capacity.

values for ϵ. For example, for the total of 2,938 and 6,629 requests, PTAS-VMPAC with $\epsilon = 0.5$ finds the optimal solutions in 1.90 and 6.23 seconds, respectively. G-VMPAC finds the results very fast since it is a greedy mechanism. However, it cannot guarantee a near optimal solution, which is the case for our proposed PTAS-VMPAC mechanism.

Fig. 3b shows the percentage of users that have been allocated by the mechanisms. Since the PTAS-VMPAC mechanism achieves the optimal solutions for all logs, it serves the same percentage of users as VCG-VMPAC. Note that VCG-VMPAC does not serve a higher number of users than G-VMPAC. This is due to the fact that the optimal mechanism finds the most valuable subset of users in order to maximize the social welfare.

To show that the execution time of PTAS-VMPAC does not depend on the values of capacity, we perform sensitivity analysis with respect to storage capacity in Fig. 4. For this figure, we select the METACENTRUM-2009-2 log, and choose different storage capacities in each experiment. Fig. 4 shows that the execution time of PTAS-VMPAC mechanism does not change by increasing or decreasing the capacity. This is not the case for VCG-VMPAC where its execution time depends on the number of requests and the available capacities. For example, the execution times of VCG-VMPAC for storage capacity of 8,000 and 24,000 are 2.56 and 14.23 seconds, respectively.

We now investigate the cases where PTAS-VMPAC cannot achieve optimal social welfare. This happens in cases where the allocated amount of resources in the optimal solution is the same as the amount of available capacities. In such cases, the rounding procedure of PTAS-VMPAC needs extra amount for each request. Therefore, the optimal solution would not fit in the available capacities. As a result, at least one of the requests would not be fulfilled by PTAS-VMPAC, leading to a lower social welfare. Since in the logs, we do not have such cases, we designed a special experiment to show such a scenario. We select one auction of METACENTRUM-2009-2, and choose the capacities in a way that PTAS-VMPAC mechanisms do not necessarily find the optimal solution. Fig. 5a shows the obtained social welfare based on the selected ϵ, where ϵ is 0.5, 0.33, 0.25 corresponding to q equal to 1, 2, 3, respectively. We also show the social welfare in the optimal case obtained by VCG-VMPAC. The results show that the obtained social welfare is within ϵ distance of the optimal social welfare. Fig. 5b shows the execution time of PTAS-VMPAC for the same selected ϵ, and the execution time of VCG-VMPAC. The results show that PTAS-VMPAC is able to find a near optimal social welfare in much shorter time. This is due to the fact the PTAS-VMPAC is a polynomial time approximation scheme.

From all the above results, we conclude that PTAS-VMPAC finds near-optimal solutions to the VMPAC problem and its execution time only depends on the number of users and the selected ϵ. These properties make PTAS-VMPAC a good candidate for deployment on the current cloud computing systems, where the capacities of the resources available for allocation are very large.

6 Conclusion

We addressed the problem of dynamic VM provisioning, allocation, and payment determination in clouds considering heterogeneous resources. We proposed a strategy-proof PTAS mechanism for autonomic resource allocation in clouds that provides incentives to the users to reveal their true valuations for the requested bundles of VM instances. We also designed a strategy-proof VCG-based mechanism using a dynamic programming approach. The objectives of the proposed mechanism are to maximize the social welfare in dynamic resource provisioning, to achieve strategy-proofness, and to lead the system into an equilibrium. We investigated the properties of our proposed PTAS mechanism by performing extensive experiments. The results

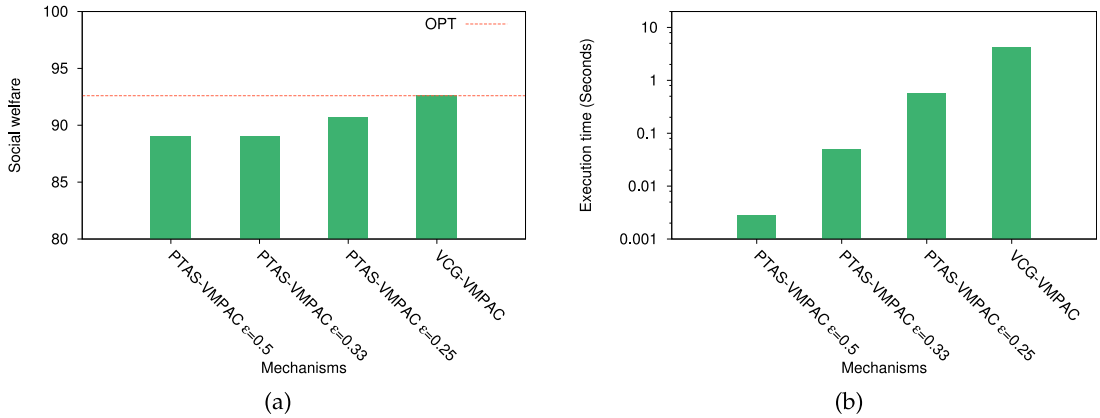

Fig. 5. PTAS-VMPAC versus VCG-VMPAC (special case): (a) Social welfare; (b) Execution time.

showed that the proposed mechanism determines near optimal allocations while giving the users incentives to report their true valuations for the bundles of VM instances. We plan to implement our mechanism as part of an integrated solution for the management of resources in an experimental cloud computing system.

Acknowledgments

This paper is a revised and extended version of [35] presented at the ACM Cloud and Autonomic Computing Conference (CAC 2013). The authors wish to express their thanks to the editor and the anonymous referees for their helpful and constructive suggestions, which considerably improved the quality of the paper. This research was supported, in part, by National Science Foundation (NSF) grants DGE-0654014 and CNS-1116787.

References

[1] Amazon. (2013). Amazon ec2 instance types [Online]. Available: http://aws.amazon.com/ec2/instance-types/
[2] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 289–302, 2007.
[3] Amazon. (2014). Amazon ec2 spot instances [Online]. Available: http://aws.amazon.com/ec2/purchasing-options/spot-instances/
[4] J. Watzl, "A framework for exchange-based trading of cloud computing commodities," Ph.D. dissertation, Ludwig Maximilian Univ. Munich, Munich, Germany, 2014.
[5] H. Fu, Z. Li, C. Wu, and X. Chu, "Core-selecting auctions for dynamically allocating heterogeneous VMs in cloud computing," in *Proc. IEEE 7th Int. Conf. Cloud Comput.*, 2014, pp. 152–159.
[6] D. Lehmann, R. Müller, and T. Sandholm, "The winner determination problem," in *Combinatorial Auctions*, P. Cramton, Y. Shoham, and R. Steinberg, Eds. Cambridge, MA, USA: MIT Press, 2006.
[7] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge, U.K.: Cambridge Univ. Press, 2007.
[8] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. 4th USENIX Conf. Netw. Syst. Des. Implementation*, 2007, vol. 7, pp. 11–13.
[9] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan, "Optimis: A holistic approach to cloud service provisioning," *Future Generation Comput. Syst.*, vol. 28, no. 1, pp. 66–77, 2012.
[10] E. Casalicchio, D. A. Menascé, and A. Aldhalaan, "Autonomic resource provisioning in cloud systems with availability goals," in *Proc. ACM Cloud Auton. Comput. Conf.*, 2013, pp. 1–10.
[11] M. Bjorkqvist, L. Chen, and W. Binder, "Opportunistic service provisioning in the cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 237–244.
[12] L. Mashayekhy, M. M. Nejad, and D. Grosu, "Cloud federations in the sky: Formation game and mechanism," *IEEE Trans. Cloud Comput.*, vol. 99, no. PrePrints, 2014.
[13] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu, "Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach," in *Proc. 31st Int. Conf. Distrib. Comput. Syst.*, 2011, pp. 571–580.
[14] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proc. 31st Int. Conf. Distrib. Comput. Syst.*, 2011, pp. 559–570.
[15] P. Kokkinos, T. Varvarigou, A. Kretsis, P. Soumplis, and E. Varvarigos, "Cost and utilization optimization of amazon ec2 instances," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, 2013, pp. 518–525.
[16] S. Zaman and D. Grosu, "Combinatorial auction-based allocation of virtual machine instances in clouds," *J. Parallel Distrib. Comput.*, vol. 73, no. 4, pp. 495–508, 2013.
[17] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: A randomized auction approach," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 433–441.
[18] H. Zhang, B. Li, H. Jiang, F. Liu, A. V. Vasilakos, and J. Liu, "A framework for truthful online auctions in cloud computing with heterogeneous user demands," in *Proc. IEEE Conf. Comput. Commun.*, 2013, pp. 1510–1518.
[19] M. Nejad, L. Mashayekhy, and D. Grosu, "Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 99, no. PrePrints, 2014.
[20] H. Zhao, M. Pan, X. Liu, X. Li, and Y. Fang, "Optimal resource rental planning for elastic applications in cloud market," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp.*, 2012, pp. 808–819.
[21] L. M. Leslie, Y. C. Lee, P. Lu, and A. Y. Zomaya, "Exploiting performance and cost diversity in the cloud," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, 2013, pp. 107–114.
[22] H. Huang, L. Wang, B. C. Tak, L. Wang, and C. Tang, "Cap 3: A cloud auto-provisioning framework for parallel processing using on-demand and spot instances," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, 2013, pp. 228–235.
[23] A. Toosi, R. Calheiros, R. Thulasiram, and R. Buyya, "Resource provisioning policies to increase IaaS provider's profit in a federated cloud environment," in *Proc. IEEE 13th Int. Conf. High Perform. Comput. Commun.*, 2011, pp. 279–287.
[24] U. Lampe, M. Siebenhaar, A. Papageorgiou, D. Schuller, and R. Steinmetz, "Maximizing cloud provider profit from equilibrium price auctions," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 83–90.
[25] Y. Wang, A. Nakao, and A. V. Vasilakos, "Heterogeneity playing key role: Modeling and analyzing the dynamics of incentive mechanisms in autonomous networks," *ACM Trans. Auton. Adaptive Syst.*, vol. 7, no. 3, p. 31, 2012.
[26] 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. IEEE 6th Int. Conf. Cloud Comput.*, 2013, pp. 188–195.
[27] A. Mu'Alem and N. Nisan, "Truthful approximation mechanisms for restricted combinatorial auctions," *Games Econ. Behav.*, vol. 64, no. 2, pp. 612–631, 2008.
[28] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *J. Finance*, vol. 16, no. 1, pp. 8–37, 1961.
[29] E. H. Clarke, "Multipart pricing of public goods," *Public Choice*, vol. 11, no. 1, pp. 17–33, 1971.
[30] T. Groves, "Incentives in teams," *Econometrica: J. Econometric Soc.*, vol. 41, no. 4, pp. 617–631, 1973.
[31] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin, Germany: Springer, 2004.
[32] P. Briest, P. Krysta, and B. Vöcking, "Approximation techniques for utilitarian mechanism design," *SIAM J. Comput.*, vol. 40, no. 6, pp. 1587–1622, 2011.
[33] GWA. (2013). Grid workloads archive [Online]. Available: http://gwa.ewi.tudelft.nl
[34] PWA. (2013). Parallel workloads archive [Online]. Available: http://www.cs.h-uji.ac.il/labs/parallel/workload/
[35] 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 Auton. Comput. Conf.*, 2013, pp. 1–10.

Lena Mashayekhy received the BSc and MSc degrees in computer science from the Iran University of Science and Technology and the University of Isfahan, respectively. She is currently working toward the PhD degree in computer science at Wayne State University, Detroit, Michigan. She has published more than 20 peer-reviewed papers in venues such as *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, *IEEE BigData*, *IEEE CLOUD*, and International Conference on Parallel Processing. 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 the BSc degree in mathematics from the Iran University of Science and Technology. He received the MSc degree in socio-economic systems engineering from the Mazandaran University of Science and Technology. He is currently working toward the MSc degree in computer science, and the PhD degree 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 papers 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. He is currently 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 90 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.