Truthful Greedy Mechanisms for Dynamic Virtual Machine Provisioning and Allocation in Clouds

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

Abstract—A major challenging problem for cloud providers is designing efficient mechanisms for Virtual Machine (VM) provisioning and allocation. Such mechanisms enable the cloud providers to effectively utilize their available resources and obtain higher profits. Recently, cloud providers have introduced auction-based models for VM provisioning and allocation which allow users to submit bids for their requested VMs. We formulate the dynamic VM provisioning and allocation problem for the auction-based model as an integer program considering multiple types of resources. We then design truthful greedy and optimal mechanisms for the problem such that the cloud provider provisions VMs based on the requests of the winning users and determines their payments. We show that the proposed mechanisms are truthful, that is, the users do not have incentives to manipulate the system by lying about their requested bundles of VM instances and their valuations. We perform extensive experiments using real workload traces in order to investigate the performance of the proposed mechanisms. Our proposed mechanisms achieve promising results in terms of revenue for the cloud provider.

Index Terms—cloud computing; truthful mechanism; virtual machine provisioning; dynamic resource allocation; greedy heuristics.

1 Introduction

The number of enterprises and individuals that are outsourcing their workloads to cloud providers has increased rapidly in recent years. Cloud providers form a large pool of abstracted, virtualized, and dynamically scalable resources allocated to users based on a pay-as-you-go model. These resources are provided as three different types of services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS provides CPUs, storage, networks and other low level resources, PaaS provides programming interfaces, and SaaS provides already created applications. In this paper, we focus on IaaS where cloud providers offer different types of resources in the form of VM instances. IaaS providers such as Microsoft Azure [1] and Amazon Elastic Compute Cloud (Amazon EC2) [2] offer four types of VM instances: small (S), medium (M), large (L), and extra large (XL).

Cloud providers face many decision problems when offering IaaS to their customers. One of the major decision problems is how to provision and allocate VM instances. Cloud providers provision their resources either statically or dynamically, and then allocate them in the form of VM instances to their customers. In the case of *static provisioning*, the cloud provider pre-provisions a set of VM instances without considering the current demand from the users, while in the case of *dynamic provisioning*, the cloud provider provisions the resources by taking into account the current users' demand. Due to the variable load demand, dynamic provisioning leads to a more efficient resource utilization and ultimately to higher revenues for the cloud provider. The aim of this study is to facilitate dynamic provisioning of multiple types of resources based on the users' requests.

To sell the VM instances to users, cloud providers can employ fixed-price and auction-based models. In the fixed-price model, the price of each type of VM instance is fixed and pre-determined by the cloud provider, while in the auction-based model, each user bids for a subset of available VM instances (bundle) and an auction mechanism decides the price and the allocation. In this study, we consider the design of mechanisms for auction-based settings. In the auction-based models, users can obtain their requested resources at lower prices than in the case of the fixed-price models. Also, the cloud providers can increase their profit by allowing users to bid on unutilized capacity. An example of such auction-based mechanism is the spot market introduced by Amazon [2]. Such mechanisms are usually executed over short time-windows (e.g., every hour) to efficiently provision the unutilized resources of the cloud provider. 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. In addition, all winning users in the Amazon spot market pay the same (per unit) price. In our setting, we allow users to request bundles of VM instances. We consider a set of users and a set of items (VM instances), where each user bids for a subset of items (bundle). Since several VM instances of the same type are available to users, the problem can be viewed as a multi-unit combinatorial auction. Each user has a private value (private *type*) for her requested bundle. In our model, the users are single minded, that means each

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

user is either assigned her entire requested bundle of VM instances and she pays for it, or she does not obtain any bundle and pays nothing. The users are also selfish in the sense that they want to maximize their own utility. It may be beneficial for them to manipulate the system by declaring a false type (i.e., different bundles or bids from their actual request).

One of the key properties of a provisioning and allocation mechanism is to give incentives to users so that they reveal their true valuations for the bundles. In general, greedy algorithms do not necessarily satisfy the properties required to achieve truthfulness (also called incentive-compatibility or strategy-proofness [3]) and they need to be specifically designed to satisfy those properties. Our goal is to design truthful greedy mechanisms that solve the VM provisioning and allocation problem in the presence of multiple types of resources (e.g., cores, memory, storage, etc.). The mechanisms allocate resources to the users such that the social welfare (i.e., the sum of users' valuations for the requested bundles of VMs) is maximized.

1.1 Our Contribution

We address the problem of VM provisioning and allocation in clouds in the presence of multiple types of resources. To the best of our knowledge, this is the first study proposing truthful mechanisms for VM provisioning and allocation in clouds that take into account the heterogeneity and the scarcity of the cloud resources. We design a truthful optimal mechanism and a family of truthful greedy mechanisms for VM provisioning and allocation that give incentives to the users to reveal their true valuations for their requested bundles of VM instances. Our proposed mechanisms consist of determining the VM provisioning and allocation and the payments for each user. Our proposed greedy mechanisms provide very fast solutions making them suitable for execution in short time-window auctions. In addition, we determine the approximation ratio of the proposed mechanisms, guaranteeing a bound for the obtained solutions. We design truthful greedy mechanisms in spite the fact that greedy algorithms, in general, do not necessarily satisfy the properties required to guarantee truthfulness. In doing so, the allocation and payment determination of the proposed mechanisms are designed to satisfy the truthfulness property. Our proposed mechanisms allow dynamic provisioning of VMs, and do not require pre-provisioning the VMs. As a result, cloud providers can fulfill dynamic market demands efficiently. A key property of our proposed mechanisms is the consideration of multiple types of resources when provisioning the VMs, which is the case in real cloud settings. Previous work considered only one type of resource and did not take into account the scarcity of each resource type when making the VM instance provisioning an allocation decisions. The novelty of our proposed mechanisms consists of taking these into account to improve the allocation decisions. We perform extensive experiments that show that our proposed greedy mechanisms are able to find near optimal allocations while satisfying the truthfulness property.

1.2 Related Work

Several researchers investigated various resource allocation problems in clouds by employing game theory. Wei *et al.* [4] formulated the resource allocation problem as a task scheduling problem with QoS constraints. They proposed a game-theoretic approximated solution. However, there is an assumption that the cloud provider knows the execution time of each subtask, which is unrealistic in cloud environments. Jain *et al.* [5] designed an efficient truthful-in-expectation mechanism for resource allocation in clouds where only one type of resource was considered. Kong *et al.* [6] designed a stochastic mechanism to allocate resources among selfish VMs in a non-cooperative cloud environment. Wang *et al.* [7] showed that system heterogeneity plays an important role in determining the dynamics of truthful mechanisms. Our proposed mechanisms take into account the heterogeneity of the systems and that of user requests when making allocation decisions. Ardagna *et al.* [8] modeled the service provisioning problem as a generalized Nash game and proved the existence of equilibria for such game. In their model, the objective of the SaaS is to maximize its revenue satisfying the service level agreement, while the objective of the IaaS is to maximize the profit by determining the spot instances price. Di Valerio *et al.* [9] formulated the service provisioning problem as a Stackelberg game, and computed the equilibrium price and allocation strategy by solving the associated optimization problem. However, both studies considered only one type of VM instances, thus, the problem they solved is a one dimensional provisioning problem.

Mechanism design theory has been employed in designing truthful allocation mechanisms in several areas. In particular, there is a large body of work in spectrum auctions, where a government or a primary license holder sells the right to use a specific frequency band in a specific area via auction-based mechanisms (e.g., [10], [11], [12], [13]). In these studies, only one type of resource (i.e., the spectrum) is available for allocation. However, in this paper, we consider several types of resources (e.g., core, memory, storage), and thus the mechanisms proposed in the above studies cannot be used in our context. Zhou *et al.* [10] proposed a truthful mechanism, that assumes the existence of k uniform channels that can be spatially reused (i.e., a channel can be allocated to more than one user simultaneously). Their greedy mechanism sorts the bidders in descending order of their bids. Wu and Vaidya [13] extended the study of Zhou *et al.* by proposing a truthful mechanism considering grouping the users based on their spatial conflicts. Their greedy mechanism is based on the ordering of the groups'

bids. However, VM instances cannot be simultaneously assigned to the users and thus, their mechanism cannot be used to solve the VM allocation problem. The closest work to ours in the spectrum allocation area is by Jia *et al.* [14] who proposed truthful mechanisms for a secondary spectrum market. The authors considered K uniform channels covering a certain region that is partitioned into small cells. This problem considers several cells available which in some sense correspond to several types of VMs in our study. However, in each cell a fixed number of uniform channels are available to be sold, whereas, in our case, each VM instance is composed of several types of heterogeneous resources. Furthermore, the mechanism proposed by Jia *et al.* [14] incorporates a simple greedy metric for ordering the users that is based on the ratio of their bids to the number of requested channels. However, our proposed mechanisms incorporate bid density metrics that not only consider the structure of VMs (i.e., the multiple resources), but also take into account the scarcity of resources. In addition, we do not limit the number of available VMs for each type of VM, and we allow dynamic provisioning of VMs.

The design of truthful mechanisms for resource allocation in clouds has been investigated by Zaman and Grosu [15], [16]. They proposed a combinatorial auction-based mechanism, CA-GREEDY, to allocate VM instances in clouds [16]. They showed that CA-GREEDY can efficiently allocate VM instances in clouds generating higher revenue than the currently used fixed price mechanisms. However, CA-GREEDY requires that the VMs are provisioned in advance, that is, it requires static provisioning. They extended their work to dynamic scenarios by proposing a mechanism called CA-PROVISION [15]. CA-PROVISION selects the set of VM instances in a dynamic fashion which reflects the market demand at the time when the mechanism is executed. However, these mechanisms do not consider several types of resources. Their proposed mechanisms only consider computational resources (i.e., cores), which is only one of the dimensions in our proposed model. In addition to this, our proposed mechanisms consider the scarcity of the resources when making provisioning and allocation decisions.

In Appendix A of the supplemental material we provide a discussion of additional related work on mechanism design, combinatorial auctions, and other methodologies for solving the VM provisioning and allocation problem.

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. In Section 3, we introduce the basic concepts of mechanism design and present the design of an optimal mechanism for VM provisioning and allocation. In Section 4, we present the proposed mechanisms and characterize their properties. In Section 5, we evaluate the mechanisms by extensive simulation experiments. In Section 6, we summarize our results and present possible directions for future research.

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

2 VM Provisioning and Allocation Problem

We consider a cloud provider offering R types of resources, $\mathcal{R} = \{1, \ldots, R\}$, to users in the form of VM instances. These types of resources include cores, memory, storage, etc. The cloud provider has restricted capacity, C_r, on each resource $r \in \mathcal{R}$ available for allocation. The cloud provider offers these resources in the form of M types of VMs, $\mathcal{VM} = \{1, \ldots, M\}$, where each VM of type $m \in \mathcal{VM}$ provides a specific amount of each type of resource $r \in \mathcal{R}$. The amount of resources of type r that one VM instance of type m provides is denoted by w_{mr}. As an example, in Table 1, we present the four types of VM instances offered by Amazon EC2 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 Large instance ($m = 3$) by: $w_{11} = 4$, $w_{12} = 7.5$ GB, and $w_{13} = 850$ GB.

We consider a set $\mathcal{U}$ of N users requesting a set of VM instances. User i, $i = 1, \ldots, N$, requests a bundle $S_i =< k_{i1}, k_{i2}, \ldots, k_{iM} >$ 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 values her requested bundle S_i at $v_i(S_i)$, where $v_i(S_i)$ is called the *valuation* of user i for bundle S_i. The valuation represents the maximum price a user is willing to pay for using the requested bundle for a unit of time. Each user can submit her request as a vector specifying the number of VM instances, and her bid. For example, $(< 1, 3, 4, 2 >, \$20)$ represents a user requesting 1 small VM instance, 3 medium VM instances, 4 large VM instances, and 2 extra large VM instances, and her bid is \$20. We denote by V the *social welfare*, which is defined as the sum of users' valuations:

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

where x_i, $i = 1, \ldots, N$, are decision variables defined as follows: $x_i = 1$, if bundle S_i is allocated to user i; and $x_i = 0$, otherwise.

To design incentive-compatible mechanisms, we consider the standard mechanism design objective, that is, maximizing the social welfare [3]. Maximizing social welfare can help a cloud provider increase its revenue

by allocating the VMs to the users who value them the most.

We formulate the problem of VM provisioning and allocation in clouds (VMPAC) as an Integer Program (called VMPAC-IP) as follows:

$$\text{Maximize } V \tag{2}$$

Subject to:

$$\sum_{i\in\mathcal{U}}\sum_{m\in\mathcal{VM}} k_{im}w_{mr}x_i \leq C_r, \forall r \in \mathcal{R} \tag{3}$$

$$x_i = \{0,1\}, \forall i \in \mathcal{U} \tag{4}$$

The solution to this problem is a vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ maximizing the social welfare. Constraints (3) ensure that the allocation of each resource type does not exceed the available capacity of that resource. Constraints (4) represent the integrality requirements for the decision variables. These constraints force the cloud provider to provision the whole bundle of VM instances and to allocate bundles to the selected users. The VMPAC problem is equivalent to the multidimensional knapsack problem (MKP) [17], where the knapsack constraints are the resource capacity constraints and the bundles are the items. The objective is to select a subset of items for the multidimensional knapsack maximizing the total value. As a result, the VMPAC problem is strongly NP-hard.

3 Mechanism Design Framework

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

3.1 Preliminaries

A mechanism $\mathcal{M} = (\mathcal{A}, \mathcal{P})$ consists of an allocation function $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_N)$ and a payment rule $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$. The allocation function determines which users receive their requested bundles, and the payment rule determines the amount that each user must pay.

In our model, there are N users in $\mathcal{U}$, and the type 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 types of all users, and by $\boldsymbol{\theta}_{-i}$, the vector of all types except user i's type (i.e., $\boldsymbol{\theta}_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_N)$). The allocation and payments depend on the users type declarations. The allocation function finds a subset $\mathcal{A}(\boldsymbol{\theta}) \subseteq \mathcal{U}$ of winning users, where $\mathcal{A}_i$ is the allocated bundle of VMs to user i.

The users are assumed to be *single-minded*. That means, user i desires only the requested bundle of VM instances, S_i, and derives a value of b_i if she gets the requested bundle or any superset of it, and zero value, otherwise. Thus, the valuation function for user i is as follows:

$$v_i(\mathcal{A}_i) = \begin{cases} b_i & \text{if } S_i \subseteq \mathcal{A}_i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The bundle of VM instances requested by a single-minded user consists of the minimum amount of resources that the user needs in order to run her job.

User i has a quasi-linear utility function $u_i(\boldsymbol{\theta}) = v_i(\mathcal{A}_i(\boldsymbol{\theta})) - \mathcal{P}_i(\boldsymbol{\theta})$, where $\mathcal{P}_i(\boldsymbol{\theta})$ is the payment for user i that the mechanism calculates based on the payment rule $\mathcal{P}$. Each user's type is private knowledge. The users may declare different types from their true types. We denote by $\hat{\theta}_i = (\hat{S}_i, \hat{b}_i)$ user i's declared type. Note that $\theta_i = (S_i, b_i)$ is user i's true type. The goal of a user is to maximize her utility, and she may manipulate the mechanism by lying about her true type to increase her utility. In our case, since the type of a user is a pair of bundle and value, the user can lie about the value by reporting a higher value in the hope to increase the likelihood of obtaining her requested bundle. These manipulations by the users will lead to inefficient allocation of resources and ultimately will reduce the revenue obtained by the cloud provider. We want to prevent such manipulations by designing truthful mechanisms for solving VMPAC. A mechanism is *truthful* if all users have incentives to reveal their true types.

Definition 1 (Truthfulness): A mechanism $\mathcal{M}$ is *truthful* (also called strategy-proof or incentive compatible [3]) if for every user i, for every type declaration of the other users $\hat{\boldsymbol{\theta}}_{-i}$, a true type declaration θ_i and any other declaration $\hat{\theta}_i$ 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 other words, a mechanism is truthful if truthful reporting is a dominant strategy for the users, that is, the users maximize their utilities by truthful reporting independently of what the other users are reporting. To obtain a truthful mechanism the allocation function $\mathcal{A}$ must be monotone and the payment rule must be based on the critical value [18].

To define monotonicity, we need to introduce a preference relation $\succeq$ on the set of types 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 type $\hat{\theta}'_i$ 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 type, then it also allocates the resources to user i with $\hat{\theta}'_i$, where $\hat{\theta}'_i \succeq \hat{\theta}_i$.

Any winning user who receives her requested bundle by declaring a type $\hat{\theta}_i$ is still wining if she requests a smaller bundle and submits a higher bid.

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

The mechanism $\mathcal{M}$ works as follows. It first receives the declared types (bundles and bids) from each participating user, and then, based on the received types

Algorithm 1 VCG-VMPAC Mechanism

1: **Input:** $\mathbf{C} = (C_1, \ldots, C_R)$; vector of resource capacities
2: {Collect user requests (types).}
3: **for all** $i \in \mathcal{U}$ **do**
4: Collect user type $\hat{\theta}_i = (\hat{S}_i, \hat{b}_i)$ from user i
5: {Allocation.}
6: (V^*, $\mathbf{x}^*$) = Solve IP-VMPAC($\hat{\boldsymbol{\theta}}, \mathbf{C}$)
7: Provisions and allocates VM instances according to $\mathbf{x}^*$.
8: {Payment.}
9: **for all** $i \in \mathcal{U}$ **do**
10: (V'^*, $\mathbf{x}'^*$) = Solve IP-VMPAC($\hat{\boldsymbol{\theta}}_{-i}, \mathbf{C}$)
11: $sum_1 = sum_2 = 0$
12: **for all** $j \in \mathcal{U}, j \neq i$ **do**
13: $sum_1 = sum_1 + \hat{b}_j x'^*_j$
14: $sum_2 = sum_2 + \hat{b}_j x^*_j$
15: $\mathcal{P}_i = sum_1 - sum_2$
16: **Output:** V^*, $\mathbf{x}^*$, $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

determines the allocation using the allocation function $\mathcal{A}$ and the payments using the payment rule $\mathcal{P}$. The payment rule $\mathcal{P}$ is based on the critical value and is defined as follows:

$$\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = \begin{cases} v_i^c & \text{if } i \text{ wins} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where v_i^c is the critical value of user i.

In the next subsection, we design a Vickrey-Clarke-Groves (VCG)-based optimal mechanism that solves the VMPAC problem.

3.2 Truthful Optimal Mechanism

We introduce a VCG-based truthful optimal mechanism that solves the VMPAC problem. A VCG-based mechanism requires an optimal allocation algorithm implementing the allocation function $\mathcal{A}$ [3]. A VCG mechanism [3] is defined as follows.

Definition 4: A mechanism $\mathcal{M} = (\mathcal{A}, \mathcal{P})$ is a Vickrey-Clarke-Groves (VCG) mechanism if $\mathcal{A}$ maximizes the social welfare, and

$$\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = \sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}_{-i})} \hat{b}_j - \sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}), j \neq i} \hat{b}_j, \quad (7)$$

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

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

Definition 5: The VCG-VMPAC mechanism consists of the optimal allocation algorithm that solves IP-VMPAC and the payment function defined by the VCG payment rule given in equation (7).

The VCG-VMPAC mechanism is given in Algorithm 1. The mechanism is run periodically by the cloud provider. VCG-VMPAC has one input parameter, the vector of resource capacities $\mathbf{C} = (C_1, \ldots, C_R)$, and three output parameters: V^*, the optimal social welfare, $\mathbf{x}^*$, the optimal allocation of VM instances to the users, and $\mathcal{P}$ the payments. The mechanism collects the requests from the users, expressed as types (lines 2-4), and determines the optimal allocation by solving the IP-VMPAC (line 6). Once the optimal allocation is determined the mechanism provisions the required number and types of VM instances and determines the payments. The users are then charged the amount determined by the mechanism (lines 9-15). The VCG payment of a user i is calculated by solving the IP-VMPAC to find the allocation and welfare obtained without user i's participation (line 10). Based on the optimal allocation to the users with and without user i's participation, the mechanism 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 11-15).

Being a VCG-based mechanism, VCG-VMPAC is truthful [3], and it determines the optimal allocation. However, the VMPAC is strongly NP-hard, and thus, the execution time of VCG-VMPAC becomes prohibitive for large instances of VMPAC. To be able to solve VMPAC in reasonable time, we resort to greedy mechanisms, which we design in the next section.

Algorithm 2 G-VMPAC-X Mechanism

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

4 Truthful Greedy Mechanisms

In this section, we present the proposed truthful greedy mechanisms and then investigate their properties.

4.1 G-VMPAC-X Truthful Greedy Mechanisms

The VMPAC problem is strongly NP-hard and there is no Fully Polynomial Time Approximation Scheme (FPTAS) for solving it, unless $P = NP$ [17]. Thus, one solution to solve VMPAC is to design heuristic approximation algorithms. In general, approximation algorithms do not necessarily satisfy the properties required to achieve truthfulness, and thus, they need to be specifically designed for truthfulness. Our goal is to design truthful greedy approximation mechanisms that solve the VMPAC problem.

We propose a family of truthful greedy mechanisms, called G-VMPAC-X. The G-VMPAC-X family is given in Algorithm 2. A mechanism from this family is executed periodically by the cloud provider. The mechanism collects the requests from the users expressed as types (lines 1-3) and determines the allocation by calling the allocation algorithm (lines 4-5). The allocation algorithm can be any version of the G-VMPAC-X-ALLOC allocation algorithms that we present later in this section. Once the allocation is determined, the mechanism provisions the required number and types of VM instances

Algorithm 3 G-VMPAC-X-ALLOC Allocation algorithms

1: **Input:** $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of types (bundle, bid)
2: **Input:** $\mathbf{C} = (C_1, \ldots, C_R)$; vector of resource capacities
3: $V = 0$
4: $\mathbf{x} \leftarrow \mathbf{0}$
5: $\tilde{\mathbf{C}} = \mathbf{C}$
6: **for all** $r \in \mathcal{R}$ **do**
7: $f_r \leftarrow 1$, for G-VMPAC-I-ALLOC; or
$f_r \leftarrow \frac{1}{C_r}$ for G-VMPAC-II-ALLOC; or
$f_r \leftarrow \frac{\sum_{i=1}^{N} \hat{a}_{ir} - C_r}{\sum_{i=1}^{N} \hat{a}_{ir}}$, for G-VMPAC-III-ALLOC
8: **for all** $i \in \mathcal{U}$ **do**
9: $d_i = \frac{\hat{b}_i}{\sqrt{\sum_{r=1}^{R} f_r \hat{a}_{ir}}}$
10: Sort $\mathcal{U}$ in decreasing order of d_i
11: **for all** $i \in \mathcal{U}$ **do**
12: $flag \leftarrow$ TRUE
13: **for all** $r \in \mathcal{R}$ **do**
14: $\tilde{C}_r = \tilde{C}_r - \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$
15: **if** $\tilde{C}_r < 0$ **then**
16: $flag \leftarrow$ FALSE
17: **break**;
18: **if** $flag$ **then**
19: $V = V + \hat{b}_i$
20: $x_i = 1$
21: $\tilde{\mathbf{C}} = \tilde{\mathbf{C}}$
22: **Output:** V, $\mathbf{x}$

(line 6). Then, the mechanism determines the payments by calling the PAY function (lines 7-8). The users are then charged the amount determined by the mechanism.

The general form of the allocation algorithm (called G-VMPAC-X-ALLOC) of this family of mechanisms is given in Algorithm 3. G-VMPAC-X-ALLOC has two input parameters: the vector of users declared types $\hat{\boldsymbol{\theta}}$, and the vector of resource capacities $\mathbf{C} = (C_1, \ldots, C_R)$; and two output parameters: V, the total social welfare and $\mathbf{x}$, the allocation of VM instances to the users. The algorithm orders the users (lines 6-10) according to a general *density* metric defined as:

$$d_i = \frac{\hat{b}_i}{\sqrt{\sum_{r=1}^{R} f_r \hat{a}_{ir}}}, \forall i \in \mathcal{U} \tag{8}$$

where $\hat{a}_{ir} = \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr}$ is the amount of each resource of type r requested by user i, and f_r is the *relevance factor* characterizing the scarcity of resources of type r. A higher f_r means a higher scarcity of resource r, thus, a lower density. That means, a user that requests more resources of a scarce type is less likely to receive her requested bundle. G-VMPAC-X-ALLOC algorithm allocates the VM instances to users in decreasing order of their densities.

The choice of relevance factors, f_r, defines the members of the G-VMPAC-X family of allocation algorithms. We consider three choices for f_r, and obtain three allocation algorithms, G-VMPAC-I-ALLOC, G-VMPAC-II-ALLOC, and G-VMPAC-III-ALLOC, as follows:

1) G-VMPAC-I-ALLOC: obtained when $f_r = 1$, $\forall r \in \mathcal{R}$. This is a direct generalization of the one-dimensional case considered by Lehmann *et al.* [19]. This generalization does not take into account the scarcity of different resources and may not work well in situations

Algorithm 4 PAY: Payment Function

1: **Input:** $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of types (bundle, bid)
2: **Input:** $\mathbf{C}$; vector of resource capacities
3: **Input:** $\mathbf{x}^*$; winning users
4: **for all** $i \in \mathcal{U}$, where $\mathcal{U}$ is sorted in decreasing order of d_i **do**
5: $\mathcal{P}_i = 0$
6: **if** x_i^* **then**
7: $l = -1$
8: $(V'^*, \mathbf{x}'^*) =$ G-VMPAC-X-ALLOC $(\hat{\boldsymbol{\theta}} \setminus \hat{\theta}_i, \mathbf{C})$
9: **for all** $j \in \mathcal{U}, d_j < d_i$ in decreasing order of d_j **do**
10: **if** $x_j^* = 0$ **and** $x_j'^*$ **then**
11: $l = j$
12: **break**;
13: **if** l **then**
14: $\mathcal{P}_i = d_l \sqrt{\sum_{r=1}^{R} f_r \hat{a}_{ir}}$
15: **else**
16: $\mathcal{P}_i = 0$
17: **Output:** $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

in which the VM instances are highly heterogeneous in terms of the resources provided.

2) G-VMPAC-II-ALLOC: obtained when $f_r = \frac{1}{C_r}$, $\forall r \in \mathcal{R}$. This addresses the scarcity issues in G-VMPAC-I, by scaling the values of f_r with the inverse of capacity C_r for each resource r.

3) G-VMPAC-III-ALLOC: obtained when $f_r = \frac{\sum_{i=1}^{N} \hat{a}_{ir} - C_r}{\sum_{i=1}^{N} \hat{a}_{ir}}$, $\forall r \in \mathcal{R}$. This relevance factor considers the relative scarcity of resources. As a result, resources with higher demands have a higher f_r and thus, contribute more to decreasing the density. Users requesting more highly demanded resources have lower densities and are less likely to receive their requested bundles.

Once the users are sorted according to their density values (line 10), the algorithm determines the allocation $\mathbf{x}$ (lines 11-22). In doing so, the algorithm checks the feasibility of allocating the requested bundle of each user (lines 12-17). If the allocation is feasible, the algorithm updates V and x_i (lines 19-20). The time complexity of the algorithms is $O(N(RM + \log N))$.

The PAY function is given in Algorithm 4. The PAY function has three input parameters, the vector of users declared types ($\hat{\boldsymbol{\theta}}$), the vector of resource capacities $\mathbf{C}$, and the optimal allocation $\mathbf{x}^*$. It has one output parameter: $\mathcal{P}$, the payment vector for the users. PAY determines the payments of users sorted according to the density metric (lines 4-16). For each user i, PAY sets her initial payment to 0 (line 5). If user i is among the winning users, PAY updates her payment (lines 7-16), otherwise the payment remains 0. The payments are based on the critical values of the winning users. In doing so, PAY calls the allocation algorithm, G-VMPAC-X-ALLOC without considering the participation of user i (line 8). Then, PAY tries to find user j such that she wins in the absence of user i, and she does not win in the presence of user i (lines 9-10). If PAY finds such a user, it stores her index l (lines 11-12), otherwise user i pays 0 (line 16). The payment of winning user i is calculated by multiplying $\sum_{r=1}^{R} \hat{a}_{ir}$ with the highest density among the losing users, user l, who would win if i would not be a winner (line 14).

TABLE 2: 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]	247	760	2,500	4.37	46.96	43.95
GWA-T-3 NorduGrid	34	1	[1-2,147]	[10-1,053,072]	24	14,000	640,000	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	9,700	4,000	2.9	94.49	39.43
METACENTRUM-2009-2	42	[1-60]	[1-61,538]	[10-2,592,130]	44	9,700	178,000	1.55	325.14	21,189.11
PIK-IPLEX-2009-1	36	[1-2560]	[1-29,360]	[10-4,815,007]	88	89,000	470,000	12.15	3,528.22	18,716.06

4.2 G-VMPAC-X Mechanisms Properties

In this subsection, we show that the G-VMPAC-X mechanisms are truthful and determine their approximation ratio. We show first that the allocation algorithms are monotone, and thus, satisfy the first requirement for truthfulness.

Theorem 1: The G-VMPAC-X-ALLOC allocation algorithms are monotone.

We provide the proof of this theorem in Appendix B of the supplemental material.

In the following, we show that the payment algorithm is based on the critical value, and thus, satisfies the second requirement for truthfulness.

Theorem 2: The payment algorithm, PAY, implements the critical value payment.

We provide the proof of this theorem in Appendix C of the supplemental material.

We now show that the proposed mechanisms are truthful.

Theorem 3: The proposed mechanisms in the G-VMPAC-X family are truthful.

Proof: The allocation algorithms in the G-VMPAC-X family are monotone (Theorem 1) and the payment (implemented by PAY) is the critical value payment (Theorem 2), therefore, according to [18], the mechanisms in the G-VMPAC-X family are truthful. □

In Appendix D of the supplemental material, we analyze the effect of untruthful reporting on the utility of the users participating in the G-VMPAC-II mechanism by considering an example.

In the following, we determine the approximation ratio of the greedy mechanisms in the G-VMPAC-X family.

Theorem 4: The approximation ratio of the mechanisms in the G-VMPAC-X family is $\sqrt{NRC_{max}}$, where $C_{max} = \max_{r \in \mathcal{R}} C_r$.

We provide the proof of this theorem in Appendix E of the supplemental material.

In our previous work [20], we proposed two greedy mechanisms having an approximation ratio of RC_{max}. In this paper, our proposed mechanisms obtain a better approximation ratio than that of [20] for many instances of the VMPAC problems that are of practical interest. This is due to the fact that in many actual instances of the VMPAC problem $RC_{max} > N$, since the number of requests is less than the total maximum capacity of the resources. Note that the obtained bound is for the extreme worst case scenario in which for G-VMPAC-I, we assume $\sum_{r=1}^{R} \hat{a}_{ir} = 1$. For example, this corresponds to a request of 1MB of storage, which is not realistic in practice. For practical scenarios, we expect the solution to be much closer to the optimal, as validated by the experimental results presented in the next section. The approximation ratio becomes constant under the realistic assumption that the size of the requests are within a given range. This is the case for the current cloud providers, which offer a bounded number of VM instances for each request (e.g., Microsoft Azure currently offers a maximum of 20 VM instances to each user). In Appendix F of the supplemental material, we explain in details why the approximation ratio is constant for these practical cases.

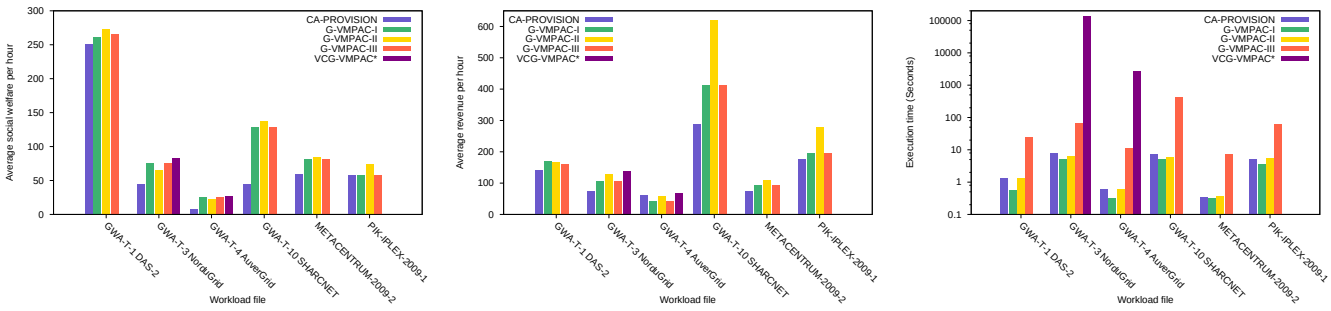
5 Experimental Results

We perform extensive experiments with real workload data in order to investigate the properties of the proposed mechanisms in the G-VMPAC-X family, and the VCG-VMPAC mechanism (optimal). We also compare our proposed mechanisms with CA-PROVISION [15]. Since CA-PROVISION considers only the computational resource, in our experiments CA-PROVISION does not use the amount of other requested resources such as memory and storage when making allocation decisions. For the VCG-VMPAC mechanism, we use the CPLEX solver to solve the VMPAC problem optimally. The CPLEX 12 solver is provided by IBM ILOG CPLEX Optimization Studio for Academics Initiative [21].

All five mechanisms were executed 59,636 times with a total of 3,514,150 user requests. The auctions are generated using six workload logs from the Grid Workloads Archive [22] and the Parallel Workloads Archive [23]. We present statistics of the logs in Table 2. The mechanisms are implemented in C++ and the experiments are conducted on Intel 2.93GHz Quad Proc Hexa Core nodes with 90GB 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

Because real users request data have not been publicly released by cloud providers yet, for our experiments, we rely on well studied and standardized workloads from both the Grid Workloads Archive [22] and the Parallel Workloads Archive [23]. From the Grid Workloads Archive, we selected four out of six available logs.



(a) (b) (c)

Fig. 1: G-VMPAC-X performance: (a) Social welfare; (b) Revenue; (c) Execution time. (*VCG-VMPAC was not able to determine the allocation for GWA-T-1 DAS-2, GWA-T-10 SHARCNET, METACENTRUM-2009-2, and PIK-IPLEX-2009-1 in feasible time, and thus, there are no bars in the plots in Figs. 1 to 7 for those cases)

These logs are: 1) DAS-2 traces from a research grid at the Advanced School for Computing and Imaging in Netherlands; 2) NorduGrid traces from the NorduGrid system; 3) AuverGrid traces from the AuverGrid system; 4) SHARCNET traces from SHARCNET clusters installed at several academic institutions in Ontario, Canada. From the Parallel Workloads Archive, we selected two logs that were recorded most recently. These logs are: 5) MetaCentrum from the national grid of the Czech republic; 6) IBM iDataPlex Cluster log from the Potsdam Institute for Climate Impact Research (PIK) in Germany. The logs are selected based on the availability of recorded both CPU and memory requests/usage. Table 3 provides a brief description of the selected workloads. The table contains the names of the log files, the durations the logs were recorded, and the total number of submitted jobs. In our experiments, each job in a log represents a user request. In addition, each hour of a log represents one auction.

We consider each log as a series of auctions, where the users can submit their requests over time to a cloud provider. We setup the auctions to run every hour just to follow the standard practice in Amazon EC2. Participants of each auction include the new users and those users who are not served and their deadline has not been exceeded. The new arriving users are indicated based on the submission time of their requests.

To generate the user requests for the experiments, we extract the data from six fields of the log files as follows: (1) JobID: the jobs identifier; (2) SubmitTime: the job submission time; (3) RunTime: the time the job needs to complete its execution; (4) ReqNProcs: the requested number of processors; (5) Used Memory: the average used memory per processor; (6) AverageCPUTimeUsed: the average CPU time over all allocated processors. Since the amount of storage usage was not recorded in the workloads, to generate the requested storage, we use the value of this field. In each log, we remove the jobs with missing values in these fields.

For each job in a log, we generate a user request. Since the logs provide data on resource usage, we consider these as values for the requested a_{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 resource type. As a result, a user request contains the requested number of CPUs, the amount of memory and the amount of storage. To generate bids for users, we generate a random number b_i for each user i between 1 and 10. We also generate a deadline for each job request which is between 3 to 6 times the job's runtime. The deadline is when a user stops bidding for her requested bundle irrespective of her allocation.

TABLE 3: Workload logs.

Logfile	Duration (hours)	Jobs
GWA-T-1 DAS-2	13,534	1,099,803
GWA-T-3 NorduGrid	8,127	276,144
GWA-T-4 AuverGrid	8,298	274,455
GWA-T-10 SHARCNET	6,909	1,018,355
METACENTRUM-2009-2	2,402	102,538
PIK-IPLEX-2009-1	20,366	742,855

5.2 Analysis of Results

We compare the performance of G-VMPAC-X, VCG-VMPAC and CA-PROVISION for different workloads. For each workload, we compute the execution time and the average social welfare, revenue, and utilization of the resources per hour for each mechanisms.

We present the results for all the selected logs. As for VCG-VMPAC (optimal), it is only able to complete the experiments for two of the logs: GWA-T-3 NorduGrid and GWA-T-4 AuverGrid. VCG-VMPAC takes 2,623.54 seconds for GWA-T-4 AuverGrid and 132,678.31 seconds for GWA-T-3 NorduGrid. For the rest of the logs, VCG-VMPAC is not able to solve the VMPAC problem for the selected workloads within 48 hours. VCG-VMPAC is unable to solve these problems due to exceeding the memory capacity (90 GB) of the machines we used to run the experiments. This is the reason that we do not present the results of the optimal VCG-VMPAC mechanism for all the logs in Figs. 1-3. This shows that the optimal mechanism is not suitable for solving large scale VMPAC problems, and thus, we need to resort to heuristic mechanisms. Because of this limitation with VCG-VMPAC, we compare VCG-VMPAC with the rest of the mechanisms in Appendix G of the supplemental

material considering only 500 hours of the logs to show the performance of all mechanisms.

First, we analyze the performance of the mechanisms in terms of social welfare. Fig. 1a shows the average social welfare per hour for the selected logs. All mechanisms obtain the highest social welfare per hour for GWA-T-1 DAS-2 because of the combination of several factors such as capacities, number of request per hour, and the percentage of users served.

CA-PROVISION performs slightly better on PIK-IPLEX-2009-1 than on the rest of the logs. This is due to the fact that in this log the CPU is the scarcest resource compared to the other resources, and this mechanism only relies on one dimension (the computational resource). As a result, using this mechanism users who request fewer CPUs with relatively high bid get higher priorities than others. Therefore, this mechanism selects the users who are more likely to be in the optimal solution, and thus, it achieves higher social welfare in this case. This is not the case for the other logs where CPU is not the only scarce resource. Since CA-PROVISION considers only CPU compared to other methods that consider all the resource types, it has the lowest performance in general.

The performance of G-VMPAC-I is susceptible to the relative magnitude of the amount of the requested resources. Therefore, if the requested resources are highly heterogeneous (say 10,000 MB of storage vs. a few number of CPUs), a resource that has a higher relative magnitude than the others becomes dominant in determining the density metric by G-VMPAC-I. As a result, that resource has the most impact on the performance of G-VMPAC-I. If such resource is scarce, then G-VMPAC-I obtains the best performance. In GWA-T-3 NorduGrid, where storage has a high relative magnitude and is scarce, G-VMPAC-I performs better than other mechanisms.

In most cases, G-VMPAC-II which uses the inverse of the capacity as a weighting factor, achieves a higher social welfare than the rest of the mechanisms. This is due to the fact that G-VMPAC-II considers the impact of all resources in order to calculate the density metric for each user.

The results show that for GWA-T-4 AuverGrid, G-VMPAC-III achieves a social welfare that is the highest. This is due to the fact that the total amount of requested resource, $\sum_{i=1}^{N} \hat{a}_{ir}$, is relatively close to the capacity of that resource, C_r in this log. In such case, G-VMPAC-III considers the impact of all resources in order to calculate the density metric for each user. However, when the sum of the requested resources are very high in comparison with the available capacity, G-VMPAC-III performs very close to G-VMPAC-I. This is due to the fact that $f_r = \frac{\sum_{i=1}^{N} \hat{a}_{ir} - C_r}{\sum_{i=1}^{N} \hat{a}_{ir}}$ approaches 1, which is the case for G-VMPAC-I.

Fig. 1b shows the average revenue per hour achieved by the cloud provider when using the five mechanisms.

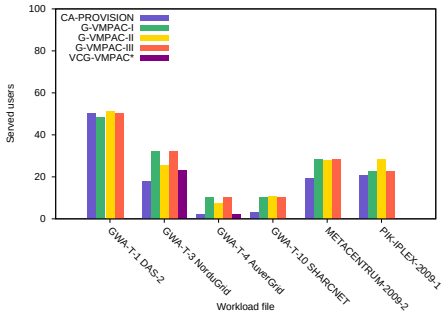


Fig. 2: G-VMPAC-X performance: Users served.
(*see Fig. 1 note on VCG-VMPAC)

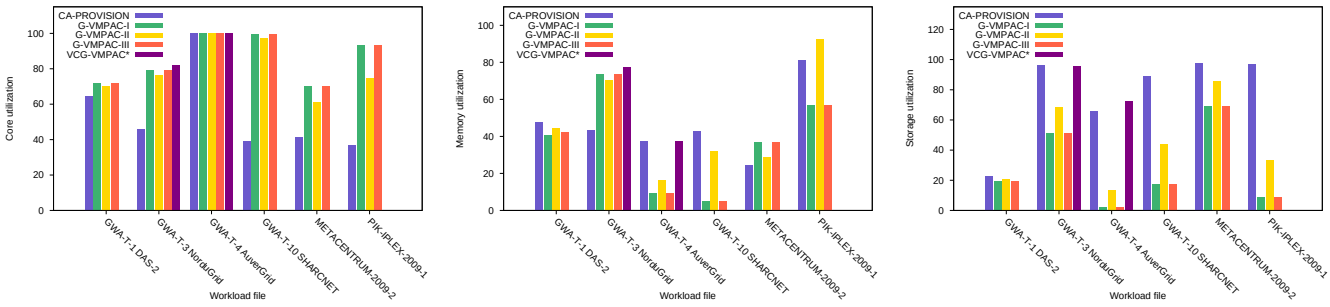
Even though, all the mechanisms try to maximize the social welfare, they also obtain high revenue for the cloud provider. G-VMPAC-II achieves the highest revenue among all the greedy mechanisms for all workloads.

Fig. 1c shows the execution times of the mechanisms on a logarithmic scale. As we expected from the time complexity of the mechanisms, the execution times of G-VMPAC-X and CA-PROVISION are in the same order of magnitude for each of the logs. The optimal mechanism, VCG-VMPAC, could not find the solutions even after 48 hours for four out of six logs. This is due to the fact that the problem gets more complex for higher number of requests, number of auction hours, and available capacity. VCG-VMPAC is able to solve VMPAC for the full GWA-T-4 AuverGrid log since the available capacity of CPU in each auction is very low. As a result, the feasible solution area becomes strictly limited, and the CPLEX solver can find the optimal solutions faster than for the rest of the logs.

Fig. 2 shows the percentage of served users for each of the five mechanisms. Note that VCG-VMPAC does not serve a higher number of users than the other mechanisms. This is due to the fact that the optimal mechanism finds the most valuable subset of users in order to maximize the social welfare.

Figs. 3a to 3c 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 social welfare not the utilization of the resources. The memory and storage utilization in the case of CA-PROVISION are higher than those of the other mechanisms. CA-PROVISION chooses users who value CPUs the most without considering their requested memory and storage. These users are more likely to requests higher amounts of memory and storage which results in a higher memory and storage utilization for CA-PROVISION.

From all the above results, we conclude that G-VMPAC-II finds near-optimal solutions to the VMPAC problem and requires small execution times. The small execution time of our proposed G-VMPAC-X mechanisms makes them good candidates for deployment on the current cloud computing systems.



(a) (b) (c)

Fig. 3: G-VMPAC-X resource utilization: (a) Cores; (b) Memory; (c) Storage. (*see Fig. 1 note on VCG-VMPAC)

6 Conclusion

We addressed the problem of dynamic VM provisioning and allocation in clouds by designing truthful mechanisms that give incentives to the users to reveal their true valuations for their requested bundles of VM instances. The proposed truthful optimal and greedy mechanisms for solving the VMPAC problem consider the presence of resources of multiple types. We determined the approximation ratio of the proposed greedy mechanisms and investigated their properties by performing extensive experiments. The results showed that the proposed greedy mechanisms determine near optimal solutions while effectively capturing the dynamic market demand, provisioning the computing resources to match the demand, and generating high revenue. In addition, the execution time of the proposed greedy mechanisms is very small. As a recommendation, G-VMPAC-II is the best choice for the cloud providers since it yields the highest revenue among the proposed greedy mechanisms. We plan to implement a prototype allocation system in an experimental cloud computing system to further investigate the performance of our proposed mechanisms.

Acknowledgments

This paper is a revised and extended version of [20] presented at the IEEE 6th Intl. Conference on Cloud Computing (CLOUD 2013). This research was supported, in part, by NSF grants DGE-0654014 and CNS-1116787.

References

[1] WindowsAzure. [Online]. Available: http://www.windowsazure.com/en-us/pricing/calculator/

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

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

[4] G. Wei, A. 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.

[5] N. Jain, I. Menache, J. Naor, and J. Yaniv, "A truthful mechanism for value-based scheduling in cloud computing," *Theory of Computing Systems*, pp. 1–19, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00224-013-9449-0

[6] Z. Kong, C.-Z. Xu, and M. Guo, "Mechanism design for stochastic virtual resource allocation in non-cooperative cloud systems," in *Proc. 4th IEEE Intl. Conf. on Cloud Computing*, 2011, pp. 614–621.

[7] Y. Wang, A. Nakao, and A. V. Vasilakos, "Heterogeneity playing key role: Modeling and analyzing the dynamics of incentive mechanisms in autonomous networks," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 7, no. 3, p. 31, 2012.

[8] D. Ardagna, B. Panicucci, and M. Passacantando, "Generalized Nash equilibria for the service provisioning problem in cloud systems," *IEEE Transactions on Services Computing*, vol. 6, no. 4, pp. 429–442, 2013.

[9] V. Di Valerio, V. Cardellini, and F. Lo Presti, "Optimal pricing and service provisioning strategies in cloud systems: a Stackelberg game approach," in *Proc. of the 6th IEEE Intl. Conf. on Cloud Computing*, 2013, pp. 115–122.

[10] X. Zhou, S. Gandhi, S. Suri, and H. Zheng, "ebay in the sky: strategy-proof wireless spectrum auctions," in *Proc. of the 14th ACM Intl. Conf. on Mobile Comp. and Networking*, 2008, pp. 2–13.

[11] X. Zhou and H. Zheng, "Trust: A general framework for truthful double spectrum auctions," in *Proc. of IEEE INFOCOM 2009*, 2009, pp. 999–1007.

[12] G. S. Kasbekar and S. Sarkar, "Spectrum auction framework for access allocation in cognitive radio networks," *IEEE/ACM Transactions on Networking*, vol. 18, no. 6, pp. 1841–1854, 2010.

[13] F. Wu and N. Vaidya, "A strategy-proof radio spectrum auction mechanism in noncooperative wireless networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 5, pp. 885–894, 2013.

[14] J. Jia, Q. Zhang, Q. Zhang, and M. Liu, "Revenue generation for truthful spectrum auction in dynamic spectrum access," in *Proc. 10th ACM Intl. Symp. on Mobile Ad Hoc Networking and Computing*, 2009, pp. 3–12.

[15] S. Zaman and D. Grosu, "Combinatorial auction-based dynamic vm provisioning and allocation in clouds," in *Proc. 3rd IEEE Intl. Conf. on Cloud Comp. Tech. and Sci.*, 2011, pp. 107–114.

[16] ——, "Combinatorial auction-based allocation of virtual machine instances in clouds," *J. of Parallel and Distributed Computing*, vol. 73, no. 4, pp. 495–508, 2013.

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

[18] A. Mu'alem and N. Nisan, "Truthful approximation mechanisms for restricted combinatorial auctions," in *Proc. 18th Nat. Conf. on Artificial Intelligence*, 2002, pp. 379–384.

[19] D. Lehmann, L. Oćallaghan, and Y. Shoham, "Truth revelation in approximately efficient combinatorial auctions," *Journal of the ACM*, vol. 49, no. 5, pp. 577–602, 2002.

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

[21] IBM ILOG CPLEX V12.1 user's manual. [Online]. Available: ftp://ftp.software.ibm.com/software/websphere/ilog/docs/

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

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

Mahyar Movahed Nejad received his BSc degree in mathematics from Iran University of Science and Technology. He received his MSc degree in socio-economic 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 distributed systems, big data analytics, game theory, network optimization, and integer programming. He is a student member of the IEEE and the INFORMS.

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. Her research interests include distributed systems, cloud computing, big data, game theory and optimization. She is a student member of the ACM, the IEEE, and the IEEE Computer Society.

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 seventy 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.