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

Mahyar Movahed Nejad
Dept. Computer Science
Wayne State University
Detroit, MI 48202, USA
Email: mahyar@wayne.edu

Lena Mashayekhy
Dept. Computer Science
Wayne State University
Detroit, MI 48202, USA
Email: mlena@wayne.edu

Daniel Grosu
Dept. Computer Science
Wayne State University
Detroit, MI 48202, USA
Email: dgrosu@wayne.edu

***Abstract*—Designing efficient mechanisms for Virtual Machine (VM) provisioning and allocation is a major challenging problem that needs to be solved by cloud providers. We formulate the VM provisioning and allocation problem in clouds as an integer program and design truthful greedy mechanisms that solve it. We show that the proposed mechanisms are truthful, that is, the users do not have incentives to lie about their requested bundles of VM instances and their valuations. We perform extensive experiments in order to investigate the performance of the proposed mechanisms.**

***Keywords*-cloud computing; truthful mechanism; resource allocation; greedy heuristic.**

I. Introduction

The number of enterprises and individuals that are outsourcing their workloads to cloud providers increased rapidly. Cloud providers can offer Infrastructure as a Service (IaaS) by providing CPUs, storage, networks and other low level resources to their customers. These different types of resources are offered in the form of Virtual Machine (VM) instances. For example, 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 employ fixed-price and auction-based mechanisms in order to provision resources in the form of VM instances and then allocate them to the customers. In the auction-based mechanisms, each user bids for a subset of available VM instances (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 user is either assigned her entire requested bundle of VMs and she pays for it, or she does not obtain any bundle and pays nothing. The users are also selfish in a 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). An example of such auction-based mechanism is the spot market introduced by Amazon [2]. Such mechanisms usually run in short time-windows (e.g., every hour) to efficiently provision the unutilized resources of the cloud provider.

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. 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.

Our Contribution. We address the problem of VM provisioning and allocation in clouds in the presence of multiple types of resources. We design truthful greedy mechanisms that give incentives to the users to reveal their true valuations for their requested bundles of VM instances. We formulate the problem as an integer program equivalent to the multidimensional knapsack problem, which is strongly NP-hard [3]. In the absence of feasible optimal algorithms for solving the problem, we design greedy mechanisms. In general, greedy algorithms do not necessarily satisfy the properties required to guarantee truthfulness. In this paper, we propose truthful greedy mechanisms in order to solve the problem of VM provisioning and allocation in clouds. We determine the approximation ratio of the proposed mechanisms and perform extensive simulation experiments. The results show that the proposed mechanisms are able to find near optimal allocations while satisfying the truthfulness property.

Related Work. Due to space limitations, we provide only a very brief review of the closely related work. Lehmann *et al.* [4] proposed a greedy truthful mechanism for single-unit combinatorial auctions. However, our focus is on the design of greedy truthful mechanisms in multi-unit settings. Several studies focused on finding solutions for multi-unit combinatorial auctions without considering the truthfulness [5], [6]. The reader is referred to [7] for a comprehensive introduction to mechanism design. Greedy algorithms for

solving the multidimensional knapsack problem (MKP) have been extensively studied [3]. However, none of these studies considered the design of truthful mechanisms.

Wood *et al.* [8] proposed an approach for dynamic provisioning of VMs by defining a unique metric based on the consumption of the three resources: CPU, network and memory. Their approach determines a new mapping of resources to VMs. Gorlach and Leymann [9] proposed a method for dynamic provisioning of services in clouds in order to optimize the distribution of services within a certain infrastructure. Xiong *et al.* [10] considered an economical provisioning where VMs are allocated to achieve a balanced resource allocation and a better overall performance. Sharma *et al.* [11] proposed a system considering the cost of VMs. They modeled this problem as an integer program to minimize the cost by reducing the time to transit to new configurations and optimizing the selection of a virtual server configuration. All the above studies focused on addressing the VM provisioning problem in clouds, however, none of them considered the design of truthful mechanisms for VM provisioning.

The closest work to ours is by Zaman and Grosu [12], [13] who proposed truthful approximation mechanisms for combinatorial auction-based allocation of VM instances in clouds in static and dynamic settings. 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. Lampe *et al.* [14] proposed a heuristic approach considering several types of resources. However, they did not propose a truthful mechanism.

Organization. The rest of the paper is organized as follows. In Section II, we describe the VM provisioning and allocation problem in clouds. In Section III, we introduce the basic concepts of mechanism design. In Section IV, we present the proposed mechanisms and characterize their properties. In Section V, we evaluate the mechanisms by extensive simulation experiments. In Section VI, we summarize our results and present possible directions for future research.

II. 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, let's consider that CPU represents the type 1 resource, memory the type 2 resource, and storage the type 3 resource. We can characterize a possible VM instance (of type $m = 1$) by: $w_{11} = 1$ core, $w_{12} = 1.6$ GB, and $w_{13} = 150$ GB.

Table I: Notation

$\mathcal{U}$	Set of users $\{1, \ldots, N\}$
$\mathcal{VM}$	Set of VMs $\{1, \ldots, M\}$
$\mathcal{R}$	Set of resources $\{1, \ldots, R\}$
S_i	The requested bundle of user $i \in \mathcal{U}$
$v_i(S_i)$	Value of the requested bundle S_i of user $i \in \mathcal{U}$
k_{im}	The number of VMs of type m requested by user $i \in \mathcal{U}$
b_i	The bid of user $i \in \mathcal{U}$
w_{mr}	The amount of resource of type $r \in \mathcal{R}$ provided by one VM instance of type $m \in \mathcal{VM}$
C_r	Capacity of resource $r \in \mathcal{R}$

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, $(< 1, 3, 4, 2 >, \$10)$ 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 \$10. 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 indicator variables defined as follows: $x_i = 1$ if bundle S_i is allocated to user i, and $x_i = 0$, otherwise. Table I summarizes the notation used throughout the paper.

The cloud provider's goal is to allocate resources to users in such a way that the allocation maximizes the revenue. This would be the most reasonable objective, but since very little is known about revenue maximization in the context of mechanism design, we will consider the standard mechanism design objective, that is, maximization of V, the sum of the users' valuations [7]. Since the valuation of a user represents her willingness to pay, we expect that maximizing the sum of the valuations will have a positive effect on increasing the revenue obtained by the cloud provider.

We formulate the problem of VM provisioning and allocation in clouds (VMPAC) as an Integer Program 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), where the knapsack constraints are the resource capacity constraints and the bundles are the items [3]. The objective is to select a subset of items for the multidimensional knapsack maximizing the total value.

III. Mechanism Design Framework

In this section, we present the basic concepts of mechanism design. 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, and the type of a user i is denoted by $\theta_i = (S_i, b_i)$. 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, $\hat{S}_i$, and zero value, otherwise. Thus, the valuation function for user i is as follows:

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

The goal is to design a truthful mechanism that maximizes the social welfare V.

We denote by $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_N)$ the vector of types of all users. $\boldsymbol{\theta}_{-i}$ is the vector of all types except user i's type (i.e., $\boldsymbol{\theta}_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_N)$). User i has a 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's i declared type. Note that $\theta_i = (S_i, b_i)$ is user's i 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* (or incentive compatible) 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 [15].

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 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:

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

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

IV. Truthful Greedy Mechanisms

The VMPAC problem is strongly NP-hard and there is no Fully Polynomial Time Approximation Scheme (FPTAS) for solving it, unless $P = NP$ [3]. 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. Our goal is to design truthful greedy approximation mechanisms that solve the VMPAC problem.

We propose a family of truthful mechanisms, called G-VMPAC-X. The general form of the allocation algorithm of this family of mechanisms is given in Algorithm 1. G-VMPAC-X has two input parameters: the vector of users

Algorithm 1 G-VMPAC-X Allocation algorithms for VMPAC

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; or $f_r \leftarrow \frac{1}{C_r}$ for G-VMPAC-II
8: **for all** $i \in \mathcal{U}$ **do**
9: $e_i = \frac{v_i}{\sum_{r=1}^{R} f_r a_{ir}}$
10: Sort $\mathcal{U}$ in decreasing order of e_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: $V = V + v_i$
20: $x_i = 1$
21: $\tilde{\mathbf{C}} = \tilde{\mathbf{C}}$
22: **Output:** V, $\mathbf{x}$

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 *efficiency* metric defined as:

$$e_i = \frac{v_i}{\sum_{r=1}^{R} f_r a_{ir}}, \forall i \in \mathcal{U} \tag{7}$$

where $a_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$ 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 efficiency. That means, a user that requests more resources of a scarce type is less likely to receive her requested bundle.

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

1) G-VMPAC-I: 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.* [4]. This generalization does not take into account the scarcity of different resources and may not work well in situations in which the VM instances are highly heterogeneous in terms of the resources provided.

2) G-VMPAC-II: 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.

Once the users are sorted according to their efficiency values, the algorithms determine the allocation $\mathbf{x}$ (lines 11-22). The time complexity of the algorithms is $O(N(RM + \log N))$.

Theorem 1: The algorithms in the G-VMPAC-X family are monotone.

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($\hat{\boldsymbol{\theta}}, \mathbf{C}$)
6: Provisions and allocates VM instances according to $\mathbf{x}^*$.
7: {Payment}
8: $\mathcal{P}$ =PAY($\hat{\boldsymbol{\theta}}, \mathbf{C}, \mathbf{x}$)

Proof: We show that the algorithms that are part of G-VMPAC-X family produce monotone allocations. In order to show this, we assume that user i with declared type $\hat{\theta}_i$ is allocated her requested bundle and show that she is still allocated if she declares type $\hat{\theta}'_i$, where $\hat{\theta}'_i \succeq \hat{\theta}_i$. Here, $\hat{\theta}'_i \succeq \hat{\theta}_i$, means that user i may request a VM bundle with fewer resources of each type or report a higher value. We separate the proof into three cases as follows.

i) User i declares a higher value, i.e., $\hat{v}'_i > \hat{v}_i$. This leads to a higher efficiency, $e'_i > e_i$ in all G-VMPAC-X algorithms. This is due to the fact that the requested amount of each resource is the same in both bundles corresponding to $\hat{\theta}_i$ and $\hat{\theta}'_i$. Thus, user i remains in the same or advances to a higher position in the greedy order when declaring $\hat{\theta}'_i$. As a result, her allocation will not change when any of the algorithms that are members of G-VMPAC-X is used.

ii) User i declares a bundle $\hat{S}'_i =< \hat{k}'_{i1}, \hat{k}'_{i2}, \ldots, \hat{k}'_{iM} >$ with fewer resources of each type than bundle $\hat{S}_i =< \hat{k}_{i1}, \hat{k}_{i2}, \ldots, \hat{k}_{iM} >$, i.e., $\sum_{m \in \mathcal{VM}} \hat{k}'_{im} w_{mr} \leq \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr}, \forall r \in \mathcal{R}$. That means, user i requests fewer resources and as a result, $a'_{ir} < a_{ir}, \forall r \in \mathcal{R}$. In G-VMPAC-I and G-VMPAC-II, this leads to a higher efficiency for user i, that is, $e'_i > e_i$.

iii) User i declares a higher value, $\hat{v}'_i > \hat{v}_i$, and a bundle $\hat{S}'_i$ with fewer resources than $\hat{S}_i$. From the above two cases, user i will still be allocated the bundle, thus remaining among the winning users.

In all three cases, user i's allocation will not change, and she remains among the winning users. This implies that all the algorithms in the G-VMPAC-X family are monotone. ■

The G-VMPAC-X family of truthful mechanisms is given in Algorithm 2. A mechanism from this family is executed periodically by the cloud provider. The mechanism collects the requests from the user 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. Once the allocation is determined the mechanism provisions the required number and types of VM instances (line 6) and determines the payments by calling the PAY function (lines 7-8). The users are then charged the amount determined by the mechanism. The PAY function is given in Algorithm 3. 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 given

Algorithm 3 PAY: Payment Function

Input: $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of types (bundle, bid)
Input: $\mathbf{C}$; vector of resource capacities
Input: $\mathbf{x}^*$; winning users
for all $i \in \mathcal{U}$, where $\mathcal{U}$ is sorted in decreasing order of e_i **do**
  $\mathcal{P}_i = 0$
  if x_i^* **then**
    $l = -1$
    $(V'^*, \mathbf{x}'^*) =$ G-VMPAC-X $(\hat{\boldsymbol{\theta}} \setminus \hat{\theta}_i, \mathbf{C})$
    for all $j \in \mathcal{U}$ in decreasing order of e_i **do**
      if $x_j^* = 0$ **and** $x_j'^*$ **then**
        $l = j$
        break;
    if l **then**
      $\mathcal{P}_i = e_l \sum_{r=1}^{R} f_r a_{ir}$
    else
      $\mathcal{P}_i = 0$
Output: $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

by $\mathbf{x}^*$. It has one output parameter: $\mathcal{P}$, the payment vector for the users. The payments are based on the critical values of the winning users. The payment of winning user i is calculated by multiplying $\sum_{r=1}^{R} a_{ir}$ with the highest bid density among the loosing users who would win if i would not be a winner. That is, the winner pays the critical value.

We now show that the proposed mechanisms are truthful.

Theorem 2: The 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 is the critical value payment (implemented by PAY), therefore, according to [15], the mechanisms in the G-VMPAC-X family are truthful. ■

We now determine the approximation ratio of the greedy mechanisms in the G-VMPAC-X family.

Theorem 3: The approximation ratio of the mechanisms in the G-VMPAC-X family is RC_{max}, where $C_{max} = \max_{r \in \mathcal{R}} C_r$.

Proof: We consider the general form of efficiency $e_i = \frac{v_i}{\sum_{r=1}^{R} f_r a_{ir}}$. Let X^* be set of users in the optimal solution, and V^* be the optimal value. Let X and V be the set of users and the value in the obtained solution by G-VMPAC-X, respectively. We need to prove that $V^* \leq V\alpha$, where α is the approximation ratio.

We define $\hat{X} = X \setminus (X \cap X^*)$ and $\hat{X}^* = X^* \setminus (X \cap X^*)$. Therefore, we have $\hat{X} \cap \hat{X}^* = \emptyset$. Based on the new sets $\hat{X}$ and $\hat{X}^*$, the corresponding values are $\hat{V}$ and $\hat{V}^*$, respectively. Now, instead of proving $V^* \leq \alpha V$, it is sufficient to prove that $\hat{V}^* \leq \alpha \hat{V}$. This is due to the fact that we can subtract the values of the users in $(X \cap X^*)$ from both V and V^*.

$$\hat{V}^* = \sum_{i \in \hat{X}^*} v_i \leq \alpha \sum_{i \in \hat{X}} v_i = \alpha \hat{V} \tag{8}$$

We define a set of users D_i for user $i, \forall i \in \mathcal{U}$ including user i such that if $j \in D_i$ then $j \geq i$ (based on the order) and $j \in X^*$ but $j \notin X$ because of user i. Meaning that, user i blocks each user in D_i from entering X. It is obvious that $X^* \subseteq \bigcup_{i \in X} D_i$. Then, $\sum_{i \in \hat{X}^*} v_i \leq \sum_{i \in \bigcup_{j \in \hat{X}} D_j} v_i \leq \alpha \sum_{i \in \hat{X}} v_i$ Therefore, it is sufficient to show for every $i \in \hat{X}$ that: $\sum_{j \in D_i} v_j \leq \alpha v_i$. Note that every $j \in D_i$ appeared after i in the greedy order and thus $e_j \leq e_i$ then

$$v_j \leq \frac{v_i \sum_{r=1}^{R} f_r a_{jr}}{\sum_{r=1}^{R} f_r a_{ir}} \tag{9}$$

Summing over all $j \in D_i$, we have:

$$\sum_{j \in D_i} v_j \leq \sum_{j \in D_i} \frac{v_i \sum_{r=1}^{R} f_r a_{jr}}{\sum_{r=1}^{R} f_r a_{ir}} \leq \frac{v_i}{\sum_{r=1}^{R} f_r a_{ir}} \sum_{j \in D_i} \sum_{r=1}^{R} f_r a_{jr} \tag{10}$$

Due to space limitation and the fact that the proof for G-VMPAC-I is similar to that for G-VMPAC-II, we show the proof only for G-VMPAC-II.

Since X^* is an allocation and $f_r = \frac{1}{C_r}$ for G-VMPAC-II we have $\sum_{j \in D_i} \sum_{r=1}^{R} \frac{a_{jr}}{C_r} \leq R$. Replacing this in equation (10) we obtain

$$\sum_{j \in D_i} v_j \leq \frac{R v_i}{\sum_{r=1}^{R} \frac{a_{ir}}{C_r}} \tag{11}$$

The worst case is when $\sum_{r=1}^{R} \frac{a_{ir}}{C_r}$ has the minimum value which is $\frac{1}{C_{max}}$, where $C_{max} = \max_{r \in \mathcal{R}} C_r$. Therefore $\sum_{j \in D_i} v_j \leq R C_{max} v_i$. As a result, the approximation ratio is $\alpha = R C_{max}$. ■

In the next section we evaluate the performance of the proposed mechanisms by performing extensive simulation experiments.

V. Experimental Results

We perform extensive simulation experiments in order to investigate the properties of the mechanisms in the G-VMPAC-X family. The G-VMPAC-X mechanisms are implemented in C++ and the experiments are conducted on an Intel 2.53GHz with 3GB RAM with Linux as the operating system.

A. Experimental Setup

We generate VM instance requests corresponding to systems with 16 to 1024 users. The number of VM instances and resource types offered by the cloud provider are the same in all the experiments. The generated requests are based on realistic data combining publicly available information provided by Amazon EC2 and Microsoft Azure as follows. We consider two setting depending on the types of VM instances available to users: homogeneous and heterogeneous VM types. Each of these VM instances has specific resource demands with respect to three available resource types: cores, memory and storage.

Table II: Homogeneous VM instance types.

	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

Table III: Heterogeneous VM instance types.

	Data Processing		Heavy Computation	
	High-Storage $m=1$	High-Storage $m=2$	High-CPU $m=3$	High-Memory $m=4$
CPU	1	2	5	6
Memory (GB)	1.7	1.7	3.4	17.1
Storage (GB)	1500	4000	20	10

In the homogeneous setting, the amount of resources in VM types are proportional, and we use the same VM types as those offered by Amazon EC2. We also set the amount of each resource type provided by a VM instance to be the same as in the specifications provided by Amazon Web Services for its Spot Instances and Elastic Computing Cloud (EC2) (see Table II).

In the heterogeneous setting, the amount of resources provided by different types of VM instances are not related. In Table III, we present the four heterogeneous types of VM instances that we use for our experiments. These types of VMs can be used for large data processing (High-Storage) or heavy computations (High-Memory and High-CPU).

Users can request between 1 and 20 VM instances of each type. We generate bids based on Amazon Spot market report on users bidding strategies [2]. Amazon regularly updates its spot price history based on the past 90 days of activity. Amazon reported that most users bid between the price of reserved instances and on-demand prices. By doing so, these users saved between 50% to 66% compared to the on demand prices. The lowest price of the reserved instances is for the *Heavy Utilization Reserved Instances* which is $\$0.013$ per hour for a small VM instance. However, the trade off is that the user's requested bundles can be reclaimed by a cloud provider if the spot price exceeds their submitted bid prices. Thus, some users bid above on-demand prices and up to twice the on-demand prices in some cases. To generate bids for users requesting homogeneous VMs, we generate a random number, b_i^0, for each user i from the range $[0.013, 0.24]$ for a small VM instance. Then, we multiply the random number by the total weights of VMs in the user's requested bundle. The total weight of a VM instance for user i is $\sum_{m=1}^{M} 2^{m-1} k_{im}$. To generate bids for users requesting heterogeneous VMs, we generate a random number, b_{im}^0, for each user i from the above-mentioned range for each VM instance $m \in \mathcal{VM}$. Then, we multiply the random number by the number of VMs of type m in the user's requested bundle, i.e., k_{im}. The parameters and their generated values for the experiments are listed in Table IV. We use the CPLEX branch-and-bound solver provided by IBM ILOG CPLEX Optimization Studio for Academics Initiative [16] for solving the VMPAC problem. We compare the results obtained by the CPLEX solver (denoted by OPT) with those obtained by the proposed mechanisms.

Table IV: Simulation Parameters

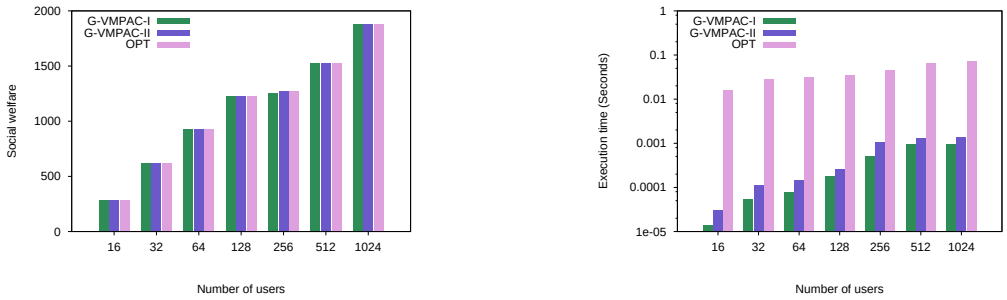
Param.	Description	Value(s) Homogeneous	Value(s) Heterogeneous
N	Number of users	[16-1024]	[16-1024]
M	Number of VM instances	4	4
R	Number of resource types	3	3
C_1	Core capacity	500	500
C_2	Memory capacity	1000 GB	1000 GB
C_3	Storage capacity	500,000 GB	500,000 GB
w_{mr}	Amount of resource r provided by a VM instance m	as in Table II	as in Table III
k_{im}	Number of requested VM m by user i	[0, 20]	[0, 20]

B. Analysis of Results

We investigate the truthfulness of our proposed G-VMPAC-X mechanisms by analyzing the effects of untruthful declarations by a user. To show that our proposed mechanisms are robust against manipulation by a user, we consider three users requesting homogeneous VMs where their true types are $(< 5, 0, 0, 0 >, \$10)$, $(< 0, 4, 0, 0 >, \$24)$, and $(< 2, 0, 0, 2 >, \$20)$, respectively. The capacity of the three resources are as follows: 30 cores, 80 GB of memory, and 6000 GB of storage. The G-VMPAC-II calculates the efficiency of the users as 24.61, 32.98, and 11.59, respectively, then allocates resources to user 1 and 2 in the case that all users declare their true types. The payments of the winning users based on PAY are $\$4.71$ and $\$8.43$, respectively.

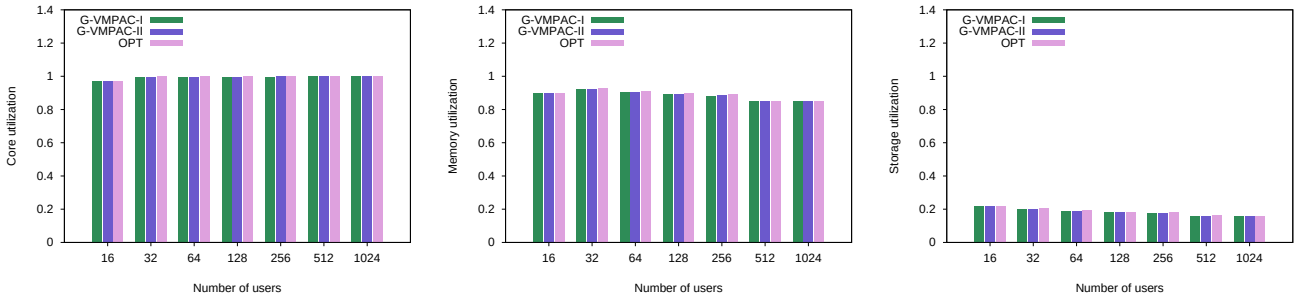
We assume that user 2 lies about her type $\hat{\theta}_2$. The consequence of such a declaration depends on her reported value v_2 and the bundle S_2. We consider different scenarios as shown in Table V, where user 2 does not reveal her true type. Case I is when the user declares her true type. In case II, when user 2 reports a value greater than her true type, she will still win and the mechanism determines the same payment for her as in case I. In case III, user 2 reports a value less than her true type, but not less than the price determined by our mechanism. In this case, the user is still winning, and pays the same amount as in case I. In case IV, user 2 reports a value below her determined payment. In this case, she will not get her requested bundle, and her utility is zero. In case V, she declares a larger bundle and still obtains the bundle due to available capacities. However, she pays more and her utility decreases. In case VI, she declares a larger bundle but becomes a loser since the cloud provider does not have enough resources to fulfill her requested bundle. As a result, her utility is zero. In all cases, the user can not increase her utility by declaring a type other than her true type.

We now compare the performance of G-VMPAC-X for different number of users. First, we analyze the performance of G-VMPAC-X in a homogeneous setting. Fig. 1a shows the social welfare for different number of users. The results show that different versions of G-VMPAC-X can obtain



(a) (b)

Figure 1: G-VMPAC-X performance (homogeneous VM instances case): (a) Social welfare; (b) Execution time.



(a) (b) (c)

Figure 2: G-VMPAC-X resource utilization (homogeneous VM instances case): (a) Cores; (b) Memory; (c) Storage.

Table V: Different scenarios for user 2's type declaration

Case	S_2	v_2	Scenario	Stat.	Pay.	Utility
I	$< 0, 4, 0, 0 >$	\$24	$\hat{v}_2 = v_2, \hat{S}_2 = S_2$	W	8.43	15.57
II	$< 0, 4, 0, 0 >$	\$30	$\hat{v}_2 > v_2, \hat{S}_2 = S_2$	W	8.43	15.57
III	$< 0, 4, 0, 0 >$	\$20	$\hat{v}_2 < v_2, \hat{S}_2 = S_2$	W	8.43	15.57
IV	$< 0, 4, 0, 0 >$	\$8	$\hat{v}_2 < v_2, \hat{S}_2 = S_2$	L	0	0
V	$< 1, 4, 0, 0 >$	\$24	$\hat{v}_2 = v_2, \hat{S}_2 > S_2$	W	9.38	14.61
VI	$< 0, 4, 0, 2 >$	\$24	$\hat{v}_2 = v_2, \hat{S}_2 > S_2$	L	0	0

almost the same social welfare as the optimal social welfare. Fig. 1b shows the execution time for cases with different number of users in a logarithmic scale. Fig. 2a, Fig. 2b, and Fig. 2c show the utilization of cores, memory and storage, respectively. In a homogeneous setting, different versions of G-VMPAC-X have similar social welfare and utilization. This is due to the fact that in all four VM types shown in Table II resource types increase proportionally. As a result, using scaling in different versions of G-VMPAC-X does not have a significant impact on the performance of G-VMPAC-X.

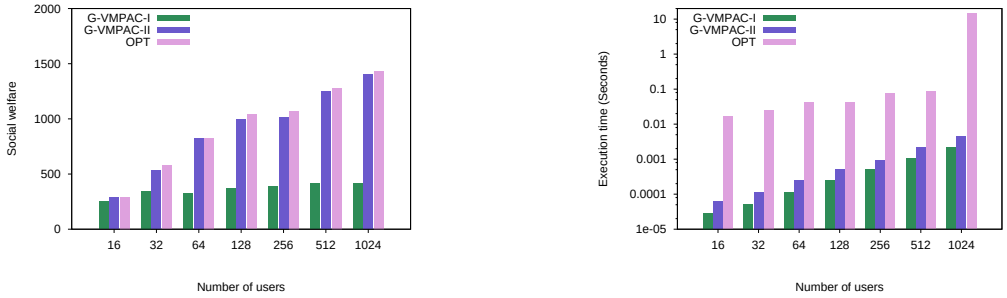
Now, we analyze the performance of G-VMPAC-X in a heterogeneous setting. Fig. 3a shows the social welfare for different number of users. The results show that G-VMPAC-II that uses scaling achieve a social welfare that is much close to the optimal, than G-VMPAC-I does. Since in this setting VM types are not related, using scaling in the greedy allocation algorithm is more beneficial. Fig. 3b shows the execution time for different number of users. G-VMPAC-X obtained the allocations much faster than the optimal algorithm. Comparing the execution time of algorithms in Fig. 1b and Fig. 3b we observe that in the heterogeneous setting the optimal algorithm needs much more time to execute than G-VMPAC-X does. For instance, on average for 1024 users, the optimal algorithm is 198.07 times slower in a heterogeneous setting than in a homogeneous setting, while for G-VMPAC-I and G-VMPAC-II this ratio is 2.17 and 3.22, respectively. Since the amounts of resources of each type in all four VM types shown in Table III are unrelated, the optimal algorithm needs more time to find the solution. Note that the VMPAC problem is strongly NP-hard.

Fig. 4a, Fig. 4b, and Fig. 4c show the utilization of cores, memory and storage, respectively. Using scaling in G-VMPAC-II keeps the utilization of resources closer to the one obtained in the optimal case. The storage utilization for G-VMPAC-I is very low (close to zero and not visible on the figure). The reason for that is that users that request storage have a very low efficiency since VMPAC-I does not use the scaling. As a result, the users requesting High-CPU and High-Memory instances are more likely to obtain their requested resources making it hard for the ones that request storage to obtain their requested VMs.

From all the above results, we conclude that G-VMPAC-II finds near-optimal solutions to the VMPAC problem and requires small execution times.

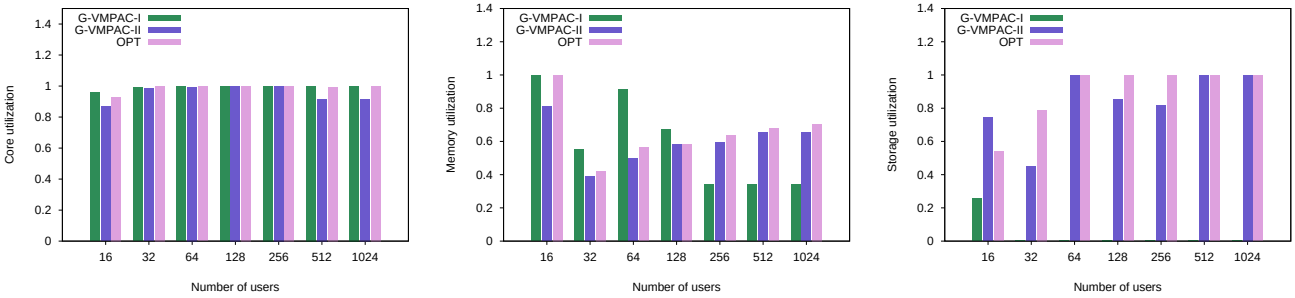
VI. Conclusion

We designed a family of truthful greedy mechanisms for solving the VMPAC problem in the presence of resources of multiple types. We determined the approximation ratio of the proposed mechanisms and investigated their properties



(a) (b)

Figure 3: G-VMPAC-X performance (heterogeneous VM instances case): (a) Social welfare; (b) Execution time.



(a) (b) (c)

Figure 4: G-VMPAC-X resource utilization (heterogeneous VM instances case: (a) Cores; (b) Memory; (c) Storage.

by performing extensive simulation experiments. The results showed that the proposed mechanisms determine near optimal allocations while giving the users incentives to report their true valuations for the bundles of VM instances. In addition, the execution time of the proposed mechanisms is very small. We plan to perform more experiments and implement a prototype allocation system in an experimental cloud computing system.

ACKNOWLEDGMENT

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

[4] 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.

[5] R. Gonen and D. Lehmann, "Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics," in *Proc. 2nd ACM Conf. on Electronic Commerce*, 2000, pp. 13–20.

[6] K. Leyton-Brown, Y. Shoham, and M. Tennenholtz, "An algorithm for multi-unit combinatorial auctions," in *Proc. of the National Conf. on Artificial Intelligence*, 2000, pp. 56–61.

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

[8] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. 4th USENIX Conference on Networked Systems Design & Implementation*, vol. 7, 2007, pp. 11–13.

[9] K. Gorlach and F. Leymann, "Dynamic service provisioning for the cloud," in *Proc. 9th IEEE Intl. Conf. on Services Computing*, 2012, pp. 555–561.

[10] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu, "Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach," in *Proc. 31st Intl. Conf. on Distrib. Comp. Syst.*, 2011, pp. 571–580.

[11] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proc. 31st Intl. Conf. on Distrib. Comp. Syst.*, 2011, pp. 559–570.

[12] 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.

[13] ——, "Combinatorial auction-based allocation of virtual machine instances in clouds," in *Proc. 2nd IEEE Intl. Conf. on Cloud Comp. Tech. and Sci.*, 2010, pp. 127–134.

[14] U. Lampe, M. Siebenhaar, A. Papageorgiou, D. Schuller, and R. Steinmetz, "Maximizing cloud provider profit from equilibrium price auctions," in *Proc. 5th IEEE Intl. Conf. on Cloud Comp.*, 2012, pp. 83–90.

[15] 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.

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