An Online Mechanism for Resource Allocation and Pricing in Clouds

Lena Mashayekhy, *Student Member, IEEE,* Mahyar Movahed Nejad, *Student Member, IEEE,* Daniel Grosu, *Senior Member, IEEE,* and Athanasios V. Vasilakos, *Senior Member, IEEE*

Abstract—Cloud providers provision their various resources such as CPUs, memory, and storage in the form of Virtual Machine (VM) instances which are then allocated to the users. The users are charged based on a pay-as-you-go model, and their payments should be determined by considering both their incentives and the incentives of the cloud providers. Auction markets can capture such incentives, where users name their own prices for their requested VMs. We design an auction-based online mechanism for VM provisioning, allocation, and pricing in clouds that consider several types of resources. Our proposed online mechanism makes no assumptions about future demand of VMs, which is the case in real cloud settings. The proposed online mechanism is invoked as soon as a user places a request or some of the allocated resources are released and become available. The mechanism allocates VM instances to selected users for the period they are requested for, and ensures that the users will continue using their VM instances for the entire requested period. In addition, the mechanism determines the payment the users have to pay for using the allocated resources. We prove that the mechanism is incentive-compatible, that is, it gives incentives to the users to reveal their actual requests. We investigate the performance of our proposed mechanism through extensive experiments.

Index Terms—cloud computing; online truthful mechanism; dynamic pricing; resource allocation.

1 Introduction

Cloud computing is gaining more market share in the IT industry by adding flexibility on resources acquisition, enabling individuals and enterprises to pay only for the resources and services they use. Cloud providers offer their services based on the pay-as-you-go model, enabling the reduction of enterprises' capital and operational costs. One of the major problems in offering such services is designing efficient mechanisms for Virtual Machine (VM) provisioning, allocation, and pricing. Such mechanisms should consider the economic incentives of both cloud users and cloud providers in finding the market equilibrium [1]. Current cloud providers such as Amazon EC2 and Microsoft Azure employ fixed-price and auction-based mechanisms in order to provision resources in the form of VM instances and sell them to the users. The auction-based mechanisms complement the fixed-price models, potentially providing the most cost-effective option for obtaining cloud resources. Obtaining the VMs in an auction market can significantly lower users' computing costs for their jobs [2]. The auction-based mechanisms provide incentives to the users to adjust consumption patterns according to availability, price, and other factors. Existing resource allocation and pricing mechanisms are offline, and thus, they need to collect the information about all users' requests and then decide the allocation of VM instances to users and the prices they need to pay. However, cloud users request VM instances over time, thus, creating an online setting for the provisioning, allocation, and pricing problem. Therefore, cloud providers need to design online mechanisms suitable for such settings in order to provide faster services and to efficiently allocate and price their resources.

One of the challenges in designing online mechanisms is dynamic pricing. A price determination function should consider the incentives of both cloud providers and users. In doing so, it should increase revenue, facilitate healthy competition among users, and increase the efficiency of resource usage. A cloud provider may increase the price to generate more profit. However, in a competitive environment, if the increase in the price is too high, the cloud provider may lose its potential users leading to a profit loss. On the other hand, if the cloud provider sets the price too low, it may become overwhelmed by high demand from users. Since the available capacity is limited, the cloud provider can serve a limited number of users with low price, leading to loss in both profit and reputation. The challenge is how the cloud provider should determine the price to maximize its profit in such competitive markets. Mechanism design considers the incentives of the participants when deciding the allocation and payment. In doing so, the price determination function should determine the payments of the users based on the value the users derive from the services [3]. A fundamental problem with significant economic implications is how the cloud should price its heterogeneous resources at different times under dynamic demand such that its overall profit

• *L. Mashayekhy, M. Nejad, and D. Grosu are with the Department of Computer Science, Wayne State University, Detroit, MI, 48202.*
A. V. Vasilakos is with the Department of Computer Science, Electrical and Space Engineering, Lulea University of Technology, 97187 Lulea, Sweden.
E-mail: mlena@wayne.edu, mahyar@wayne.edu, dgrosu@wayne.edu, vasilako@ath.forthnet.gr

is maximized.

We consider an online market with multiple self-interested users who are competing for cloud resources. In online settings, all users arrive and depart dynamically requiring making decisions without having information about the future. Each user name her own price for a bundle of VM instances, and specifies the amount of time the bundle must be allocated and a deadline. Each user has private information about her requested bundle, and this information is not necessarily reflected in her submitted request. This is due to the fact that the users are self-interested, and they may manipulate the system in order to maximize their utility. A key property of our proposed mechanism, called incentive-compatibility, is to give incentives to users to reveal their actual requests including the price, the VM bundle, the length of time of using the requested VM bundle, and the deadline for their requested bundles. The objective of the mechanism is to allocate cloud resources to the users who value them the most. The mechanism also calculates the price that each user must pay to the cloud provider. The allocation and pricing mechanisms used by the current cloud computing providers do not require the users to explicitly specify a length of time for using a VM at the time of submitting their requests. These cloud providers charge the users a fixed price per hour for each VM instance used. Our proposed mechanism provides more flexibility allowing the users to specify the length of time for which they would like to acquire the bundle of VMs and a deadline. We believe that our setting provides more opportunities to providers, to optimize their operating costs and to increase their profits, as well as to users, who will be able to better express their requirements in order to maximize their utilities.

In this paper, we design an online mechanism for the VM allocation and pricing problem in clouds in the presence of multiple types of resources (e.g., cores, memory, storage, etc.). Our proposed mechanism is online and thus, makes no assumptions about future demand and supply of VMs, which is the case in real cloud settings. Our proposed online mechanism calculates the allocation and payment as users arrive at the system and place their requests. Our proposed mechanism demonstrates the benefits of quick response, revenue maximization, and incentive compatibility which are critical when providing online cloud services.

1.1 Our Contribution

We address the problem of online VM provisioning, allocation, and pricing in clouds in the presence of multiple types of resources. We design an offline incentive-compatible mechanism and an online incentive-compatible mechanism for VM allocation and pricing that give incentives to the users to reveal their actual true requests. Our proposed offline mechanism is optimal given that the information on all the future requests is known a priori. However, our proposed online mechanism makes no assumptions about the future demand for VMs, which is the case in real cloud settings. Our proposed online mechanism is invoked as soon as a user places a request or some allocated resources are released and become available. The mechanism not only provisions and allocates resources dynamically, but also determines the users' payments such that the incentive-compatibility property is guaranteed. We compare the performance of the optimal mechanism with that of online mechanism. The proposed online mechanism provides very fast solutions making it suitable for execution in real-time settings. We perform extensive experiments showing that the proposed online mechanism is able to find near optimal solutions.

1.2 Related Work

Mechanism design [4] is a sub-field of game theory aiming at reaching systems' equilibria having desired properties such as high revenue [4]. There is a rich body of work on mechanism design considering *static systems* in which all participants are present and a one-time decision is made to find a solution [5], [6], [7]. Such systems are considered in an offline setting, whereas in online mechanism design, all participants arrive and depart dynamically, requiring making decisions without having information about the future. The problem of online mechanism design was introduced by Friedman and Parkes [8]. They proposed strategy-proof online mechanisms, where truthful revelation of a user's valuation is a dominant strategy equilibrium. For an introduction to online mechanism design, the reader is referred to Parkes [9]. Several online variants of Vickrey-Clarke-Groves (VCG) mechanisms were proposed by Gershkov and Moldovanu [10] and by Parkes and Singh [11]. These mechanisms focus on Bayesian-Nash incentive compatibility. However, these studies rely on a model of future availability, as well as future supply. Hajiaghayi et al. [12] designed online mechanisms for auctioning identical items, where users have three parameters as private information: value, arrival time, and departure time. However, they assumed that the number of users is known in advance. Hajiaghayi et al. [13] investigated online mechanisms for re-usable items in which items can be allocated to different users at different time slots. They mainly focused on unit-length requests. Porter [14] studied the problem of online scheduling of a re-usable resource in model-free setting, and characterized the monotonicity properties.

Researchers approached the problem of resource provisioning and allocation in clouds from different points of view [15], [16], [17], [18]. Jangjaimon and Tzeng [16] designed an enhanced adaptive incremental checkpointing mechanism for multithreaded applications on resource-as-a-service clouds under spot instance pricing. The objective of their approach is to reduce the expected job turn-around time and the cost. Kuo et al. [17] proposed a 3-approximation algorithm for the VM placement problem to minimize the maximum access latency. Xiao et al. [19] studied the automatic scaling problem in clouds. They proposed a color set algorithm to decide the application placement and load distribution. Their proposed algorithm is invoked periodically, and it reduces the number of instances in order to save energy. Lee and Zomaya [20] proposed two energy-conscious task consolidation heuristics for clouds with the goal of maximizing resource utilization considering both active and idle energy consumption. Papagianni et al. [21] tackled the problem of providing a unified resource allocation framework for networked clouds with the goal of minimizing the cost of resource mapping procedures. Ghazar and Samaan [22] proposed a pricing mechanism for virtual network services to regulate the demand for their shared substrate network resources. Guazzone et al. [23] proposed a framework for dynamic management of computing resources in order to achieve suitable QoS levels and to reduce the amount of energy consumption for providing services. HoseinyFarahabady et al. [24] studied the problem of task assignment on hybrid-clouds. They proposed two approximation methods for two different cases of known and unknown running time of available tasks. More specifically, they designed a fully polynomial-time randomized approximation scheme based on a Monte Carlo sampling method for the case of unknown running time. Leslie et al. [25] proposed a framework for resource allocation and job scheduling of VMs aiming to cost efficiently execute deadline-constrained jobs. Their proposed framework ensures quality of service in terms of cost, deadline compliance and service reliability. Cao et al. [26] proposed a pricing model to maximize profit considering different factors of a cloud such as the amount of services, the workload of an application, the cost of renting, and the cost of energy consumption. In addition, they proposed a queuing model in order to find optimal configuration of a multiserver system. All of these prior works assume that the information is publicly known, and none of them considers a competitive setting, in which the requests characteristics are private to the users.

Recently, the concepts of game theory and mechanism design have been employed in the design of cloud resource management mechanisms [27], [28], [29]. Feng et al. [28] proposed a game theoretic approach considering multiple competing cloud providers. They proposed iterative price determination algorithms for cloud providers to maximize their profits when offering IaaS (Infrastructure as a Service). Zhang et al. [29] proposed a randomized mechanism for VM allocation in clouds in an auction market. Their proposed mechanism is truthful in expectation and is based on a pair of primal and dual LPs (Linear Programs). It considers a different settings than ours in which the the requests from the users do not specify the duration of the time the VM bundle is requested, the arrival time, and the deadline; it only specifies the bundle of VM and its valuation. Prasad et al. [30] proposed a cloud resource procurement approach which not only automates the selection of cloud providers but also implements dynamic pricing. They proposed a strategy-proof mechanism based on VCG, a Bayesian mechanism, and an optimal mechanism for resource procurement where a user performs a reverse auction for procuring resources from cloud providers. Wang et al. [31] proposed a generalized dominant resource fairness mechanism for the multi-resource allocation problem, where there are multiple heterogeneous servers. Their proposed mechanism improves the resource utilization leading to shorter job completion times. In our previous studies, we proposed truthful mechanisms for VM allocation in clouds in periodic-time (offline) settings [32], [33], [34]. However, none of these studies consider online settings.

Online resource management in clouds has recently attracted a great deal of attention. Hua et al. [35] proposed a scalable distributed scheme in cloud data centers considering the network architecture design and data placement. Their proposed network scheme leverages the off-line precomputation to improve online cloud services. Zhang et al. [36] proposed a bandwidth cost minimization approach for uploading deferral big data to a cloud or a federation of clouds. In doing so, they designed a heuristic smoothing algorithm and an efficient distributed randomized online algorithm. Abbasi et al. [37] proposed an online algorithm to minimize operational cost of a set of geo-distributed data centers. Song et al. [38] proposed an online bin packing approach that uses virtualization technology to allocate cloud resources dynamically based on application demands. Their proposed approach supports green computing by optimizing the number of servers used. Zhao et al. [39] proposed an online algorithm for dynamic VM pricing across data-centers in a geo-distributed cloud in order to maximize the overall profit. Zhang et al. [40] proposed an online auction mechanism for resource allocation in clouds in the presence of only one type of resources. They assumed that job lengths and bids are within known intervals. Zaman and Grosu [41] proposed a truthful online mechanism for provisioning and allocation of VM instances in clouds. However, their mechanism assumes that the cloud provider offers only one type of resources, computational resources. The current work is different from the two above-mentioned studies since it considers the existence of several resource types, being more suitable for use in real cloud settings. Note that considering one resource makes the problem NP-hard, while in our study, we tackle a much more challenging problem which is strongly NP-hard. Therefore, satisfying incentive-compatibility in our settings brings about more challenges. In addition, unlike the above-mentioned studies we do not consider any assumptions on the bids and their distributions, and thus, creating a general framework for the online setting.

TABLE 1: VM instance types offered by Amazon EC2.

	Small $m = 1$	Medium $m = 2$	Large $m = 3$	Extralarge $m = 4$
CPU	1	2	4	8
Memory (GB)	1.7	3.75	7.5	15
Storage (GB)	160	410	850	1690

1.3 Organization

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

2 VM Allocation and Pricing Problem

In this section, we model the online VM allocation and pricing (OVMAP) problem in the presence of multiple types of resources. A cloud provider offers R different types of resources, $\mathcal{R} = \{1, \dots, R\}$, such as cores, memory, storage, etc. These resources are provisioned in the form of M types of VM instances $\mathcal{VM} = \{1, \dots, M\}$ and then offered to the users. Each VM instance of type $m \in \mathcal{VM}$ has a specific amount of each type of resource $r \in \mathcal{R}$, denoted by w_{mr}. The capacity C_r for each resource $r \in \mathcal{R}$ available for allocation is limited. In Table 1, we show the four types of VM instances offered by Amazon EC2 US West (Northern California) Region. 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 Medium instance ($m = 2$) by: $w_{21} = 2$, $w_{22} = 3.75$ GB, and $w_{23} = 410$ GB.

A set $\mathcal{U}$ of N users are requesting a set of VM instances for a certain amount of time in order to execute their jobs on the cloud. User i, $i \in \mathcal{U}$, requests a bundle $S_i = \langle k_{i1}, k_{i2}, \dots, k_{iM} \rangle$ of M types of VM instances, where k_{im} is the number of requested VM instances of type $m \in \mathcal{VM}$. In addition, she specifies a bid b_i for her requested bundle S_i. User i's request is denoted by $\theta_i = (S_i, a_i, l_i, d_i, b_i)$, where a_i is the arrival time of her request, l_i is the amount of time for which the requested bundle must be allocated, and d_i is the deadline for her job completion. For example, request $(\langle 4, 3, 1, 2 \rangle, 2, 1, 7, \$15)$ represents a user requesting 4 Small VM instances, 3 Medium VM instances, 1 Large VM instance, and 2 Extra large VM instances; the request arrives at time 2, needs 1 unit of time to execute, expires at time 7, and her bid is \$15. We denote by $\sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$, the total amount of each resource of type r that user i has requested.

We define $\delta_i = d_i - l_i$ as the time by which S_i must be allocated to user i in order for her job to complete its execution. If the cloud provider allocates a requested bundle, the request is never preempted. User i values her requested bundle S_i at b_i, which is the maximum price a user is willing to pay for using the requested bundle if it is allocated within time window $[a_i, \delta_i]$. The users are assumed to be *single-minded*. That means, user i desires only S_i and derives a value of b_i if she gets S_i, or any superset of it, for the specified time before its deadline, and zero value, otherwise.

The standard objective of mechanism design is to maximize welfare [42], which can help a cloud provider increase its revenue. This is due to the fact that the mechanism allocates the VMs to the users who value them the most. The *welfare*, V, is the sum of users' valuations, $V = \sum_{i \in \mathcal{U}} b_i \cdot x_i$, where x_i, $i \in \mathcal{U}$, are decision variables defined as follows: $x_i = 1$, if bundle S_i is allocated to user i within time window $[a_i, \delta_i]$; and $x_i = 0$, otherwise. Our goal is to design an online incentive-compatible mechanism maximizing V, that is, a mechanism that solves OVMAP.

We also define the offline version of OVMAP, called VMAP, which considers that the information on all the future requests is known a priori. In order to formulate VMAP as an integer program we define the decision variables over time $t \in \mathcal{T}$ as follows:

$$X_{it} = \begin{cases} 1 & \text{if } S_i \text{ is allocated to } i \text{ at } t, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

In addition, we define indicator parameters as follows:

$$y_{it} = \begin{cases} 1 & \text{if } a_i \le t \le \delta_i, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The feasibility of the allocation to user i is indicated by y_{it}. This indicator parameter ensures that the allocation of the requested bundle is within time window $[a_i, \delta_i]$.

We formulate the problem of offline VM allocation and pricing (VMAP) as an Integer Program (called VMAP-IP) as follows:

$$\text{Maximize} \sum_{i \in \mathcal{U}} \sum_{t \in \mathcal{T}} b_i \cdot y_{it} \cdot X_{it} \tag{3}$$

Subject to:

$$\sum_{t \in \mathcal{T}} X_{it} \le 1, \forall i \in \mathcal{U} \tag{4}$$

$$\sum_{i \in \mathcal{U}} \sum_{\omega = t - l_i + 1}^{t} \sum_{m \in \mathcal{VM}} k_{im} w_{mr} y_{i\omega} X_{i\omega} \le C_r, \quad \forall r \in \mathcal{R}, \forall t \in \mathcal{T} \tag{5}$$

$$X_{it} = \{0, 1\}, \forall i \in \mathcal{U}, \forall t \in \mathcal{T} \tag{6}$$

$$y_{it} = \{0, 1\}, \forall i \in \mathcal{U}, \forall t \in \mathcal{T} \tag{7}$$

The objective function is to maximize welfare V, where $x_i = \sum_{t \in \mathcal{T}} y_{it} \cdot X_{it}$. Constraints (4) ensure that the request of each user is fulfilled at most once. Constraints (5) guarantee that the allocation of each resource type does not exceed the available capacity of that resource for any given time. Constraints (6) and (7) represent the integrality requirements for the decision variables and indicator parameters. These constraints force the cloud provider to provision the whole bundle of VM instances and to allocate bundles to the selected users. The VMAP problem is strongly NP-hard by a simple reduction from the multidimensional knapsack problem [43]. Note that VMAP-IP assumes that the information about all users' requests is available at the time of solving it. As a result, if solved, VMAP-IP finds the optimal allocation of cloud resources in an offline setting. However, in an online setting, we do not have the information about future requests (such as arrivals), and thus, we have to rely on online mechanisms that solve the OVMAP problem. Our goal is to design such an online incentive-compatible mechanism that solve the OVMAP problem.

3 Mechanism Design Framework

In this section, we first present the basic concepts of mechanism design and then propose an offline optimal mechanism.

3.1 Preliminaries of Mechanism Design

In general, a deterministic mechanism $\mathcal{M}$, is defined as a tuple $(\mathcal{A}, \mathcal{P})$, where $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_N)$ is the allocation function that determines which users receive their requested bundles, and $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$ is the payment rule that determines the amount that each user must pay for the allocated bundles. In our model, each user $i \in \mathcal{U}$ is characterized by her actual request denoted by θ_i. Each user's request is private knowledge. The users may submit different requests from their actual (true) requests. We denote by $\hat{\theta}_i = (\hat{S}_i, \hat{a}_i, \hat{l}_i, \hat{d}_i, \hat{b}_i)$ user i's submitted request. Note that $\theta_i = (S_i, a_i, l_i, d_i, b_i)$ is user i's actual request. The valuation function $v_i(\hat{\theta}_i)$ of user i is defined as follows:

$$v_i(\hat{\theta}_i) = \begin{cases} b_i & \text{if } \hat{S}_i \text{ is allocated by } \mathcal{A} \\ & \quad \wedge (S_i \subseteq \hat{S}_i) \wedge (t_i \leq \delta_i) \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where t_i is the time at which $\hat{S}_i$ has been allocated to user i. The goal is to design incentive-compatible mechanisms that maximize the welfare V, where $V = \sum_{i \in \mathcal{U}} v_i(\hat{\theta}_i) \cdot x_i$.

We denote by $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$ the vector of requests of all users. In addition, $\hat{\boldsymbol{\theta}}_{-i}$ is the vector of all requests except user i's request (i.e., $\hat{\boldsymbol{\theta}}_{-i} = (\hat{\theta}_1, \ldots, \hat{\theta}_{i-1}, \hat{\theta}_{i+1}, \ldots, \hat{\theta}_N)$). The utility function of user i is *quasi-linear*, and thus, it is defined as the difference between her valuation and payment, $u_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}) = v_i(\hat{\theta}_i) - \mathcal{P}_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$, where $\mathcal{P}_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$ is the payment for user i calculated by the mechanism using the payment rule $\mathcal{P}$.

Definition 1 (Individual rationality): A mechanism is *individually-rational* if for every user i reporting her actual request θ_i we have $u_i(\theta_i, \hat{\boldsymbol{\theta}}_{-i}) \geq 0$, for all other users requests $\hat{\boldsymbol{\theta}}_{-i}$.

In other words, a mechanism is individually-rational if a truthful user can always achieve as much utility from participation as without participation. Therefore, users reporting truthfully their requests will never incur losses (i.e., negative utility) by participating in the mechanism. However, such mechanisms do not give incentives to users to report their requests truthfully. The goal of a self-interested user is to maximize her utility, and she may manipulate the mechanism by lying about her actual request. In our case, the request of a user consists of a bundle, an arrival time, an amount of time for which the requested bundle must be allocated, a deadline, and a value. As a result, a user can lie about any of these parameters in the hope to increase her utility. These manipulations may reduce the revenue of the cloud provider. Our goal is to prevent such manipulations by designing incentive-compatible mechanisms for solving OVMAP. A mechanism is *incentive-compatible* if all users have incentives to reveal their actual requests.

Definition 2 (Incentive compatibility): A mechanism $\mathcal{M}$ is *incentive-compatible* (or truthful) if for every user i, for every submitted requests of the other users $\hat{\boldsymbol{\theta}}_{-i}$, an actual request θ_i and any other submitted request $\hat{\theta}_i$ of user i, we have that $u_i(\theta_i, \hat{\boldsymbol{\theta}}_{-i}) \geq u_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$.

In an incentive-compatible mechanism, truthful reporting is a dominant strategy for the users. In other words, it is in the users best interest to submit their actual request irrespective of other users requests. To design an incentive-compatible mechanism, we need to design a monotone allocation function $\mathcal{A}$, while the payment rule must be based on the critical payment [44].

For our model, we define monotonicity in terms of the following preference relation $\succeq$ on the set of requests: $\hat{\theta}'_i \succeq \hat{\theta}_i$ if $\hat{S}_i \succeq \hat{S}'_i$, $\hat{a}'_i \leq \hat{a}_i$, $\hat{l}'_i \leq \hat{l}_i$, $\hat{d}'_i \geq \hat{d}_i$, and $\hat{b}'_i \geq \hat{b}_i$ for user i. Moreover, $\hat{S}'_i \succeq \hat{S}_i$ if $\sigma'_{ir} \leq \sigma_{ir}, \forall r \in \mathcal{R}$. That means the request $\hat{\theta}'_i$ is more preferred than $\hat{\theta}_i$ if user i requests a smaller bundle, submits an earlier request, the bundle for a shorter time period, a later deadline, and submits a higher value. In our setting, users cannot report an earlier arrival (i.e., $\hat{a}_i \leq a_i$), a shorter length (i.e., $\hat{l}_i \leq l_i$), or a later deadline (i.e., $\hat{d}_i \geq d_i$) than their true arrival time, true length, and true deadline. There is no reason for a user to submit her request earlier than when her job is ready for execution. Declaring a shorter length does not allow the completion of the job. Reporting a later deadline may result in getting her bundle too late to complete her job on time.

Definition 3 (Monotonicity): If a *monotone* allocation function $\mathcal{A}$ allocates the resources to user i with $\hat{\theta}_i$, then it also allocates the resources to that user with $\hat{\theta}'_i$, where $\hat{\theta}'_i \succeq \hat{\theta}_i$.

In other words, $\mathcal{A}$ is monotone if any winning user who receives her requested bundle by declaring a request $\hat{\theta}_i$ is still wining if she submits a more preferred request.

In addition to a monotone allocation function $\mathcal{A}$, any incentive-compatible mechanism $\mathcal{M}$ has a payment rule $\mathcal{P}$. To avoid manipulations and satisfy incentive-compatibility, the payment $\mathcal{P}_i$ of any user i, must be independent of her request [42]. In this setting, a payment rule that satisfies the critical payment property along with a monotone allocation function are sufficient conditions to obtain an incentive-compatible mechanism [42]. In the following, we describe the critical payment property.

Definition 4 (Critical payment): If $\mathcal{A}$ is monotone, for every θ_i, there exist a unique value b_i^c, called *critical payment*, such that $\forall\hat{\theta}_i \succeq (S_i, a_i, l_i, d_i, b_i^c)$, $\hat{\theta}_i$ is a winning declaration, and $\forall\hat{\theta}_i \prec (S_i, a_i, l_i, d_i, b_i^c)$, $\hat{\theta}_i$ is a losing declaration.

For given requests $\hat{\boldsymbol{\theta}}_{-i}$ and allocation function $\mathcal{A}$, $\hat{\theta}_i$ is a winning declaration if $i \in \mathcal{A}(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$ (i.e., $x_i = 1$); otherwise we say that $\hat{\theta}_i$ is a losing declaration.

We define the payment rule $\mathcal{P}$ based on the critical payment as follows. $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = b_i^c$, if user i is a winning user, and $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = 0$, otherwise. A winning user is a user who is selected by the allocation function to receive her request (i.e., $x_i = 1$). We denote by b_i^c, the critical payment of user i.

In the next subsection, we incorporate our proposed VMAP-IP in the design of a Vickrey-Clarke-Groves (VCG)-based optimal mechanism which computes the allocation and payment offline.

3.2 Incentive-Compatible Offline Optimal Mechanism

In this section, we present a VCG-based optimal mechanism that solves VMAP, the offline version of OVMAP problem. Since the setting is offline, our proposed mechanism has all the information about the users such as their arrival, deadline, requested time, requested bundle, etc, and thus, it finds the optimal solution. Any VCG-based mechanism [42] requires an optimal allocation algorithm implementing the allocation function $\mathcal{A}$. A VCG mechanism is defined as follows. A mechanism is a Vickrey-Clarke-Groves (VCG) mechanism if the allocation function $\mathcal{A}$ maximizes V, and the payment function $\mathcal{P}$ is defined as follows:

$$\mathcal{P}_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}) = \sum_{j\in\mathcal{A}(\hat{\boldsymbol{\theta}}_{-i})} v_j(\hat{\theta}_j) - \sum_{j\in\mathcal{A}(\hat{\boldsymbol{\theta}}), j\neq i} v_j(\hat{\theta}_j), \forall i \in \mathcal{U}, \tag{9}$$

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

We design a VCG-based mechanism, called VCG-VMAP, that solves the VMAP problem, by incorporating our proposed VMAP-IP and its optimal solution along with the above-mentioned VCG payment rule. The optimal offline VCG-VMAP mechanism is shown in Algorithm 1. The mechanism has as input the vector of resource capacities $\mathbf{C} = (C_1, \ldots, C_R)$. VCG-VMAP collects all the requests (lines 1-3), and when it has all

Algorithm 1 Optimal Offline Mechanism: VCG-VMAP ($\mathbf{C}$)

```
1: for all $i \in \mathcal{U}$ do
2:     Collect user request $\hat{\theta}_i = (\hat{S}_i, \hat{a}_i, \hat{l}_i, \hat{d}_i, \hat{b}_i)$
3: $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$
4: ($V^*$, $\mathbf{x}^*$) = Solution of IP-VMAP($\hat{\boldsymbol{\theta}}$)
5: Provisions and allocates VM instances based on $\mathbf{x}^*$.
6: for all $i \in \mathcal{U}$ do
7:     ($V'^*$, $\mathbf{x}'^*$) = Solution of IP-VMAP($\hat{\boldsymbol{\theta}}_{-i}$)
8:     $\mathcal{P}_i = \sum_{j\in\mathcal{U}, j\neq i} \hat{b}_j(x_j'^* - x_j^*)$
9: Output: $V^*$; $\mathbf{x}^*$; $\mathcal{P}$
```

the information about the requests, it determines the optimal allocation of resources to users by solving the IP-VMAP given in Equations (3) to (7) (line 4). Then, the mechanism provisions the resources in the form of VM instances based on the requested number and types of VM instances of winning users (line 7). Finally, the mechanism determines the payment of each user (lines 6-8). In doing so, VCG-VMAP finds the allocation and welfare obtained without each user's participation (line 7). Then, the mechanism charges each user based on the welfare obtained with and without her participation (line 8). The mechanism returns three parameters: V^*, the optimal welfare, $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_N^*)$, the optimal allocation of VM instances to the users, and $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$ their payments.

Because VCG-VMAP is designed with an optimal allocation function and uses the VCG payment rule, it is incentive-compatible [42]. However, VMAP is strongly NP-hard, and thus, the execution time of VCG-VMAP becomes prohibitive for large instances of VMAP. In addition, VCG-VMAP computes the allocation and payment offline since it has all the information about future demands. However, in a real settings this information is not available to the cloud providers and requires designing online mechanisms. In Section 4, we introduce our proposed online mechanism.

4 Online Mechanism for VM Allocation and Pricing

Our goal is to design an incentive-compatible greedy mechanism that solves the OVMAP problem in online settings.

The VM instances have R dimensions, where the R dimensions correspond to the R types of resources. Since the cloud provider provisions resources in the form of VM instances, any bundle of VMs can be seen as one R-dimensional item. Without loss of generality, we consider that the smallest item in the R-dimensional space contains one unit of each resources. This assumption does not restrict our proposed model since the resource capacities can be normalized to their units. As a result, the total volume of available items to allocate to the users is $\prod_{r\in\mathcal{R}} C_r$. In this section, we present an incentive-compatible online mechanism for the OVMAP problem, called OVMAP.

Algorithm 2 OVMAP Mechanism (Event, $\mathcal{A}, \mathcal{P}$)

```
1: $t \leftarrow$ Current time
2: $\mathcal{Q}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U},\ i$ has not been allocated$\}$
3: $\tilde{\mathcal{Q}}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U},\ (i$ has been allocated$) \wedge$
       (its job has not finished yet)$\}$
4: for all $i \in \mathcal{U}$ do
5:     for all $r \in \mathcal{R}$ do
6:         $\sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$
7: for all $r \in \mathcal{R}$ do
8:     $C_r^t \leftarrow C_r - \sum_{i|\hat{\theta}_i \in \tilde{\mathcal{Q}}^t} \sigma_{ir}$
9: $\mathcal{C}^t \leftarrow (C_1^t, \ldots, C_R^t)$; vector of resource capacities at time $t$
10: if $\mathcal{Q}^t = \emptyset$ or $\mathcal{C}^t = \mathbf{0}$ then
11:     return
12: $\mathcal{A}^t \leftarrow$ OVMAP-ALLOC$(t, \mathcal{Q}^t, \mathcal{C}^t)$
13: $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}^t$
14: $\mathcal{P} \leftarrow \mathcal{P} \cup \{\hat{b}_i | (\hat{\theta}_i, t) \in \mathcal{A}^t\}$
15: $\mathcal{P} \leftarrow$ OVMAP-PAY$(t, \mathcal{Q}^t, \mathcal{A}, \mathcal{P}, \mathcal{C}^t)$
16: return $\mathcal{A}, \mathcal{P}$
```

Algorithm 3 OVMAP-ALLOC$(t, \mathcal{Q}^t, \mathcal{C}^t)$

```
1: $\mathcal{A}^t \leftarrow \emptyset$
2: for all $i | \hat{\theta}_i \in \mathcal{Q}^t$ do
3:     $f_i = \frac{\hat{b}_i}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}}$
4: Sort all $\hat{\theta}_i \in \mathcal{Q}^t$ in non-increasing order of $f_i$
5: for all $\hat{\theta}_i \in \mathcal{Q}^t$ in non-increasing order of $f_i$ do
6:     $\hat{\mathcal{C}} = \mathcal{C}^t$
7:     $flag \leftarrow$ TRUE
8:     for all $r \in \mathcal{R}$ do
9:         $\hat{C}_r = \hat{C}_r - \sigma_{ir}$
10:        if $\hat{C}_r < 0$ then
11:            $flag \leftarrow$ FALSE
12:            break;
13:    if $flag$ then
14:        $\mathcal{C}^t = \hat{\mathcal{C}}$
15:        $\mathcal{A}^t \leftarrow \mathcal{A}^t \cup (\hat{\theta}_i, t)$
16: Output: $\mathcal{A}^t$
```

4.1 OVMAP Mechanism

The OVMAP mechanism is given in Algorithm 2. OVMAP is an event handler, that is, it is invoked when a new user request arrives or some allocated VM instances become available. OVMAP takes as input an Event, the current allocation set $\mathcal{A}$, and the payment set $\mathcal{P}$. An Event is either a release of resources or an arrival of a user request. In lines 1 to 8, OVMAP sets the current time to t and initializes four variables as follows:

$\mathcal{Q}^t$: the set of requests of the users that have not been allocated. Formally,
$\mathcal{Q}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U}, t \leq \hat{\delta}_i \wedge \nexists t_i < t : (\hat{\theta}_i, t_i) \in \mathcal{A}\}$;
$\tilde{\mathcal{Q}}^t$: the set of requests of the users that have been allocated and their jobs have not finished yet. Formally,
$\tilde{\mathcal{Q}}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U} \wedge \exists t_i < t : (\hat{\theta}_i, t_i) \in \mathcal{A} \wedge t_i + \hat{l}_i > t\}$;
σ_{ir}: the amount of each resource of type r requested by user i; and,
C_r^t: the available capacity of the resource r at time t.

The mechanism stores the resource capacities at time t as a vector $\mathcal{C}^t$ (line 9). Then, it proceeds only if resources and requests are available. OVMAP determines the allocation by calling OVMAP-ALLOC. The allocation function OVMAP-ALLOC returns $\mathcal{A}^t$, the set of users who would receive their requested bundles at time t (line 12). The mechanism then updates the overall allocation set $\mathcal{A}$ using the newly determined set $\mathcal{A}^t$. Then, the mechanism determines the payment of users. The payment of users in $\mathcal{A}^t$ are inserted into the payment set with $\mathcal{P}_i = \hat{b}_i$ as their initial payment (line 14). The payment function OVMAP-PAY returns updated set $\mathcal{P}$ containing the payment of users at time t (line 15). The payment of user i is going to be updated several times until $t = \delta_i$, i.e., until the time instance the user must obtain the requested bundle. OVMAP-PAY calculates the payments for these users and updates the payment set $\mathcal{P}$.

4.2 Allocation algorithm of OVMAP

The allocation algorithm OVMAP-ALLOC is given in Algorithm 3. We define a metric called the *bid density*. OVMAP-ALLOC algorithm allocates the VM instances to users in decreasing order of their bid densities. OVMAP-ALLOC considers the setting in which a set $\mathcal{U}$ of N users are requesting a heterogeneous set of VM instances for *any length* of time in order to execute their applications/jobs on the cloud. It also considers a continuous-time model such that $t \in [0, T]$. Note that the request time length for any user i is $\hat{l}_i \geq 1$. The bid density is defined as follows:

$$f_i = \frac{\hat{b}_i}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}} \tag{10}$$

The bid of user i for a bundle of VM instances for time $\hat{l}_i$ can be interpreted as requesting a hyper-rectangle with volume $\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}$ in the $(R+1)$-dimensional space defined by the R resource types and the time. User i values this volume at $\hat{b}_i$, if allocated. Hence, f_i represents how much user i values one unit of volume from the $(R+1)$-dimensional space.

OVMAP-ALLOC sorts all requests in non-increasing order of bid densities, f_i (line 4). Then the algorithm allocates bundles requested by the sorted users in $\mathcal{Q}^t$ while resources last (lines 5-15). OVMAP-ALLOC checks if it can fulfill the request of user i (lines 8-12). If there are not enough resources, user i will not be selected, and her request will be rejected after the current time passes δ_i (by removing user i from $\mathcal{Q}^t$). If there are enough resources, user i will be allocated (line 15) and the amount of available resources will be updated (line 14).

The mechanism uses the non-increasing order of bid densities for allocation because the cloud provider is interested in users who want to pay more per unit of their resources per unit of time. OVMAP-ALLOC tries to maximize the sum of the reported values of the users who get their requested bundles. Finally, OVMAP-ALLOC returns the set $\mathcal{A}^t$ of users who are selected for

Algorithm 4 OVMAP-PAY($t, \mathcal{Q}^t, \mathcal{A}, \mathcal{P}, \mathcal{C}^t$)

```
1: $\mathcal{W} = \{\hat{\theta}_i | \exists t' \le t : (\hat{\theta}_i, t') \in \mathcal{A} \wedge t \le \hat{\delta}_i\}$
2: $\hat{\mathcal{Q}} = \mathcal{Q}^t \cup \mathcal{W}$
3: for all $i | \hat{\theta}_i \in \hat{\mathcal{Q}}$ do
4:     $f_i = \frac{\hat{b}_i}{\hat{l}_i \cdot \prod_{r\in\mathcal{R}} \sigma_{ir}}$
5: for all $\hat{\theta}_i \in \mathcal{W}$ in non-increasing order of $f_i$ do
6:     $\hat{\mathcal{C}} \leftarrow \mathcal{C}^t$
7:     for all $r \in \mathcal{R}$ do
8:         $\hat{C}_r = \hat{C}_r + \sigma_{ir}$
9:     $q = -1$;
10:    $\bar{\mathcal{A}} \leftarrow$ OVMAP-ALLOC($t, \mathcal{Q}^t \setminus \hat{\theta}_i, \hat{\mathcal{C}}$)
11:    for all $\hat{\theta}_j \in \mathcal{Q}^t \cap \{\hat{\theta}_j | (\hat{\theta}_j, t) \notin \mathcal{A}^t \wedge (\hat{\theta}_j, t) \in \bar{\mathcal{A}}\}$
               in non-increasing order of $f_j$, where $f_j < f_i$ do
12:        $q = j$;
13:        break;
14:    if $q$ then
15:        $\mathcal{P}_i \leftarrow \min(f_q \cdot \hat{l}_i \cdot \prod_{r\in\mathcal{R}} \sigma_{ir}, \mathcal{P}_i)$
16:    else
17:        $\mathcal{P}_i \leftarrow 0$
18: Output: $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_N)$
```

allocation at time t.

4.3 Payment function of OVMAP

The payment function OVMAP-PAY is given in Algorithm 4. This function calculates the *critical payment* of each user i if her requested bundle is allocated at t. The critical payment of user i is the minimum value that she must report to get her requested bundle at time t. OVMAP-PAY determines the set $\mathcal{W}$ of requests of users who are allocated and still feasible for allocation at t (line 1). Then, it determines the set $\hat{\mathcal{Q}}$ of requests of users who are allocated or not allocated at t (line 2). OVMAP-PAY calculates f_i for all users in $\hat{\mathcal{Q}}$ (lines 3-4). Then, OVMAP-PAY determines the payment for all users that have been allocated at time t (lines 5-17). The payment of user i is calculated based on the critical value payment. To determine the critical payment, we eliminate user i from the system, add back to C^r the resources allocated to user i, and identify a losing user that becomes a winner because of user i elimination. The value reported by this losing user is the critical value of user i. Thus, only the resources allocated to user i are placed back into C^r (lines 7-8). Then, it calls the allocation algorithm, OVMAP-ALLOC, without considering the participation of user i (line 10). Then, OVMAP-PAY tries to find a user j who had not been allocated at t when user i participated, and would have been allocated at t if user i did not participate (lines 11-17). If OVMAP-PAY finds such a user, it stores her index q (line 12), and it determines the payment of user i based on the density of user q (line 15); otherwise user i pays 0 (line 17). In other words, the payment of user i is calculated by multiplying $\hat{l}_i \cdot \prod_{r\in\mathcal{R}} \sigma_{ir}$ with the highest density among losing users, (i.e., that of user q), who would win if user i would not participate. This is the minimum value to be reported by user i such that she gets her requested bundle. Finally, the set $\mathcal{P}^t$ is returned to the mechanism.

4.4 Example of OVMAP Execution

We show the execution of the mechanism by considering a setting in which the users bid as shown in Table 2. For simplicity, we consider $R = 1$, that is only one resource type is available (e.g., core). As a result, σ_{i1} is the total amount of resources that user i has requested. For example, user 1's bid $\hat{\theta}_1$, contains the following information: her requested resource is $\hat{\sigma}_{11} = 3$, she submits her bid at $\hat{a}_1 = 0$, she requests the bundle for $\hat{l}_1 = 3$ time units, her deadline is $\hat{d}_1 = 5$, and she values the allocation of the bundle for the entire time at $\hat{b}_1 = 5$. We also show for each user, the value of $\hat{\delta}_i = \hat{d}_i - \hat{l}_i$, the time by which the bundle must be allocated to meet the deadline, and $f_i = \frac{\hat{b}_i}{\hat{l}_i \times \hat{\sigma}_{i1}}$, the bid density.

We assume that the available capacity of resource 1 for allocation is 5 units. We show the execution of OVMAP for this setting in Figure 1 and Table 3. In Figure 1a, we show the initial state of the system, in Figure 1b, we show the system state at time $t = 0$, and continue with $t = 1, 2, 3$ in the subsequent figures. We also show the values of C_1^t and sets of $\mathcal{Q}^t$, $\tilde{\mathcal{Q}}^t$, $\mathcal{A}$, and $\mathcal{P}$ for each of the above time instances as a time diagram in Table 3. As a reminder, $\mathcal{Q}^t$ is the set of bids of users that participate at time t, $\tilde{\mathcal{Q}}^t$ is the set of bids of users that are holding some resources at time t (including those who win their bids at time t), C_1^t is the amount of resources available after allocation at time t, and $\mathcal{A}$ and $\mathcal{P}$ are the allocation and payment sets.

In the second column of Table 3, we show the initial system state and the subsequent columns represent the state of the system at time $t = 0, 1, 2, 3$, respectively. Figure 1a shows the initial state, where all resources are available and there are no outstanding bids. In column 2 of Table 3, we see that all sets are empty and $C_1^{0-} = 5$, since all resources are available for allocation. Users 1 and 2 submit their bids at $t = 0$ and hence OVMAP is invoked. Now, $\mathcal{Q}^t = \{\hat{\theta}_1, \hat{\theta}_2\}$, since both users will participate in the mechanism. As shown in Table 2, $f_1 > f_2$, therefore user 1 is allocated a bundle of size $\hat{\sigma}_{11} = 3$. User 2's request ($\hat{\sigma}_{21} = 3$) cannot be satisfied by the remaining resources ($C_1^t = 2$), thus $\hat{\theta}_2$ remains in set $\mathcal{Q}^t$. $\hat{\theta}_1$ is included in the set $\tilde{\mathcal{Q}}^t$, since user 1 is now receiving some resources. Finally, $(\hat{\theta}_1, 0)$ is added to set $\mathcal{A}$ and $(\hat{\theta}_1, 4)$ is added to set $\mathcal{A}$, since the value of the payment for user 1 determined in line 15 of OVMAP-PAY is $\mathcal{P}_1^t = f_2 \cdot \hat{l}_1 \cdot \hat{\sigma}_{11} = 4$.

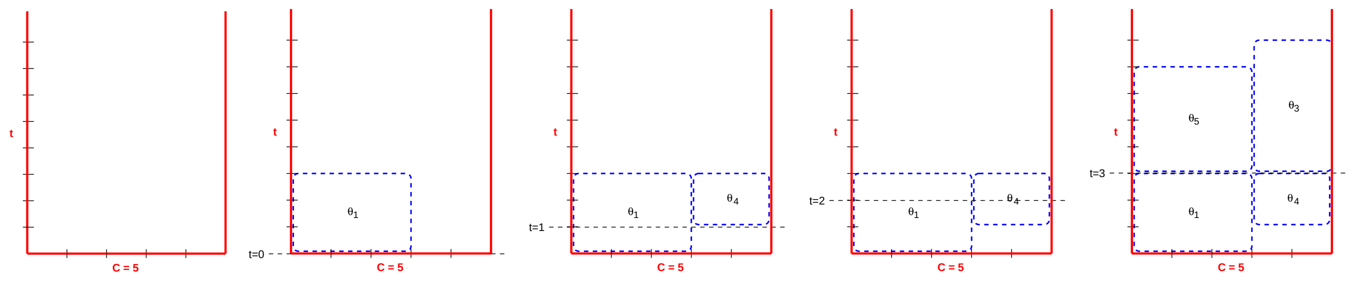
At time $t = 1$, users 3 and 4 submit their bids, and their

TABLE 2: User bids

$\hat{\theta}_i$	$\hat{\sigma}_{i1}$	$\hat{a}_i$	$\hat{l}_i$	$\hat{d}_i$	$\hat{b}_i$	$\hat{\delta}_i$	f_i
$\hat{\theta}_1$	3	0	3	5	5	2	0.56
$\hat{\theta}_2$	3	0	3	4	4	1	0.44
$\hat{\theta}_3$	2	1	5	8	6	3	0.60
$\hat{\theta}_4$	2	1	2	5	3	3	0.75
$\hat{\theta}_5$	3	3	4	9	8	5	0.67
$\hat{\theta}_6$	3	3	6	10	9	4	0.50

TABLE 3: Execution of OVMAP

t	$t=0_-$	$t=0$	$t=1$	$t=2$	$t=3$
$\mathcal{Q}^t$	$\emptyset$	$\{\hat\theta_1,\hat\theta_2\}$	$\{\hat\theta_4,\hat\theta_3,\hat\theta_2\}$	$\{\hat\theta_3\}$	$\{\hat\theta_5,\hat\theta_3,\hat\theta_6\}$
$\tilde{\mathcal{Q}}^t$	$\emptyset$	$\{\hat\theta_1\}$	$\{\hat\theta_4,\hat\theta_1\}$	$\{\hat\theta_4,\hat\theta_1\}$	$\{\hat\theta_5,\hat\theta_3\}$
C_1^t	5	2	0	0	0
$\mathcal{A}$	$\emptyset$	$\{(\hat\theta_1,0)\}$	$\{(\hat\theta_1,0),(\hat\theta_4,1)\}$	$\{(\hat\theta_1,0),(\hat\theta_4,1)\}$	$\{(\hat\theta_1,0),(\hat\theta_4,1),(\hat\theta_5,3),(\hat\theta_3,3)\}$
$\mathcal{P}$	$\emptyset$	$(4,-)$	$(4,-,-,2.4)$	$(4,0,-,2.4)$	$(4,0,0,2.4,6,-)$



(a) Initial state (b) $t=0$ (c) $t=1$ (d) $t=2$ (e) $t=3$

Fig. 1: Execution of OVMAP

bids are included in the set $\mathcal{Q}^t$. User 4 has the highest bid density (i.e., $f_4 > f_3 > f_2$) and $\hat\sigma_{41} \leq C_1^t$, therefore user 4 obtains the requested bundle. At this time, should user 4 not participate, user 3 would have received her requested bundle, therefore $(\hat\theta_4, 2.4)$ is added to the set $\mathcal{P}$, where the payment of 2.4 is the product $f_3 \cdot \hat l_4 \cdot \hat\sigma_{41}$. The payment for user 1 does not change, since user 2 would still obtain her required bundle should user 1 not participate. Figure 1c shows the allocation.

At $t = 2$, $C_1^t = 0$, thus OVMAP is not invoked. However, user 2's deadline for allocation $\delta_2 = 1$ has passed and she leaves the system. Since user 2 did not obtained her bundle, her final payment is zero. Figure 1d shows the allocation of resources at $t = 2$.

At $t = 3$, both users 1 and 4 complete their jobs and bids $\hat\theta_5$ and $\hat\theta_6$ are submitted. User 3's request is still not allocated. According to the ordering on the bid densities, users 5 and 3 obtain their requested bundles. User 3's payment is zero, since the remaining user 6 would not obtain her bundle even if user 3 would have not participated. In a scenario with reserve price, user 3's payment will be set to the reserve price. Since $\delta_1 = 2$ has passed, user 1's payment will not change. We show the outcome in Figure 1e. The process continues like this, as more users submit their requests.

To show the importance of incentive compatibility, we consider the following small example with two users, i and j, arriving at the same time and competing for a unit of resource. The true valuations of users i and j for the unit of resource are \$5 and \$3, respectively. We consider a scenario in which the cloud provider implements a fist-price type auction to allocate the resources. That is, the user with the highest bid wins and pays the price she bids. If the users bid truthfully (bid the same as their valuations), user i wins and pays \$5. The revenue of the provider is \$5. Since user i wants to maximize her utility (decrease her payment), she may misreport. If she submits \$3.01 as her bid, she still wins, and she pays \$3.01. However, the revenue of the provider reduces from \$5 to \$3.01. As a result, user i can benefit by missreporting, thus obtaining a higher utility. If user i submits \$2 as her bid, she will not win. In this scenario, a user needs to know how other users bid in order to strategize on how to bid to maximize her utility. Now we consider a scenario using our proposed mechanism. In the case that users i and j submit their actual true bids, our mechanism selects user i as the winning user, and charge her \$3 based on the critical payment (bid of the losing user). The revenue of the provider is \$3. If user i misreports and submits \$4 as her bid, she still wins. However, she still pays \$3, and the revenue of the provider does not change. As a result, user i cannot benefit by missreporting and change the provider revenue. This scenario shows that a user cannot change the revenue of the cloud provider by missreporting her true valuations. This leads to more stable revenues for the cloud provider. Another benefit is that the user does not need to strategize since she will maximize her utility only by reporting her true valuation.

4.5 Properties of OVMAP

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

Theorem 1: OVMAP mechanism is individually rational.

Proof: 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. This can be easily seen from the structure of the OVMAP mechanism. In line 15 of Algorithm 4, the payment for user i is set to $\mathcal{P}_i = f_q \cdot \hat l_i \cdot \prod_{r\in\mathcal{R}} \sigma_{ir}$, where user q is the user who would have won if user i did not participate. Since user q

TABLE 4: Statistics of workload logs.

Logfile	Avg jobs per hour	Range of CPU	Range of memory (MB)	Range of Storage (MB)	Available CPUs	Memory Capacity (MB)	Storage Capacity (MB)	Avg CPU per job	Avg memory per job (MB)	Avg storage per job (MB)
GWA-T-1 DAS-2	81	[1-128]	[1-4,295]	[10-51,070]	50	100	100	4.37	46.96	43.95
GWA-T-3 NorduGrid	34	1	[1-2,147]	[10-1,053,072]	24	1,400	50,000	1	595.6	93,888.77
GWA-T-4 AuverGrid	33	1	[1.7-3,668]	[10-259,316]	7	8,800	640,000	1	374.3653	27,805.86
GWA-T-10 SHARCNET	147	[1-3000]	[1-32,021]	[10-2,087,029]	85	2,000	1,000	2.9	94.49	39.43
METACENTRUM-2009-2	42	[1-60]	[1-61,538]	[10-2,592,130]	44	100	20,000	1.55	325.14	21,189.11
PIK-IPLEX-2009-1	36	[1-2560]	[1-29,360]	[10-4,815,007]	88	89,000	4,700	12.15	3,528.22	18,716.06

appears after user i in the decreasing order of the density metric, we have, $f_q \leq f_i$, and thus, OVMAP-PAY always computes a payment $\mathcal{P}_i \leq b_i$. As a result, the utility of user i (i.e., $u_i = b_i - \mathcal{P}_i \geq 0$) is non-negative, and she never incurs a loss. In addition, a truthful user who does not win is not incurring a loss since she obtains 0 utility. This proves the individual-rationality of OVMAP mechanism. □

We now prove that the OVMAP mechanism is incentive-compatible. In order to prove that the mechanism is incentive-compatible, we need to show that the allocation algorithm is monotone, and the payment function is based on the critical payment.

Theorem 2: OVMAP mechanism is incentive-compatible.

Proof: We first show that the allocation algorithm OVMAP-ALLOC is monotone. If user i wins by reporting $\hat{\theta}_i$, then she will also win if she reports a more preferred request $\hat{\theta}'_i \geq \hat{\theta}_i$. Clearly, if user i reports $\hat{b}'_i \geq \hat{b}_i$, her bid $\hat{\theta}'_i$ will be allocated if $\hat{\theta}_i$ is also allocated. Similarly, if a user gets the allocation by reporting $\hat{d}_i$, she will also get it by reporting $\hat{d}'_i \geq \hat{d}_i$. Similar reasoning applies for the other parameters in the request of the user.

We now prove that the payment function implemented by OVMAP-PAY is based on the critical payment. In doing so, we need to show that $\mathcal{P}_i$ determined by OVMAP-PAY is the minimum value that user i must report to get the allocation. User i's payment is $\mathcal{P}_i = f_q \cdot \hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}$ (line 15), where q is the index of user q appearing after user i based on the non-increasing order of the bid density (line 11), and she would have won if user i did not participate. We consider that user i submits a lower value $\hat{b}'_i < \mathcal{P}_i$. User i's new bid density is $f'_i = \frac{\hat{b}'_i}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}} < \frac{\mathcal{P}_i}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}}$. By replacing $\mathcal{P}_i$, we have $f'_i < \frac{f_q \cdot \hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}}$. Thus, we have $f'_i < f_q$, that is, user i will appear after user q who did not win. As a result, if user i reports a bid below the minimum value (i.e., $\mathcal{P}_i$), she loses; otherwise she wins. This unique value is the critical payment for user i. This, together with the fact that losing users pay zero, show that the payment function implemented by OVMAP is the critical payment.

Since the payment is the critical payment and the allocation function is monotone, it follows from Parkes [9] that OVMAP is incentive-compatible. □

Theorem 3: The time complexity of OVMAP mechanism is polynomial.

Proof: The time complexity of OVMAP-ALLOC is $O(N(\log N + MR))$. This is because sorting the requests requires $O(N \log N)$, while checking the feasibility of the allocation for each user requires $O(NMR)$. Similar reasoning applies to OVMAP-PAY. As a result, the time complexity of OVMAP mechanism is polynomial. □

5 Experimental Results

We perform extensive experiments with real workload data in order to investigate the properties of our proposed online mechanism, and the offline optimal VCG-VMAP mechanism. For the VCG-VMAP mechanism, we use the CPLEX 12 solver [45] to solve the VMAP problem optimally. The data that drives our experiments consists of six workload logs from the Grid Workloads Archive [46] and the Parallel Workloads Archive [47]. 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 university grid system. In this section, we describe the experimental setup and analyze the experimental results.

5.1 Experimental Setup

Since real users request data have not been publicly released by cloud providers yet, we rely on well studied and standardized workloads from both the Grid Workloads Archive [46] and the Parallel Workloads Archive [47]. We selected the following six logs based on the availability of both recorded CPU and memory requests/usage: i) DAS-2 traces from a research grid at the Advanced School for Computing and Imaging in Netherlands; ii) NorduGrid traces from the NorduGrid system; iii) AuverGrid traces from the AuverGrid system; iv) SHARCNET traces from SHARCNET clusters installed at several academic institutions in Ontario, Canada. v) MetaCentrum from the national grid of the Czech republic; vi) IBM iDataPlex Cluster log from the Potsdam Institute for Climate Impact Research (PIK) in Germany. In our experiments, a user request is represented by a job in a log. We present statistics of the logs in Table 4.

Each log represents a series of requests, where the users arrive over time, and they can submit their requests to a cloud provider. The following fields of the log files are extracted to represent different features of the users' requests. (1) JobID: the user's identifier; (2)

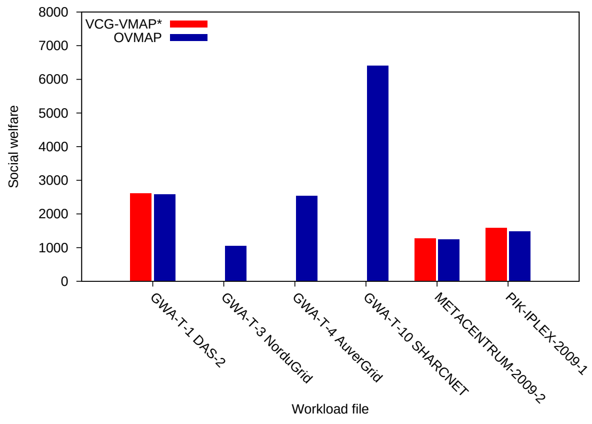


Fig. 2: OVMAP vs. VCG-VMAP: Welfare
(*VCG-VMAP was not able to determine the allocation for GWA-T-3 NorduGrid, GWA-T-4 AuverGrid, and GWA-T-10 SHARCNET in feasible time, and thus, there are no bars in the plots for those cases)

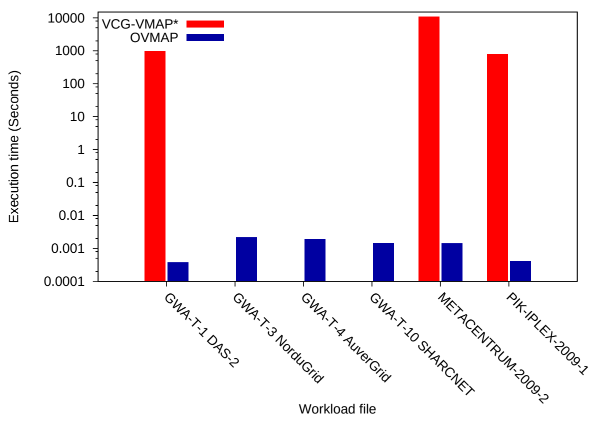


Fig. 3: OVMAP vs. VCG-VMAP: Execution time
(*see Fig.2 note on VCG-VMAP)

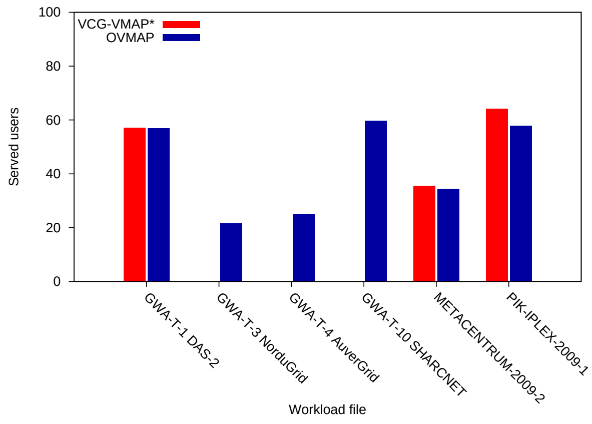


Fig. 4: OVMAP vs. VCG-VMAP: Users served
(*see Fig.2 note on VCG-VMAP)

SubmitTime: the arrival time of the request; (3) RunTime: the amount time for which the requested bundle must be allocated; (4) ReqNProcs: the requested number of processors; (5) Used Memory: the requested amount of memory. Since the amount of storage usage was not recorded in the workloads, we generate the requested storage as shown in Table 4. We consider these resource usage as values for the requested σ_{ir}, the amount of each resource of type r requested by user i, where i is a job in a log, and r is a type of resource. We generate a random number b_i between 1 and 10 to represent user i's bid. For a deadline of a request, we choose a random number between 3 to 6 times the job's runtime. We use the job's runtime as the requested length of the job. We select 100 hours of the logs containing 706, 842, 1523, 1865, 677, and 416 requests for the afore-mentioned logs, respectively.

5.2 Analysis of Results

We compare the performance of OVMAP and VCG-VMAP for different workloads. For each workload, we record the execution time, the welfare, the revenue, the percent of users served and the utilization for each mechanism. Users served is the number of users who received their requests for their entire requested time. The utilization of each resource type is calculated as the percentage of allocated resource out of the total capacity of that resource over the entire time. We now present the results obtained by OVMAP for the selected logs.

We analyze the performance of OVMAP and VCG-VMAP in terms of welfare, execution time, the percent of users served, resource utilization, and revenue. In this case, users are requesting a heterogeneous set of VM instances for a length of time. The optimal mechanism, VCG-VMAP, could not find the solutions even after 72 hours for three out of the six logs. This is due to the fact that the problem becomes more complex for different job lengths, higher number of requests, and greater available capacity. Fig. 2 shows the welfare achieved by the mechanisms. VCG-VMAP is not able to determine the allocation for GWA-T-3 NorduGrid, GWA-T-4 AuverGrid, and GWA-T-10 SHARCNET in feasible time, and thus, there are no bars in the plots for those cases. For the remaining logs, the results show that OVMAP obtains a welfare very close to that obtained by the optimal VCG-VMAP mechanism. On average the optimality gap is 3.7%. For example, OVMAP and VCG-VMAP obtain welfares of 1233.57 and 1274.25 for the METACENTRUM-2009-2 log, respectively, leading to a 3.19% optimality gap. Such results are very promising given the fact that OVMAP is an online mechanism which does not have any information about future demand. However, VCG-VMAP is an offline mechanism and has all the information available a priori. Fig. 3 shows the execution times of the mechanisms on a logarithmic scale. As we expected from the time complexity of the mechanism, the execution time of OVMAP is very small. However, the execution time of the optimal offline mechanism, VCG-VMAP, is more than six order of magnitudes greater than that of OVMAP for each of the logs. Note that the online setting requires mechanisms with very small execution times. Since OVMAP obtains close to optimal welfare and is

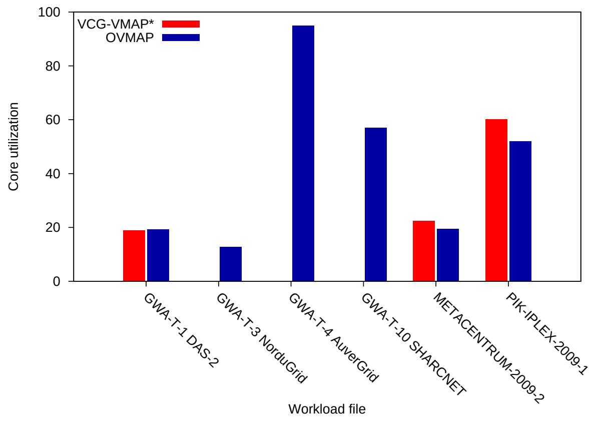


Fig. 5: OVMAP vs. VCG-VMAP: Core utilization
(*see Fig.2 note on VCG-VMAP)

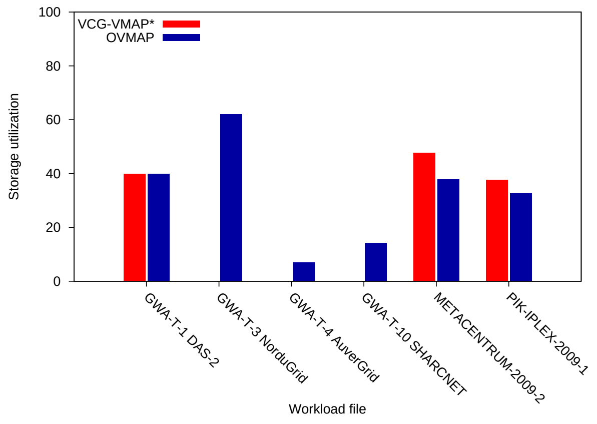


Fig. 7: OVMAP vs. VCG-VMAP: Storage utilization
(*see Fig.2 note on VCG-VMAP)

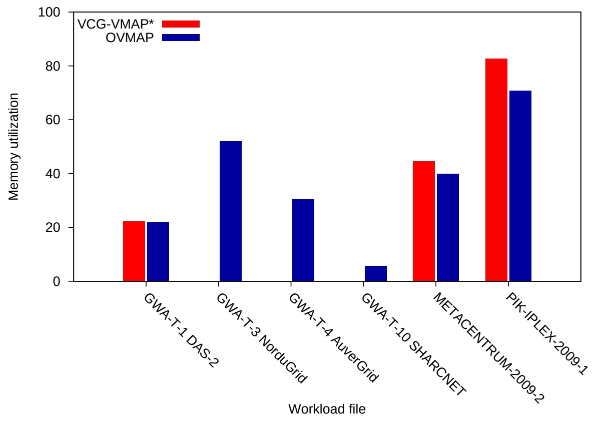


Fig. 6: OVMAP vs. VCG-VMAP: Memory utilization
(*see Fig.2 note on VCG-VMAP)

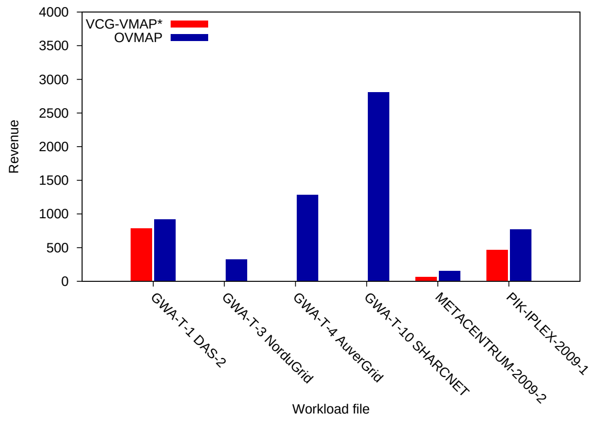


Fig. 8: OVMAP vs. VCG-VMAP: Revenue
(*see Fig.2 note on VCG-VMAP)

very fast it is very suitable for solving the allocation and pricing problem is online settings. We measured the execution time of the mechanism for processing each of the requests from the traces and calculated its average. The average execution time of the mechanism for processing a request is 1.27 microseconds. This shows that the mechanism is very fast and can be used in online settings. Since for each request the system will need to instantiate a VM to serve the request, the total time required to process and serve a request is given by the sum of the time to instantiate a VM and the time to run the mechanism. Since the time to instantiate a VM is in the order of tens of seconds, the contribution of the mechanism to the total time to process a request is negligible. The total time to process a request will basically impose an upper bound on the arrival rate of requests. However, since not all of the requests will be allocated (no VM instantiated) the upper bound on the arrival rate will be much grater than the bound imposed by the total time to process a request.

Fig. 4 shows the percentage of served users for the mechanisms. The percentage of served users obtained by OVMAP is very close to that of VCG-VMAP due to its close to optimal solution. This is due to the fact that the solution determined by OVMAP is very close to the optimal solution. Figs. 5 to 7 show the utilization of cores, memory and storage, respectively. Note that a higher utilization does not show the effectiveness of the mechanisms. The objective of all the mechanisms is maximizing the welfare not the utilization of the resources. Fig. 8 shows the revenue achieved by the cloud provider when using the mechanisms. OVMAP is able to obtain a higher revenue than that of the VCG-VMAP. For example, for log GWA-T-1 DAS-2, OVMAP and VCG-VMAP obtain total revenues of \$916.25 and \$783.67, respectively, corresponding to 16.91% higher revenue using OVMAP. Note that the VCG-VMAP is optimal in terms of welfare and not the revenue. This is due to the fact that VCG-VMAP fulfills more requests (i.e., more users are allocated), that is, it accepts more bids. Accepting more bids, reduces the price charged to users and implicitly the revenue.

From the above results we can conclude that OVMAP decides the allocation and pricing much faster than VCG-VMAP and achieves a welfare closer to the optimal. As a result, OVMAP is suitable for making allocation decisions and price determination in real-time.

6 Conclusion

The nature and dynamics of users' demand in cloud markets necessitates designing online mechanisms. Such online mechanisms make no assumptions about future demands. In this paper, we proposed an online incentive-compatible mechanism, OVMAP, for VM allocation and pricing in clouds. The OVMAP mechanism not only provisions and allocates resources dynamically, but also determines the price that users must pay for their requested VMs. Our proposed mechanism provides incentives to the users to reveal their actual requests facilitating a healthy competition among users. We proved that OVMAP is individually-rational and incentive-compatible. In addition, we proposed an optimal offline mechanism in order to compare its performance with our proposed online mechanism. The experimental results showed that the proposed online mechanism obtains better revenue and decides the allocation much faster than the offline mechanism. For future work, we plan to design and investigate new online mechanisms in federated cloud settings.

Acknowledgment

We would like to thank the Associate Editor and the reviewers for their helpful and constructive suggestions, which considerably improved the quality of the paper. This paper is a revised and extended version of [48] presented at the 7th IEEE Intl. Conference on Cloud Computing (CLOUD 2014). This research was supported in part by NSF grants DGE-0654014 and CNS-1116787.

References

[1] R. Buyya, C. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *Proc. 10th IEEE Intl. Conf. on High Perf. Comp. and Comm.*, 2008, pp. 5–13.

[2] Amazon EC2 Instance Types. [Online]. Available: http://aws.amazon.com/ec2/instance-types/

[3] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in grid computing," *Concurrency and computation: practice and experience*, vol. 14, no. 13-15, pp. 1507–1542, 2002.

[4] N. Nisan and A. Ronen, "Algorithmic mechanism design," in *Proc. of the 31st annual ACM Symp. on Theory of computing*, 1999, pp. 129–140.

[5] J. D. Hartline and B. Lucier, "Bayesian algorithmic mechanism design," in *Proc. of the 42nd ACM Symposium on Theory of computing*, 2010, pp. 301–310.

[6] J. Feigenbaum, M. Schapira, and S. Shenker, "Distributed algorithmic mechanism design," *Algorithmic Game Theory*, pp. 363–384, 2007.

[7] S. Dobzinski and S. Dughmi, "On the power of randomization in algorithmic mechanism design," in *Proc. of the 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2009, pp. 505–514.

[8] E. J. Friedman and D. C. Parkes, "Pricing wifi at starbucks: issues in online mechanism design," in *Proceedings of the 4th ACM conference on Electronic commerce*, 2003, pp. 240–241.

[9] D. C. Parkes, "Online mechanisms," in *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, Éva Tardos, and V. V. Vazirani, Eds. Cambridge University Press, 2007.

[10] A. Gershkov and B. Moldovanu, "Efficient sequential assignment with incomplete information," *Games and Economic Behavior*, vol. 68, no. 1, pp. 144–154, 2010.

[11] D. C. Parkes and S. Singh, "An MDP-based approach to online mechanism design," in *Proc. 17th Annual Conf. on Neural Information Processing Systems*, 2003.

[12] M. T. Hajiaghayi, R. Kleinberg, and D. C. Parkes, "Adaptive limited-supply online auctions," in *Proc. of the 5th ACM conference on Electronic commerce*, 2004, pp. 71–80.

[13] M. Hajiaghayi, R. Kleinberg, M. Mahdian, and D. C. Parkes, "Online auctions with re-usable goods," in *Proc. 6th ACM Conf. on Electronic Commerce*, 2005, pp. 165–174.

[14] R. Porter, "Mechanism design for online real-time scheduling," in *Proc. 5th ACM Conf. on Electronic Commerce*, 2004, pp. 61–70.

[15] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," *Proceedings of the IEEE*, vol. 102, no. 1, pp. 11–31, 2014.

[16] I. Jangjaimon and N.-F. Tzeng, "Effective cost reduction for elastic clouds under spot instance pricing through adaptive checkpointing," *IEEE Transactions on Computers*, p. 1, 2014.

[17] J.-J. Kuo, H.-H. Yang, and M.-J. Tsai, "Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems," in *Proc. of IEEE INFOCOM*, 2014.

[18] L. Wang, F. Zhang, A. V. Vasilakos, C. Hou, and Z. Liu, "Joint virtual machine assignment and traffic engineering for green data center networks," *SIGMETRICS Performance Evaluation Review*, vol. 41, no. 3, pp. 107–112, 2013.

[19] Z. Xiao, Q. Chen, and H. Luo, "Automatic scaling of internet applications for cloud computing services," *IEEE Transactions on Computers*, 2014.

[20] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[21] C. Papagianni, V. Maglaris, C. Cervell, A. Leivadeas, S. Papavassiliou *et al.*, "On the optimal allocation of virtual resources in cloud computing networks," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1060–1071, 2013.

[22] T. Ghazar and N. Samaan, "Pricing utility-based virtual networks," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 119–132, June 2013.

[23] M. Guazzone, C. Anglano, and M. Canonico, "Energy-efficient resource management for cloud computing infrastructures," in *Proc. of the 3rd IEEE Intl. Conf. on Cloud Computing Technology and Science*, 2011, pp. 424–431.

[24] M. HoseinyFarahabady, Y. C. Lee, and A. Y. Zomaya, "Randomized approximation scheme for resource allocation in hybrid-cloud environment," *The Journal of Supercomputing*, pp. 1–17, 2014.

[25] L. M. Leslie, Y. C. Lee, P. Lu, and A. Y. Zomaya, "Exploiting performance and cost diversity in the cloud," in *Proc. of the 6th IEEE International Conference on Cloud Computing*, 2013, pp. 107–114.

[26] J. Cao, K. Hwang, K. Li, and A. Y. Zomaya, "Optimal multiserver configuration for profit maximization in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1087–1096, 2013.

[27] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," *The Journal of Supercomputing*, vol. 54, no. 2, pp. 252–269, 2010.

[28] Y. Feng, B. Li, and B. Li, "Price competition in an oligopoly market with multiple iaas cloud providers," *IEEE Transactions on Computers*, vol. 63, no. 1, pp. 59–73, 2014.

[29] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: A randomized auction approach," in *Proc. of IEEE INFOCOM*, 2014.

[30] A. Prasad and S. Rao, "A mechanism design approach to resource procurement in cloud computing," *IEEE Transactions on Computers*, 2014.

[31] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *Proc. of IEEE INFOCOM*, 2014.

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

[33] M. Nejad, L. Mashayekhy, and D. Grosu, "Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 2, pp. 594 – 603, February 2015.

[34] L. Mashayekhy, M. Nejad, and D. Grosu, "A PTAS mechanism for provisioning and allocation of heterogeneous cloud resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2014.

[35] Y. Hua, X. Liu, and H. Jiang, "Antelope: A semantic-aware data cube scheme for cloud data center networks," *IEEE Transactions on Computers*, 2014.

[36] L. Zhang, Z. Li, C. Wu, and M. Chen, "Online algorithms for uploading deferrable big data to the cloud," in *Proc. of IEEE INFOCOM*, 2014.

[37] Z. Abbasi, M. Pore, and S. K. Gupta, "Online server and workload management for joint optimization of electricity cost and carbon footprint across data centers," in *Proc. 28th IEEE Intl. Symp. on Parallel & Dist. Proc.*, 2014.

[38] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Transactions on Computers*, 2014.

[39] J. Zhao, H. Li, C. Wu, Z. Li, Z. Zhang, and F. Lau, "Dynamic pricing and profit maximization for clouds with geo-distributed datacenters," in *Proc. of IEEE INFOCOM*, 2014.

[40] 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. of IEEE INFOCOM*, 2013.

[41] S. Zaman and D. Grosu, "An online mechanism for dynamic vm provisioning and allocation in clouds," in *Proc. of the 5th IEEE Intl. Conf. on Cloud Computing*, 2012, pp. 253–260.

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

[43] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004.

[44] A. Mu'alem and N. Nisan, "Truthful approximation mechanisms for restricted combinatorial auctions," in *Proc. of the 18th national conf. on Artificial intelligence*. American Association for Artificial Intelligence, 2002, pp. 379–384.

[45] IBM ILOG CPLEX V12.1 user's manual. [Online]. Available: ftp://ftp.software.ibm.com/software/websphere/ilog/docs/optimization/cplex/psusrmancplex.pdf

[46] Grid workloads archive. [Online]. Available: http://gwa.ewi.tudelft.nl

[47] Parallel workloads archive. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/

[48] L. Mashayekhy, M. M. Nejad, D. Grosu, and A. V. Vasilakos, "Incentive-compatible online mechanisms for resource provisioning and allocation in clouds," in *Proc. of the 7th IEEE Intl. Conf. on Cloud Computing*, 2014.

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, and ICPP. 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 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. 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.

Athanasios V. Vasilakos is currently a professor with the Lulea University of Technology, Sweden. He has authored or co-authored over 200 technical papers in major international journals and conferences. He is author/co-author of five books and 20 book chapters in the area of communications. He has served as general chair/technical program committee chair for many international conferences. He served or is serving as an editor and/or guest editor for many technical journals, such as the IEEE Transactions on Network and Service Management, the IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, the IEEE Transactions on Information Technology in Biomedicine, the IEEE Transactions on Computers, the ACM Transactions on Autonomous and Adaptive Systems, IEEE JSAC special issues of May 2009, January 2011, and March 2011, IEEE Communications Magazine, ACM/Springer Wireless Networks (WINET), and ACM/Springer Mobile Networks and Applications (MONET). He is founding editor-in-chief of the International Journal of Adaptive and Autonomous Communications Systems (IJAACS) and the International Journal of Arts and Technology (IJART). He is general chair of the Council of Computing of the European Alliances for Innovation.