1-1-2014

Truthful Mechanisms For Resource Allocation And Pricing In Clouds

Mahyar Movahednejad
Wayne State University,

Truthful Mechanisms for Resource Allocation and Pricing in Clouds

by

Mahyar Movahed Nejad

THESIS

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

2014

MAJOR: COMPUTER SCIENCE

Approved by:

_____

Advisor Date

DEDICATION

To my parents Maryam and Ali, and my brother Mehran.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my advisor Dr. Grosu who introduced me to an exciting research area which ultimately resulted in this thesis. I am thankful for his mentorship, support and encouragement, and for everything he has taught me without which this thesis would have not been completed. Dr. Grosu has also been more than a thesis advisor for me, providing me with ongoing advice on broader research studies including my dissertation research.

I would like to thank my thesis committee members Dr. Fotouhi and Dr. Shi for their time and considerations.

I have been greatly privileged to work with my best friend Lena Mashayekhy during my graduate studies. Her brilliance contributed significantly to the improvement of our research projects.

I would like to express my sincere gratitude to my parents Maryam and Ali, and my brother Mehran. I always feel their endless love and support.

I will treasure the knowledge and personal growth that I have gained throughout my graduate studies, and I hope they empower me to do good with good analytics.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 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 thesis, we focus on IaaS where cloud providers offer different types of resources in the form of VM instances. IaaS providers such as Microsoft Azure [5] and Amazon Elastic Compute Cloud (Amazon EC2) [1] 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 [1]. 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 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 [43]) 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 and grids by employing game theory. Wei *et al.* [49] 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.* [19] designed an efficient truthful-in-expectation mechanism for resource allocation in clouds where only one type of resource was considered. Kong *et al.* [23] designed a stochastic mechanism to allocate resources among selfish VMs in a non-cooperative cloud environment. Mashayekhy and Grosu [29, 31, 33] investigated the problem of federating resources in grids by employing coalitional game theory and designed grid federation formation mechanisms. They also studied the problem of federating resources in grids considering the trust relationship among grid service providers [32]. Mashayekhy and Grosu [30] addressed the problem of federation formation in clouds and designed a coalitional game-based mechanism that enables the cloud providers to dynamically form a cloud federation maximizing their profit. Wang *et al.* [48] 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.* [15] 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., [21, 50, 53, 54]). In these studies, only one type of resource (i.e., the spectrum) is available for allocation. However, in this thesis, 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.* [53] 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 [50] 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.* [20] 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.* [20] 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 [51, 52]. They proposed a combinatorial auction-based mechanism, CA-GREEDY, to allocate VM instances in clouds [52]. 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 [51]. 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.

The design of truthful mechanisms for several classes of combinatorial optimization problems was initiated by Nisan and Ronen [42]. Efficiently computable truthful mechanisms for several problems has since been proposed and investigated by Archer and Tardos [7], Mu'alem and Nisan [37], and Awerbuch *et al.* [9]. The reader is referred to Nisan *et al.* [43] for a comprehensive introduction to mechanism design. Rothkopf *et al.* [44] were the first to investigate the winner determination problem and the complexity of solving combinatorial auctions. Sandholm [45] proved that solving the winner determination problem is NP-complete. Zurel and Nisan [55] presented a heuristic algorithm for combinatorial auctions. For a detailed survey on combinatorial auctions the reader is referred to [14]. Lehmann *et al.* [25] proposed a greedy truthful mechanism for single-unit combinatorial auctions where all items are non-identical. However, our focus is on the design of greedy truthful mechanisms in multi-unit settings. In a multi-unit combinatorial auction, there exists many types of items and many identical items of each type. Several studies focused on finding solutions for multi-unit combinatorial auctions without considering the truthfulness [18, 26]. Bartal *et al.* [10] proposed a truthful mechanism for multi-unit combinatorial auctions where each item has a fixed number of units. This is not the case in clouds due to the fact that the resources are provisioned dynamically based on the user requests, and the number of VMs are not known.

The VM provisioning and allocation problem considering multiple types of resources can be formulated as a multidimensional knapsack problem which is a class of General Assignment Problem (GAP). Mansini and Speranza [28] proposed an exact algorithm for solving

small size multidimensional knapsack problems. Their approach is based on the optimal solution of subproblems by improving the efficiency of the branch-and-bound method for the integer program formulation. Shmoys and Tardos [47] proposed a 2-approximation for GAP without considering the truthfulness property. There are several studies on designing truthful mechanisms for GAP. Aggarwal and Hartline [6] designed truthful mechanisms for auctions by modeling them as knapsack problems. Greedy algorithms for solving the multidimensional knapsack problem (MKP) have been extensively studied by Kellerer *et al.* [22]. However, none of these studies considered the design of truthful mechanisms.

Several researchers investigated the problem of VM provisioning in clouds from different points of view and applied various methodologies to solve it. Calheiros *et al.* [12] designed a provisioning technique using a queuing network system model that dynamically adapts to workload changes related to applications. Their prediction-based approach determines the number of allocated VMs. Bi *et al.* [11] proposed a dynamic provisioning technique for multi-tier applications in cloud based on queuing networks. However, their objective is to minimize the total number of VM instances allocated to the users based on their request specifications. Ellens *et al.* [16] analyzed the problem of allocating resources to different users with multiple service request classes. They modeled a cloud provider using a queuing system with different priority classes. Fang *et al.* [17] proposed a Cloud Resource Prediction and Provisioning scheme (RPPS) that predicts the future demand using the ARIMA model and performs proactive resource provisioning for cloud applications. In RPPS, a cloud provider can dynamically add VMs. Lampe *et al.* [24] proposed a heuristic approach considering several types of resources. However, they did not propose a truthful mechanism. Chaisiri *et al.* [13] proposed an optimal cloud resource provisioning algorithm to minimize the total cost of provisioning resources. They modeled the problem as a stochastic integer program, where there are several cloud providers and one consumer. The algorithm considers all possible combinations of VMs. Being a NP-hard problem, the algorithm for solving it does not scale well to larger input sizes. Shi *et al.* [46] formulated the problem of VM allocation as a Mixed Integer Program by considering fixed price VM instances. However, they concluded that this optimal approach is not practical for medium

and large problems. The focus was on maximizing the cloud provider's profit without considering the users' incentives for manipulating the allocation mechanisms by untruthful reporting.

1.3 Organization

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

CHAPTER 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 2.1, we present the four types of VM instances offered by Amazon EC2 at the time of writing this thesis. 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:

$$V = \sum_{i \in \mathcal{U}} v_i(S_i) \cdot x_i \tag{2.1}$$

Table 2.1: 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

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 [43]. 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:

$$\text{Maximize } V \tag{2.2}$$

Subject to:

$$\sum_{i \in \mathcal{U}} \sum_{m \in \mathcal{VM}} k_{im} w_{mr} x_i \leq C_r, \forall r \in \mathcal{R} \tag{2.3}$$

$$x_i = \{0, 1\}, \forall i \in \mathcal{U} \tag{2.4}$$

The solution to this problem is a vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ maximizing the social welfare. Constraints (2.3) ensure that the allocation of each resource type does not exceed the available capacity of that resource. Constraints (2.4) 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) [22], 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.

CHAPTER 3: MECHANISM DESIGN FRAMEWORK

In this chapter, 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:

$$v_i(\mathcal{A}_i) = \begin{cases} b_i & \text{if } S_i \subseteq \mathcal{A}_i \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

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 [43]) 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 [37].

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 $\hat{\theta}'_i \succeq \hat{\theta}_i$.*

Algorithm 1 VCG-VMPAC Mechanism

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

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:

$$\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = \begin{cases} v_i^c & \text{if } i \text{ wins} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

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

In the next section, 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 $\mathcal{A}$ [43]. A VCG mechanism [43] 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*

$$\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = \sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}_{-i})} \hat{b}_j - \sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}), j \neq i} \hat{b}_j, \tag{3.3}$$

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 (3.3).*

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

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: **end for**
5: {Allocation}
6: $(V^*, \mathbf{x}^*)$ = G-VMPAC-X-ALLOC$(\hat{\boldsymbol{\theta}}, \mathbf{C})$
7: Provisions and allocates VM instances according to $\mathbf{x}^*$.
8: {Payment}
9: $\mathcal{P}$ =PAY$(\hat{\boldsymbol{\theta}}, \mathbf{C}, \mathbf{x})$

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 [43], 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 chapter.

CHAPTER 4: TRUTHFUL GREEDY MECHANISMS

In this chapter, 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 $P = NP$ [22]. 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 chapter. Once the allocation is determined, the mechanism provisions the required number and types of VM instances (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 in-

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: $\hat{\mathbf{C}} = \mathbf{C}$
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: **end for**
9: **for all** $i \in \mathcal{U}$ **do**
10: $d_i = \frac{\hat{b}_i}{\sqrt{\sum_{r=1}^{R} f_r \hat{a}_{ir}}}$
11: **end for**
12: Sort $\mathcal{U}$ in decreasing order of d_i
13: **for all** $i \in \mathcal{U}$ **do**
14: $flag \leftarrow$ `TRUE`
15: **for all** $r \in \mathcal{R}$ **do**
16: $\tilde{C}_r = \hat{C}_r - \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$
17: **if** $\tilde{C}_r < 0$ **then**
18: $flag \leftarrow$ `FALSE`
19: **break**;
20: **end if**
21: **end for**
22: **if** $flag$ **then**
23: $V = V + \hat{b}_i$
24: $x_i = 1$
25: $\hat{\mathbf{C}} = \tilde{\mathbf{C}}$
26: **end if**
27: **end for**
28: **Output:** V, $\mathbf{x}$

stances to the users. The algorithm orders the users (lines 6-10) according to a general *density* metric defined as:

$$d_i = \frac{\hat{b}_i}{\sqrt{\sum_{r=1}^{R} f_r \hat{a}_{ir}}}, \forall i \in \mathcal{U} \tag{4.1}$$

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.* [25]. 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-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

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: $\quad \mathcal{P}_i = 0$
6: $\quad$ **if** x_i^* **then**
7: $\quad\quad l = -1$
8: $\quad\quad (V'^*, \mathbf{x}'^*) =$ G-VMPAC-X-ALLOC $(\hat{\boldsymbol{\theta}} \setminus \hat{\theta}_i, \mathbf{C})$
9: $\quad\quad$ **for all** $j \in \mathcal{U}, d_j < d_i$ in decreasing order of d_j **do**
10: $\quad\quad\quad$ **if** $x_j^* = 0$ **and** $x_j'^*$ **then**
11: $\quad\quad\quad\quad l = j$
12: $\quad\quad\quad\quad$ **break**;
13: $\quad\quad\quad$ **end if**
14: $\quad\quad$ **end for**
15: $\quad\quad$ **if** l **then**
16: $\quad\quad\quad \mathcal{P}_i = d_l \sqrt{\sum_{r=1}^{R} f_r \hat{a}_{ir}}$
17: $\quad\quad$ **else**
18: $\quad\quad\quad \mathcal{P}_i = 0$
19: $\quad\quad$ **end if**
20: $\quad$ **end if**
21: **end for**
22: **Output:** $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

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).

4.2 G-VMPAC-X Mechanisms Properties

In this section, 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.*

Proof. We show that the algorithms that are part of G-VMPAC-X-ALLOC 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., $\hat{b}'_i > \hat{b}_i$. This leads to a higher density, $d'_i > d_i$ 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-ALLOC 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, $a'_{ir} < a_{ir}, \forall r \in \mathcal{R}$. It is easy to see that in G-VMPAC-I-ALLOC and G-VMPAC-II-ALLOC, this leads to a higher density for user i, that is, $d'_i > d_i$. We now show that $d'_i > d_i$ for G-VMPAC-III. In the G-VMPAC-III case, $\sum_{i'=1}^{N} a_{i'r} - a_{ir}$ and C_r have the same values in both type declarations ($\hat{\theta}_i$ and $\hat{\theta}'_i$). In the following, we denote by $\alpha = \sum_{i'=1}^{N} a_{i'r} - a_{ir}$. Since $a'_{ir} < a_{ir}$, by multiplying with $-C_r$ both sides of the inequality, we obtain $-C_r a'_{ir} > -C_r a_{ir}$. Then, by adding $\alpha(\alpha + a_{ir} + a'_{ir} - C_r)$ and $a_{ir} a'_{ir}$ on both sides of the inequality, we obtain $(\alpha + a'_{ir})(\alpha + a_{ir} - C_r) > (\alpha + a_{ir})(\alpha + a'_{ir} - C_r)$, therefore $\frac{\alpha + a_{ir} - C_r}{\alpha + a_{ir}} > \frac{\alpha + a'_{ir} - C_r}{\alpha + a'_{ir}}$. Thus, $f_r > f'_r$, which implies $d'_i > d_i$.

iii) User i declares a higher value, $\hat{b}'_i > \hat{b}_i$, 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 allocation algorithms in the G-VMPAC-X-ALLOC family are monotone. □

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.*

Proof. To prove that PAY determines the critical value payment for the users, we need to show that $\mathcal{P}_i$ is the critical value for user i. Note that $\mathcal{P}_i = d_l\sqrt{\sum_{r=1}^{R} f_r a_{ir}}$, where l is the index of user l who would have won if user i did not participate, and she appears after user i based on the decreasing order of the density metric. We separate the proof into two cases as follows:

i) User i declares a higher value than $\mathcal{P}_i$, (i.e., $\hat{b}'_i > \mathcal{P}_i$). In this case, she wins and pays the same amount $\mathcal{P}_i$.

ii) User i declares a lower value than $\mathcal{P}_i$, (i.e., $\hat{b}'_i < \mathcal{P}_i$). This leads to a density

$$d'_i = \frac{\hat{b}'_i}{\sqrt{\sum_{r=1}^{R} f_r a_{ir}}} < \frac{\mathcal{P}_i}{\sqrt{\sum_{r=1}^{R} f_r a_{ir}}}.$$

Since $\mathcal{P}_i = d_l\sqrt{\sum_{r=1}^{R} f_r a_{ir}}$, we obtain

$$d'_i < \frac{d_l\sqrt{\sum_{r=1}^{R} f_r a_{ir}}}{\sqrt{\sum_{r=1}^{R} f_r a_{ir}}} = d_l.$$

As a result, $d'_i < d_l$, and user i is not a winning user.

These show that the payment $\mathcal{P}_i$ is the minimum valuation that user i must bid to obtain her required bundle. In the case in which user i is not a winner, she pays 0, thus, satisfying the properties of the critical value payment. As a result, the payment determined by PAY is the critical value payment. □

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 [37], the mechanisms in the G-VMPAC-X family are truthful. □

In the following, we analyze the effect of untruthful reporting on the utility of the users participating in the G-VMPAC-II mechanism by considering an example. The behavior of the other two proposed mechanisms is similar and we do not present them here. To show that our proposed mechanism, G-VMPAC-II, is robust against manipulation by a user, we consider three users requesting VM instances of the type given in Table 1. The true types of the three users are as follows: user 1, ($< 5, 0, 0, 0 >$, \$10); user 2, ($< 0, 4, 0, 0 >$, \$25) and user 3, ($< 2, 0, 0, 2 >$, \$15). 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 density for each of the users as 15.68, 29.31, and 11.73, 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 determined by G-VMPAC-II are \$7.47 and \$10.0, respectively.

We assume that user 2 lies about her type $\hat{\theta}_2$. The consequence of such a declaration depends on her reported value b_2 and the bundle S_2. We consider different scenarios as shown in Table 4.1, where user 2 does not reveal her true type. Case I is when user 2 declares her true type. In case II, user 2 reports a value greater than her true type and she still wins, 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, and thus, the truthfulness property is satisfied.

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* $C_{max} = \max_{r \in \mathcal{R}} C_r$.

Table 4.1: Different scenarios for user 2's type declaration

Case	S_2	b_2	Scenario	Stat.	Pay.	Utility
I	$< 0, 4, 0, 0 >$	\$25	$\hat{b}_2 = b_2$, $\hat{S}_2 = S_2$	W	10.0	15.0
II	$< 0, 4, 0, 0 >$	\$30	$\hat{b}_2 > b_2$, $\hat{S}_2 = S_2$	W	10.0	15.0
III	$< 0, 4, 0, 0 >$	\$20	$\hat{b}_2 < b_2$, $\hat{S}_2 = S_2$	W	10.0	15.0
IV	$< 0, 4, 0, 0 >$	\$5	$\hat{b}_2 < b_2$, $\hat{S}_2 = S_2$	L	0	0
V	$< 1, 4, 0, 0 >$	\$25	$\hat{b}_2 = b_2$, $\hat{S}_2 > S_2$	W	10.55	14.45
VI	$< 0, 4, 0, 3 >$	\$25	$\hat{b}_2 = b_2$, $\hat{S}_2 > S_2$	L	0	0

Proof. 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^*.

$$\hat{V}^* = \sum_{i \in \hat{X}^*} \hat{b}_i \leq \alpha \sum_{i \in \hat{X}} \hat{b}_i = \alpha \hat{V} \tag{4.2}$$

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, $\sum_{i \in \hat{X}^*} \hat{b}_i \leq \sum_{i \in \bigcup_{j \in \hat{X}} D_j} \hat{b}_i \leq \alpha \sum_{i \in \hat{X}} \hat{b}_i$. Therefore, it is sufficient to show for every $i \in \hat{X}$ that: $\sum_{j \in D_i} \hat{b}_j \leq \alpha \hat{b}_i$. Note that every $j \in D_i$ appeared after i in the greedy order, and thus, $d_j \leq d_i$. Then,

$$\hat{b}_j \leq \frac{\hat{b}_i \sqrt{\sum_{r=1}^{R} f_r a_{jr}}}{\sqrt{\sum_{r=1}^{R} f_r a_{ir}}} \tag{4.3}$$

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

$$\sum_{j \in D_i} \hat{b}_j \leq \sum_{j \in D_i} \frac{\hat{b}_i \sqrt{\sum_{r=1}^{R} f_r a_{jr}}}{\sqrt{\sum_{r=1}^{R} f_r a_{ir}}}$$
$$\leq \frac{\hat{b}_i}{\sqrt{\sum_{r=1}^{R} f_r a_{ir}}} \sum_{j \in D_i} \sqrt{\sum_{r=1}^{R} f_r a_{jr}} \tag{4.4}$$

Using the Cauchy-Schwarz inequality on $\sum_{j \in D_i} \sqrt{\sum_{r=1}^{R} f_r a_{jr}}$ of Equation (4.4), we have:

$$\left(\sum_{j \in D_i} \sqrt{\sum_{r=1}^{R} f_r a_{jr}} \right)^2 \leq \sum_{j \in D_i} 1 \sum_{j \in D_i} \sum_{r=1}^{R} f_r a_{jr} \tag{4.5}$$

Using square root on both sides we obtain:

$$\sum_{j \in D_i} \sqrt{\sum_{r=1}^{R} f_r a_{jr}} \leq \sqrt{\sum_{j \in D_i} 1} \sqrt{\sum_{j \in D_i} \sum_{r=1}^{R} f_r a_{jr}} \tag{4.6}$$

Replacing this in Equation (4.4) we obtain

$$\sum_{j \in D_i} \hat{b}_j \leq \frac{\hat{b}_i}{\sqrt{\sum_{r=1}^{R} f_r a_{ir}}} \sqrt{\sum_{j \in D_i} 1} \sqrt{\sum_{j \in D_i} \sum_{r=1}^{R} f_r a_{jr}} \tag{4.7}$$

We now consider each of the three mechanisms individually and determine their approximation ratios.

G-VMPAC-I: Since X^* is an allocation and $f_r = 1$ for G-VMPAC-I we have

$$\sum_{j \in D_i} \sum_{r=1}^{R} a_{jr} \leq \sum_{r=1}^{R} C_r \leq R C_{max} \tag{4.8}$$

where $C_{max} = \max_{r \in \mathcal{R}} C_r$. Replacing this in Equation (4.7) we obtain

$$\sum_{j \in D_i} \hat{b}_j \leq \frac{\hat{b}_i \sqrt{RC_{max}}}{\sqrt{\sum_{r=1}^{R} a_{ir}}} \sqrt{\sum_{j \in D_i} 1} \tag{4.9}$$

The worst case is when $\sum_{r=1}^{R} a_{ir}$ has the minimum value, which is 1. In addition, we have $\sqrt{\sum_{j \in D_i} 1} = \sqrt{N}$. Therefore,

$$\sum_{j \in D_i} \hat{b}_j \leq \hat{b}_i \sqrt{NRC_{max}} \tag{4.10}$$

As a result, the approximation ratio is $\alpha = \sqrt{NRC_{max}}$.

G-VMPAC-II: Since X^* is an allocation and $f_r = \frac{1}{C_r}$ for G-VMPAC-II we have

$$\sum_{j \in D_i} \sum_{r=1}^{R} \frac{a_{jr}}{C_r} \leq R \tag{4.11}$$

Replacing this in Equation (4.7) we obtain

$$\sum_{j \in D_i} \hat{b}_j \leq \frac{\hat{b}_i \sqrt{R}}{\sqrt{\sum_{r=1}^{R} \frac{a_{ir}}{C_r}}} \sqrt{\sum_{j \in D_i} 1} \tag{4.12}$$

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 $C_{max} = \max_{r \in \mathcal{R}} C_r$. In addition, $\sqrt{\sum_{j \in D_i} 1} = \sqrt{N}$. Therefore,

$$\sum_{j \in D_i} \hat{b}_j \leq \hat{b}_i \sqrt{NRC_{max}} \tag{4.13}$$

As a result, the approximation ratio is $\alpha = \sqrt{NRC_{max}}$.

G-VMPAC-III: Since X^* is an allocation and $f_r = \frac{\sum_{i=1}^{N} a_{ir} - C_r}{\sum_{i=1}^{N} a_{ir}}$ for G-VMPAC-III we have

$$\sum_{j \in D_i} \sum_{r=1}^{R} \frac{\sum_{i=1}^{N} a_{ir} - C_r}{\sum_{i=1}^{N} a_{ir}} a_{jr} \leq RC_{max} \tag{4.14}$$

where $C_{max} = \max_{r \in \mathcal{R}} C_r$. Replacing this in Equation (4.7) we obtain

$$\sum_{j \in D_i} \hat{b}_j \leq \frac{\hat{b}_i \sqrt{RC_{max}}}{\sqrt{\sum_{r=1}^{R} \frac{\sum_{i=1}^{N} a_{ir} - C_r}{\sum_{i=1}^{N} a_{ir}} a_{ir}}} \sqrt{\sum_{j \in D_i} 1} \tag{4.15}$$

The worst case is when $\sum_{r=1}^{R} \frac{\sum_{i=1}^{N} a_{ir} - C_r}{\sum_{i=1}^{N} a_{ir}} a_{ir}$ has the minimum value, which is 1. In addition, $\sqrt{\sum_{j \in D_i} 1} = \sqrt{N}$.

$$\sum_{j \in D_i} \hat{b}_j \leq \hat{b}_i \sqrt{NRC_{max}} \tag{4.16}$$

As a result, the approximation ratio is $\alpha = \sqrt{NRC_{max}}$.

Thus, the mechanisms from G-VMPAC-X family achieve an approximation ratio of $\sqrt{NRC_{max}}$. □

In our previous work [40], we proposed two greedy mechanisms having an approximation ratio of RC_{max}. In this thesis, our proposed mechanisms obtain a better approximation ratio than that of [40] 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 chapter. 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 Theorem 4, we obtain an approximation ratio for the mechanisms in the G-VMPAC-X family by analyzing the extreme worst case scenario. 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).

Following the proof of Theorem 4, let $\hat{X}^*$ be the set of users in the optimal solution, and $\hat{X}$ be the set of users in the obtained solution by G-VMPAC-X, where $\hat{X} \cap \hat{X}^* = \emptyset$. We consider a set of users D_i for user i such that if $j \in D_i$ then $j \geq i$ (based on the order) and $j \in \hat{X}^*$ but $j \notin \hat{X}$ because of user i. Meaning that, user i blocks each user in D_i from entering $\hat{X}$. However, based on the bound on the number of VM instances available for a user imposed by the cloud provider, user i can only block a given number q of users. As a result, $\sum_{j \in D_i} 1 = q$, where q is a constant. That is, every user in the greedy approach can block a constant number of users from the optimal solution to appear in the greedy solution. In addition, in order for user i to be able to block q users, its request should be $a_{ir} \simeq q a_{jr}, \forall r$. Note that this is the worst case scenario given the fact that requests are within a given range.

We now consider the general form for the three mechanisms, and determine their approximation ratio for the practical cases described above. We continue from Equation (4.7). By replacing $\sum_{j \in D_i} 1$ and a_{ir} in Equation (4.7), we have:

$$\sum_{j \in D_i} \hat{b}_j \leq \frac{\hat{b}_i}{\sqrt{\sum_{r=1}^{R} f_r q a_{jr}}} \sqrt{q} \sqrt{q \sum_{r=1}^{R} f_r a_{jr}} \tag{4.17}$$

Then,

$$\sum_{j \in D_i} \hat{b}_j \leq \hat{b}_i \sqrt{q} \tag{4.18}$$

As a result, the approximation ratio achieved by the G-VMPAC-X mechanisms is $\sqrt{q}$ for instances of the VMPAC problem encountered in practice, where q is a constant.

CHAPTER 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 [51]. 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 [3].

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 [2] and the Parallel Workloads Archive [4]. We present statistics of the logs in Table 5.1. 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 chapter, 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 [2] and the Parallel Workloads Archive [4]. From the Grid Workloads Archive, we selected four out of six available logs. 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

Table 5.1: 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)
GWA-T-1 DAS-2	81	[1-128]	[1-4,295]	[10-51,070]	247	760	2,500
GWA-T-3 NorduGrid	34	1	[1-2,147]	[10-1,053,072]	24	14,000	640,000
GWA-T-4 AuverGrid	33	1	[1.7-3,668]	[10-259,316]	7	8,800	640,000
GWA-T-10 SHARCNET	147	[1-3000]	[1-32,021]	[10-2,087,029]	85	9,700	4,000
METACENTRUM-2009-2	42	[1-60]	[1-61,538]	[10-2,592,130]	44	9,700	178,000
PIK-IPLEX-2009-1	36	[1-2560]	[1-29,360]	[10-4,815,007]	88	89,000	470,000

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 5.2 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

Table 5.2: 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

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.

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 the large scale experiments.

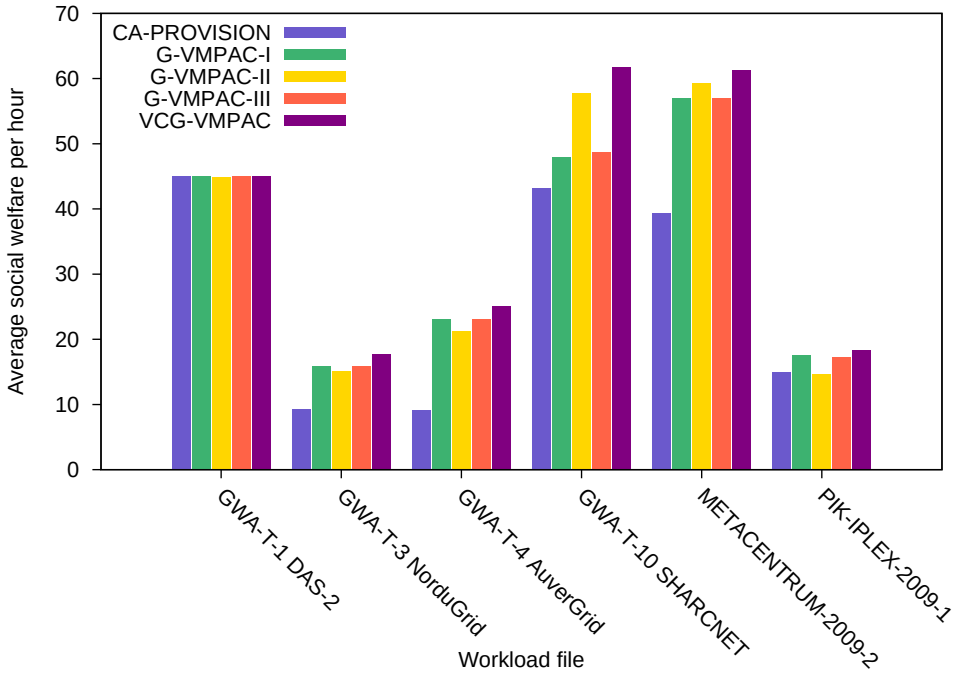

Figure 5.1: G-VMPAC-X performance (small scale experiments): Social welfare

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 the small scale experiments considering only 500 hours of the logs to show the performance of all mechanisms.

5.2.1 Small-scale

We compare the performance of the mechanisms by considering small scale experiments consisting of only 500 auction hours from the selected logs.

Fig. 5.1 shows that the achieved average social welfare per hour for the proposed mechanisms is very close to that of the optimal mechanism, VCG-VMPAC. However, as shown in Fig. 5.2, the execution time of our proposed mechanisms, G-VMPAC-X, are about four to five orders of magnitude lower than that of VCG-VMPAC. Note that this is only for the first 500 auction hours. Fig. 5.3 shows the average revenue per hour achieved by the cloud provider using the mechanisms. G-VMPAC-II achieves the highest revenue among all the

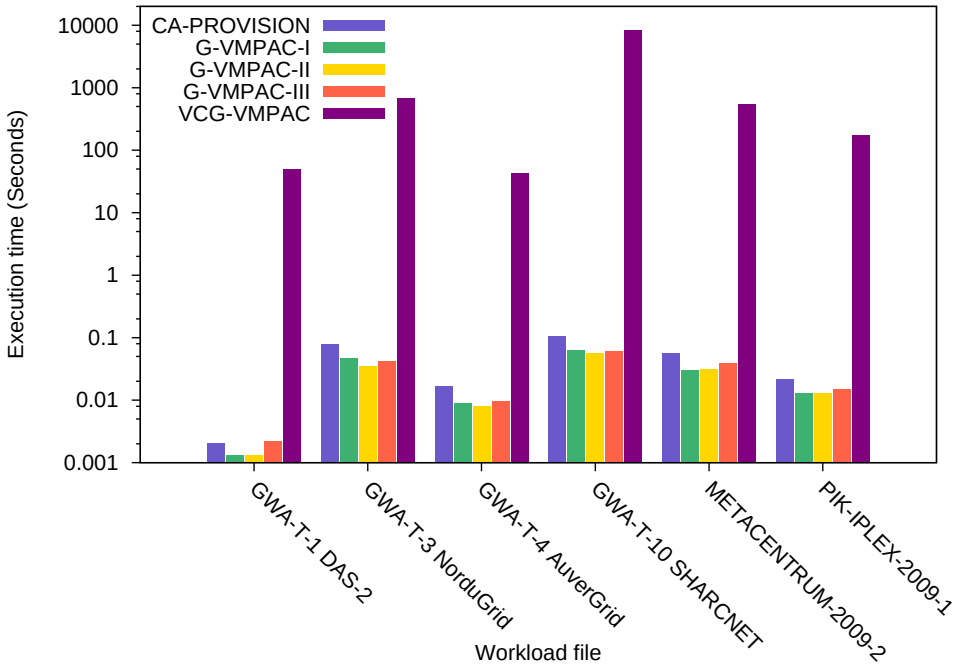

Figure 5.2: G-VMPAC-X performance (small scale experiments): Execution time

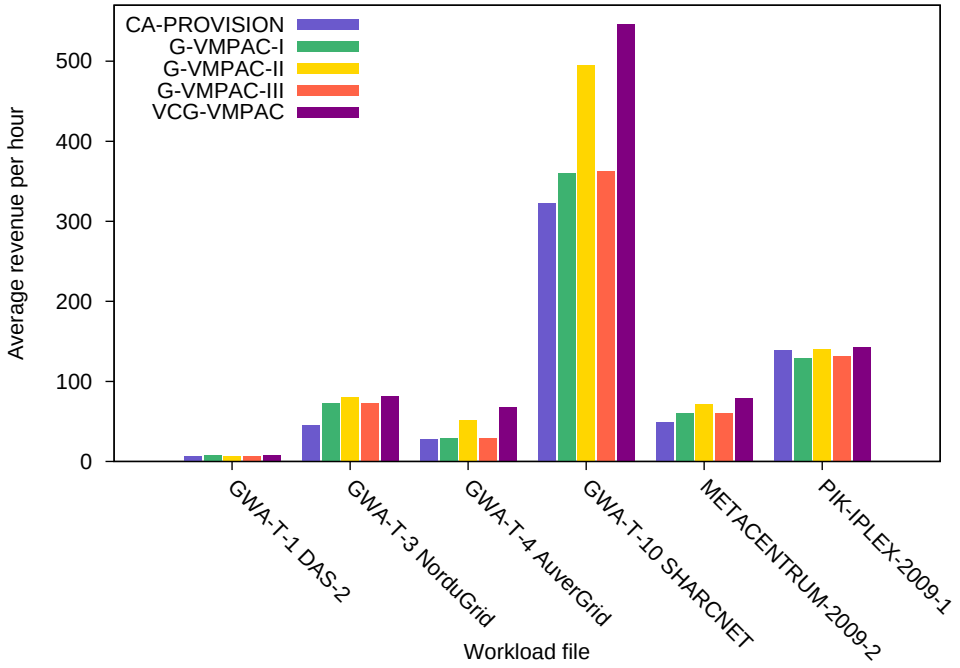

Figure 5.3: G-VMPAC-X performance (small scale experiments): Revenue mechanisms for all workloads.

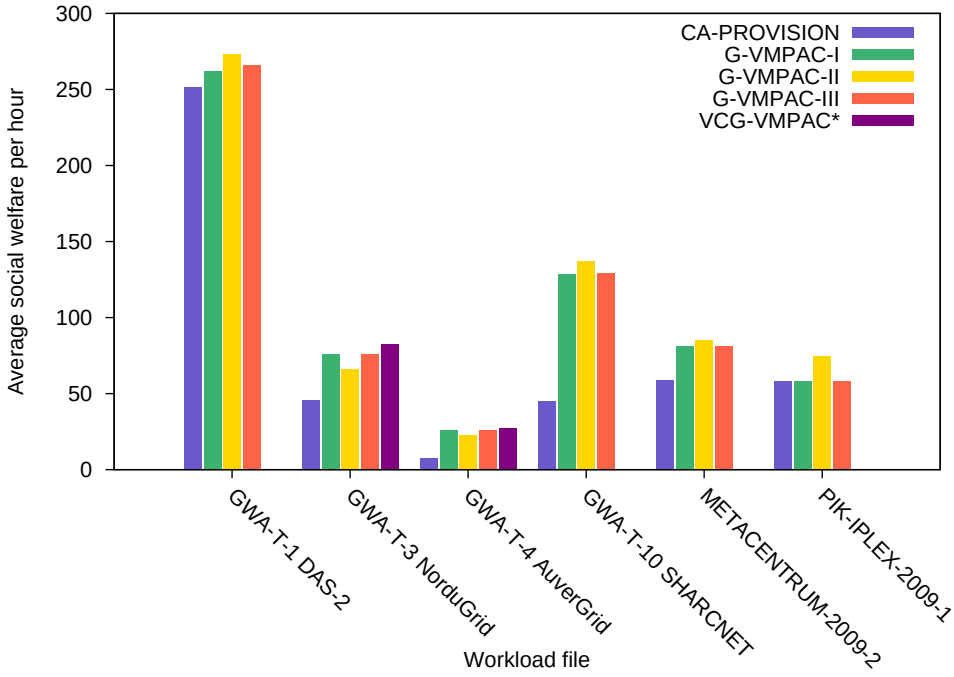

Figure 5.4: G-VMPAC-X performance: Social welfare (*VCG-VMPAC was not able to determine the allocation for GWA-T-1 DAS-2, GWA-T-10 SHARCNET, METACENTRUM-2009-2, and PIK-IPLEX-2009-1 in feasible time, and thus, there are no bars in the plots in Figs. 1 to 7 for those cases)

5.2.2 Large-scale

First, we analyze the performance of the mechanisms in terms of social welfare. Fig. 5.4 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.

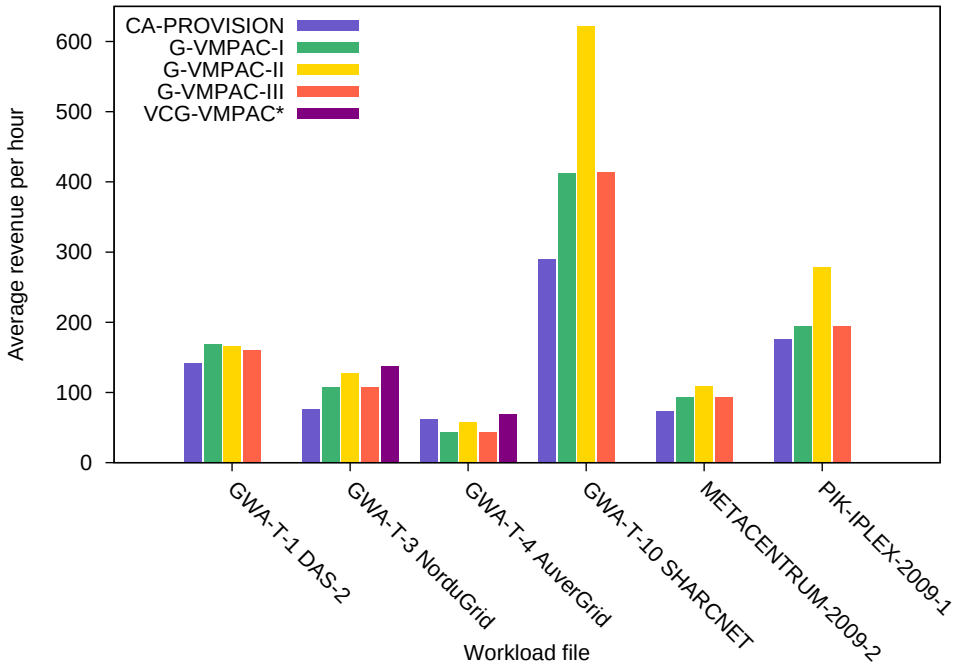

Figure 5.5: G-VMPAC-X performance: Revenue

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-

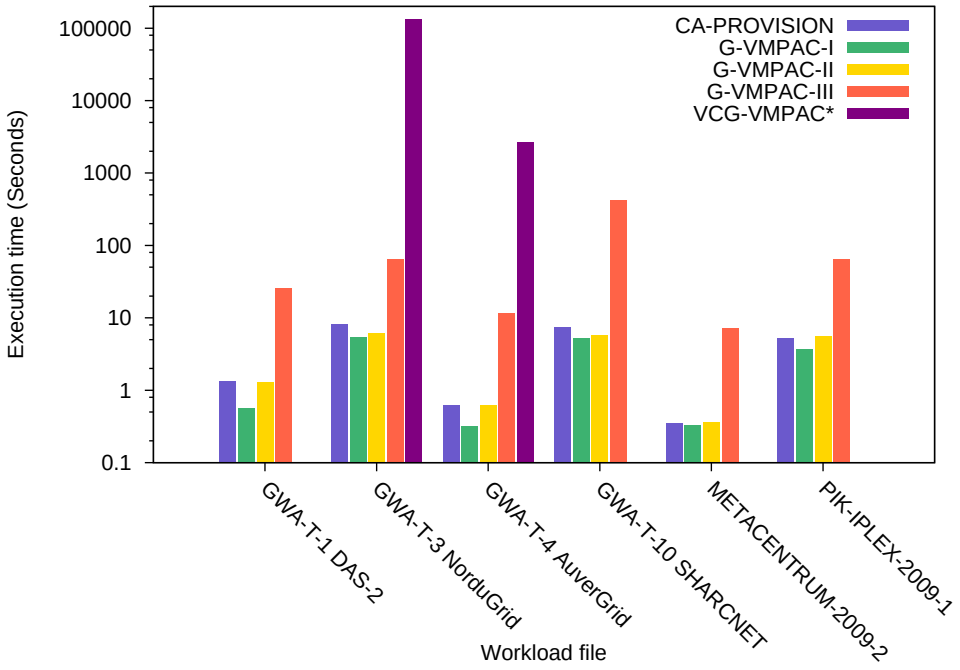

Figure 5.6: G-VMPAC-X performance: Execution time

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. 5.5 shows the average revenue per hour achieved by the cloud provider when using the five mechanisms. 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. 5.6 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

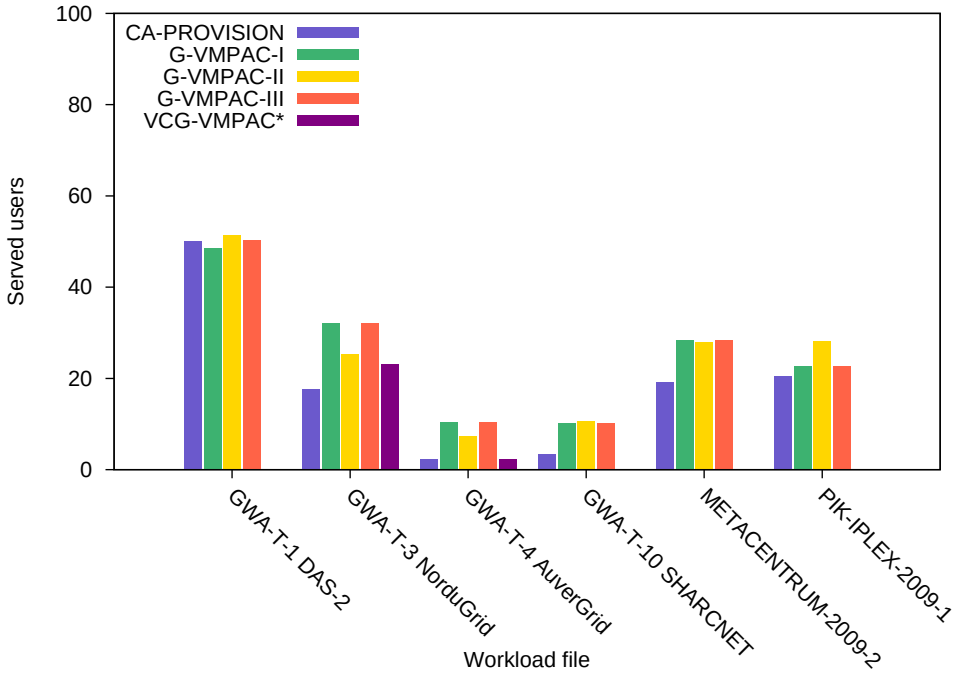

Figure 5.7: G-VMPAC-X performance: Users served.
(*see Fig. 1 note on VCG-VMPAC)

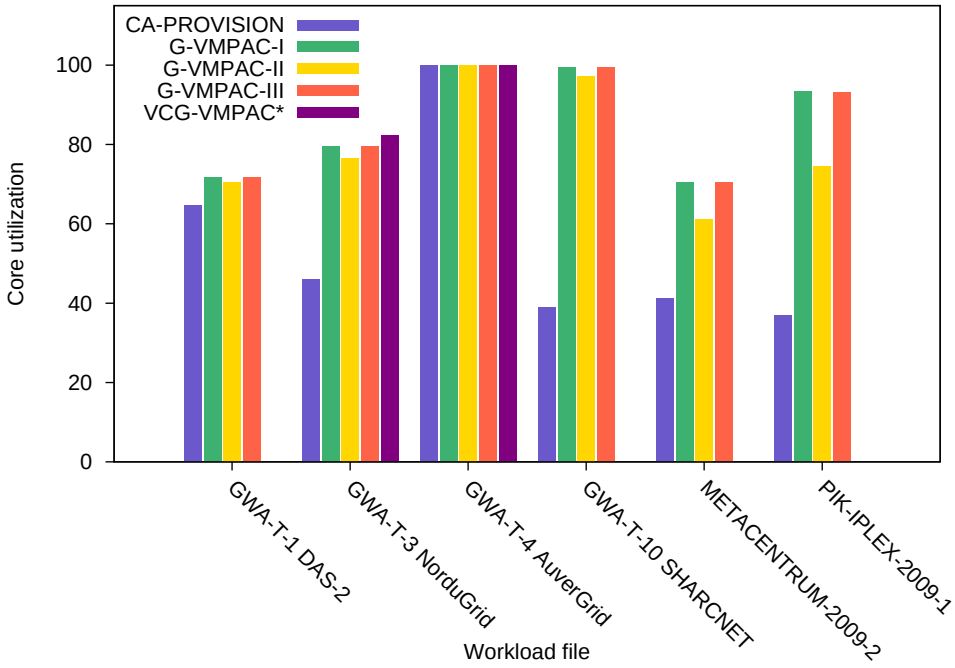

Figure 5.8: G-VMPAC-X core utilization (*see Fig. 1 note on VCG-VMPAC)

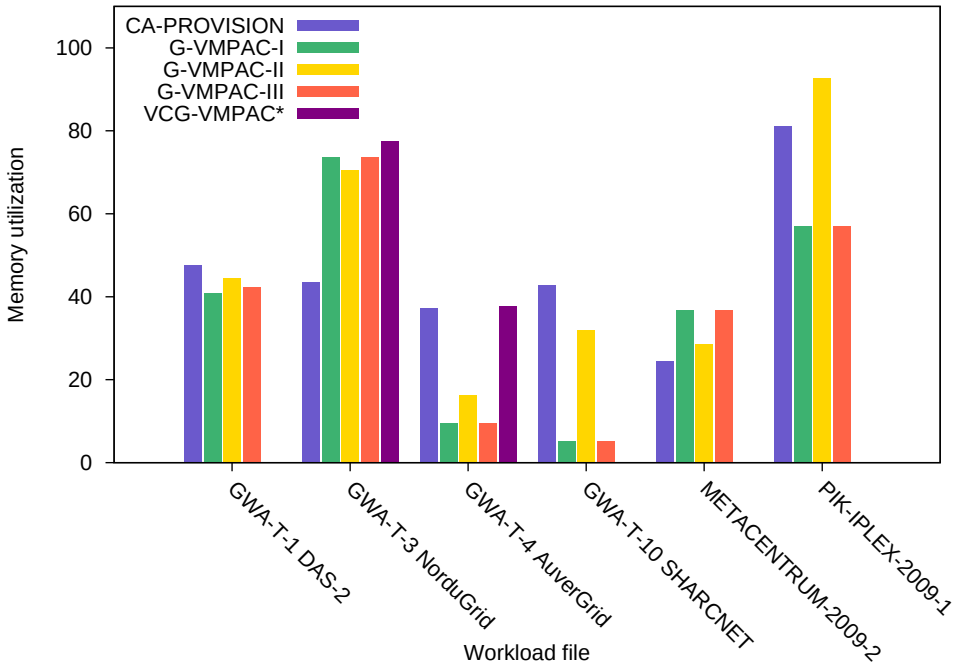

Figure 5.9: G-VMPAC-X memory utilization (*see Fig. 1 note on VCG-VMPAC)

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. 5.7 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. 5.8 to 5.10 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

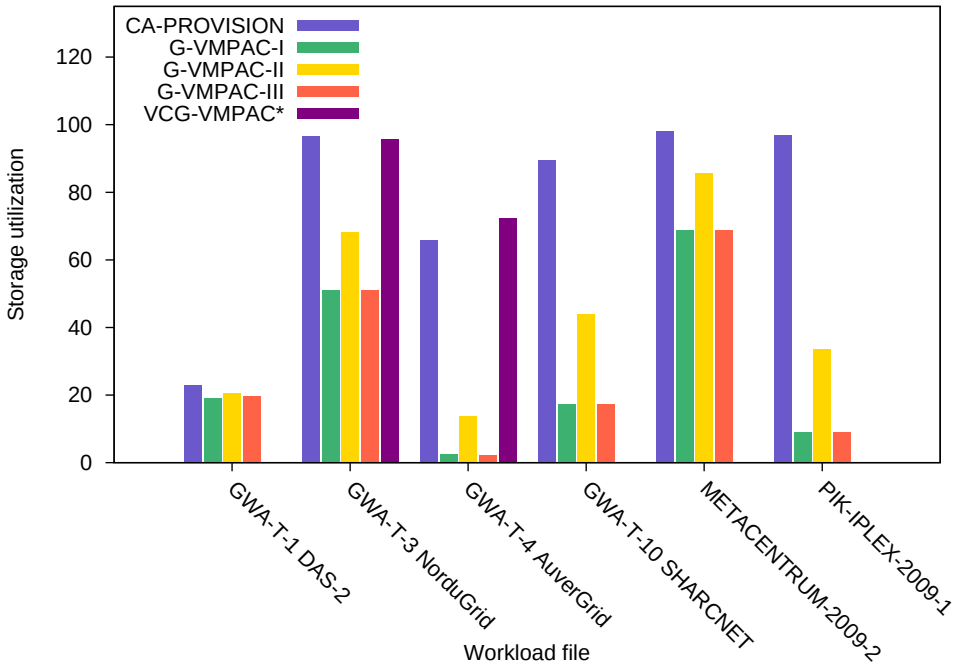

Figure 5.10: G-VMPAC-X storage utilization (*see Fig. 1 note on VCG-VMPAC)

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.

CHAPTER 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.

List of Publications

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

   M. Nejad, L. Mashayekhy and D. Grosu

   IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol. 25, 2014. [38]

2. **Energy-aware Scheduling of MapReduce Jobs**

   L. Mashayekhy, M. Nejad, D. Grosu, D. Lu, W. Shi

   Proc. of the 3rd IEEE International Congress on Big Data (BigData'14) - Research Track, Alaska, USA, June 2014. [34]

3. **Incentive-Compatible Online Mechanisms for Resource Provisioning and Allocation in Clouds**

   L. Mashayekhy, M. Nejad, D. Grosu, A. Vasilakos

   Proc. of the 7th IEEE International Conference on Cloud Computing (CLOUD'14) - Research Track, Alaska, USA, June 2014. (Acceptance rate: 20%) [35]

4. **A Truthful Approximation Mechanism for Autonomic Virtual Machine Provisioning and Allocation in Clouds**

   L. Mashayekhy, M. Nejad and D. Grosu

   Proc. of the ACM Cloud and Autonomic Computing Conference (CAC'13), Miami, USA, August 2013. [36]

5. **A Family of Greedy Mechanisms for Dynamic Virtual Machine Provisioning and Allocation in Clouds**

   M. Nejad, L. Mashayekhy and D. Grosu

   Proc. of the IEEE 6th International Conference on Cloud Computing (CLOUD'13) - Research Track, Santa Clara Marriott, USA, July 2013. (Acceptance rate: 18%) [40]

6. **Effects of Traffic Network Dynamics on Hierarchical Community-based Representations of Large Road Networks**

   M. Nejad, L. Mashayekhy and R. Chinnam

   Proc. of the 15th IEEE International Intelligent Transportation Systems Conference (ITSC'12), pp. 1900-1905, Anchorage, USA, September 2012. [39]

7. **State Space Reduction in Modeling Traffic Network Dynamics for Dynamic Routing under ITS**

   M. Nejad, L. Mashayekhy, A. Taghavi and R. Chinnam

Proc. of the 14th IEEE International Intelligent Transportation Systems Conference (ITSC'11), pp. 277-282, Washington DC, USA, October 2011. [41]

8. **Designing customer-oriented catalogs in e-CRM using an effective self-adaptive genetic algorithm**

   I. Mahdavi, M. Nejad, F. Adbesh

   Expert Systems with Applications, Volume 38, No. 1, January 2011. [27]

REFERENCES

[1] Amazon EC2 Instance Types.

[2] Grid workloads archive.

[3] IBM ILOG CPLEX V12.1 user's manual.

[4] Parallel workloads archive.

[5] WindowsAzure.

[6] G. Aggarwal and J. Hartline. Knapsack auctions. In *Proc. of the 17th annual ACM-SIAM symposium on Discrete algorithm*, pages 1083–1092, 2006.

[7] A. Archer and É. Tardos. Truthful mechanisms for one-parameter agents. In *Proc. of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 482–491, 2001.

[8] D. Ardagna, B. Panicucci, and M. Passacantando. Generalized Nash equilibria for the service provisioning problem in cloud systems. *IEEE Transactions on Services Computing*, 6(4):429–442, 2013.

[9] B. Awerbuch, Y. Azar, and A. Meyerson. Reducing truth-telling online mechanisms to online optimization. In *Proc. of the 35th annual ACM symposium on Theory of computing*, pages 503–510, 2003.

[10] Y. Bartal, R. Gonen, and N. Nisan. Incentive compatible multi unit combinatorial auctions. In *Proc. of the 9th Conf. on Theoretical Aspects of Rationality and Knowledge*, pages 72–87, 2003.

[11] J. Bi, Z. Zhu, R. Tian, and Q. Wang. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In *Proc. of the 3rd IEEE Intl. Conf. on Cloud Computing*, pages 370–377, 2010.

[12] R. N. Calheiros, R. Ranjan, and R. Buyya. Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In *Proc. of the 40th Intl. Conf. on Parallel Processing*, pages 295–304, 2011.

[13] S. Chaisiri, B.-S. Lee, and D. Niyato. Optimization of resource provisioning cost in cloud computing. *IEEE Transactions on Services Computing*, 5(2):164–177, 2012.

[14] S. De Vries and R. V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on computing*, 15(3):284–309, 2003.

[15] 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*, pages 115–122, 2013.

[16] W. Ellens, M. Zivkovic, J. Akkerboom, R. Litjens, and H. van den Berg. Performance of cloud computing centers with multiple priority classes. In *Proc. of the 5th IEEE Intl. Conf. on Cloud Computing*, pages 245–252, 2012.

[17] W. Fang, Z. Lu, J. Wu, and Z. Cao. Rpps: A novel resource prediction and provisioning scheme in cloud data center. In *Proc. of the 9th IEEE Intl. Conf. on Services Computing*, pages 609–616, 2012.

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

[19] N. Jain, I. Menache, J. Naor, and J. Yaniv. A truthful mechanism for value-based scheduling in cloud computing. *Theory of Computing Systems*, pages 1–19, 2013.

[20] 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*, pages 3–12, 2009.

[21] G. S. Kasbekar and S. Sarkar. Spectrum auction framework for access allocation in cognitive radio networks. *IEEE/ACM Transactions on Networking*, 18(6):1841–1854, 2010.

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

[23] 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*, pages 614–621, 2011.

[24] 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 Computing*, pages 83–90, 2012.

[25] D. Lehmann, L. Oćallaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):577–602, 2002.

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

[27] I. Mahdavi, M. Movahednejad, and F. Adbesh. Designing customer-oriented catalogs in e-crm using an effective self-adaptive genetic algorithm. *Expert Systems with Applications*, 38(1):631–639, 2011.

[28] R. Mansini and M. Speranza. Coral: An exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing*, 24(3):399–415, 2012.

[29] L. Mashayekhy and D. Grosu. A merge-and-split mechanism for dynamic virtual organization formation in grids. In *Proc. 30th IEEE Intl. Conf. on Performance Computing and Communications Conference*, pages 1–8, 2011.

[30] L. Mashayekhy and D. Grosu. A coalitional game-based mechanism for forming cloud federations. In *Proc. of the 5th IEEE Intl. Conf. on Utility and Cloud Computing*, pages 223–227, 2012.

[31] L. Mashayekhy and D. Grosu. A distributed merge-and-split mechanism for dynamic virtual organization formation in grids. In *Proc. 11th IEEE Intl. Conf. on Network Computing and Applications*, pages 36–43, 2012.

[32] L. Mashayekhy and D. Grosu. A reputation-based mechanism for dynamic virtual organization formation in grids. In *Proc. 41st IEEE Intl. Conf. on Parallel Processing*, pages 108–117, 2012.

[33] L. Mashayekhy and D. Grosu. A merge-and-split mechanism for dynamic virtual organization formation in grids. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):540–549, 2014.

[34] L. Mashayekhy, M. Nejad, D. Grosu, D. Lu, and W. Shi. Energy-aware scheduling of mapreduce jobs. In *Proc. of the 3rd IEEE International Congress on Big Data*, 2014.

[35] L. Mashayekhy, M. Nejad, D. Grosu, and A. Vasilakos. Incentive-compatible online mechanisms for resource provisioning and allocation in clouds. In *Proc. of the 7th IEEE International Conference on Cloud Computing*, 2014.

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

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

[38] M. Nejad, L. Mashayekhy, and D. Grosu. Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1–1, 2014.

[39] M. M. Nejad, L. Mashayekhy, and R. B. Chinnam. Effects of traffic network dynamics on hierarchical community-based representations of large road networks. In *Proc. of*

the 15th International IEEE Conference on Intelligent Transportation Systems, pages 1900–1905, 2012.

[40] 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*, pages 188–195, 2013.

[41] M. M. Nejad, L. Mashayekhy, A. Taghavi, and R. B. Chinnam. State space reduction in modeling traffic network dynamics for dynamic routing under its. In *Proc. of the 14th International IEEE Conference on Intelligent Transportation Systems*, pages 277–282, 2011.

[42] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. of the 31st annual ACM symposium on Theory of computing*, pages 129–140, 1999.

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

[44] M. H. Rothkopf, A. Pekeč, and R. M. Harstad. Computationally manageable combinational auctions. *Management science*, 44(8):1131–1147, 1998.

[45] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial intelligence*, 135(1):1–54, 2002.

[46] L. Shi, B. Butler, D. Botvich, and B. Jennings. Provisioning of requests for virtual machine sets with placement constraints in iaas clouds. In *Proc. IFIP/IEEE Intl. Symposium on Integrated Network Management*, pages 499–505, 2013.

[47] D. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1):461–474, 1993.

[48] 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*, 7(3):31, 2012.

[49] 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*, 54(2):252–269, 2010.

[50] F. Wu and N. Vaidya. A strategy-proof radio spectrum auction mechanism in noncooperative wireless networks. *IEEE Transactions on Mobile Computing*, 12(5):885–894, 2013.

[51] 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.*, pages 107–114, 2011.

[52] S. Zaman and D. Grosu. Combinatorial auction-based allocation of virtual machine instances in clouds. *J. of Parallel and Distributed Computing*, 73(4):495–508, 2013.

[53] 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*, pages 2–13, 2008.

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

[55] E. Zurel and N. Nisan. An efficient approximate allocation algorithm for combinatorial auctions. In *Proc. of the 3rd ACM Conf. on Electronic Commerce*, pages 125–136, 2001.

ABSTRACT

Truthful Mechanisms for Resource Allocation and Pricing in Clouds

by

Mahyar Movahed Nejad

May 2014

Advisor: Dr. Daniel Grosu

Major: Computer Science

Degree: Master of Science

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.

AUTOBIOGRAPHICAL STATEMENT

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