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

Lena Mashayekhy  Mahyar Movahed Nejad  Daniel Grosu
Dept. Computer Science  Dept. Computer Science  Dept. Computer Science
Wayne State University  Wayne State University  Wayne State University
Detroit, MI 48202, USA  Detroit, MI 48202, USA  Detroit, MI 48202, USA
mlena@wayne.edu  mahyar@wayne.edu  dgrosu@wayne.edu

ABSTRACT

One of the major challenges faced by the cloud providers is to allocate and provision the resources such that their profit is maximized and the resources are utilized efficiently. We address this challenge by designing an autonomic VM (Virtual Machine) provisioning and allocation mechanism that adapts to the changing user demands. We show that the proposed mechanism is a PTAS (Polynomial-Time Approximation Scheme) and that it is 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 properties of the mechanism.

Categories and Subject Descriptors

K.6 [Management of Computing and Information Systems]: Installation Management—pricing and resource allocation

General Terms

Cloud computing, Autonomic Computing

Keywords

cloud computing, autonomic resource allocation, PTAS, truthful mechanism.

1. INTRODUCTION

The number of enterprises and individuals that are outsourcing their workloads to cloud providers has been increasing rapidly. Clouds 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CAC’13, August 5–9, 2013, Miami, FL, USA.
Copyright 2013 ACM 978-1-4503-2172-3/13/08 ...$15.00.

In this paper, we focus on IaaS where cloud providers offer different types of resources in the form of VM instances. The types of VM instances that a cloud provider offers are known to the users. For instance, Microsoft Azure [3] 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 provision resources in the form of VM instances and then allocate them to users. In static provisioning the cloud provider pre-provisions a set of VM instances without considering the actual demand from the users, while in dynamic provisioning the cloud provider provisions the resources taking into account the actual user demand. Due to a variable load demand, dynamically provisioning resources leads to a more efficient resource utilization and ultimately to higher revenues for the cloud provider.

The ever-growing complexity in IaaS makes human administration and management inefficient and, in most of the cases, unfeasible. Therefore, avoiding direct management actions in resource allocation, VM provisioning, and monitoring, requires self-management and self-optimizing mechanisms. The aim of this paper is to design such mechanisms that facilitate autonomic provisioning of cloud resources based on the user demand and the availability of resources. The proposed mechanisms can be incorporated in system tools for self-managing the cloud infrastructure [14].

The current allocation mechanisms used by the cloud providers when offering IaaS are fixed-price or auction-based. In the fixed-price form, the price of each type of resources are fixed while in the auction-based users submit bids for their requested resources. Recently, the focus of resource provisioning and allocation in clouds is moving towards auction-based market models. Users can obtain their requested resources at lower prices than in the case of fixed-price, and the cloud providers can increase their profit by allowing users to bid on unutilized capacity. An instance of such a business model is the spot market introduced by Amazon [1]. In an auction-based resource allocation model there is a set of users and a set of items where each user bids for a subset of items (bundle). In this study, VMs are considered as items. 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 is, they are interested in a single bundle of VM instances and bid only for that bundle. A single minded user obtains the specified value if she is allocated the whole bundle of VM instances (or any superset of it) and zero value, if she is allocated any other bundle. The users are also selfish.
in the sense that they want to maximize their own 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).

In this paper, we aim at designing a truthful polynomial time approximation scheme (PTAS) mechanism that solves the VM instance provisioning and allocation problem. The goal is to find an allocation of resources to the users maximizing the social welfare, where the social welfare is the sum of users valuations. Our proposed mechanism for VM provisioning and allocation drives the system into an equilibrium in which the users do not have incentives to manipulate the system by untruthfully reporting their VM bundle requests and valuations.

1.1 Our Contributions

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 a truthful PTAS mechanism for solving the problem of VM instance provisioning and allocation in clouds. First, we design a truthful exact mechanism based on Vickrey-Clarke-Groves (VCG) mechanism [13] that uses a dynamic programming algorithm to optimally select the winning users. In the absence of feasible optimal algorithms for solving this problem, we then design an approximation algorithm. In general, approximation algorithms do not necessarily satisﬁe the properties required to achieve truthfulness. Our proposed mechanism, called PTAS-VMPAC (PTAS Virtual Machine Provisioning and Allocation in Clouds), is a truthful PTAS mechanism that gives incentives to users to reveal their true valuations for the requested bundle of VM instances. We analyze the properties of the PTAS-VMPAC mechanism and perform extensive simulation experiments. The results show that PTAS-VMPAC determines near optimal allocations while satisfying the truthfulness property.

1.2 Related Work

Developing efﬁcient resource provisioning policies is a major challenging problem in clouds. Wood et al. [15] proposed an approach for dynamic provisioning of virtual machines 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 [8] proposed a method for dynamic provisioning of services in clouds in order to optimize the distribution of services within a certain infrastructure. Ghodsi et al. [7] studied fair allocation of multiple resource types where users share resources. Their proposed approach is strategy-proof which means a user cannot increase her allocation by lying about her request. Ferrer et al. [6] proposed a toolkit for the cloud service and infrastructure providers. The toolkit aims to provide a foundation for a reliable cloud computing industry, by addressing the whole service life cycle. Incorporating business level objectives in resource management policies of clouds were studied in [4, 10, 11]. The focus was on maximizing the proﬁt or revenue without considering the users incentives for manipulating the allocation mechanisms by untruthful reporting. However, our main focus is maximizing social welfare in dynamic resource provisioning in order to achieve truthfulness and to lead the system to an equilibrium. We also propose an approach for price determination of the VMs.

Table 1: VM instance types offered by Amazon EC2.

<table>
<thead>
<tr>
<th></th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
<th>Extra Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Memory (GB)</td>
<td>1.7</td>
<td>3.75</td>
<td>7.5</td>
<td>15</td>
</tr>
<tr>
<td>Storage (GB)</td>
<td>160</td>
<td>410</td>
<td>850</td>
<td>1690</td>
</tr>
</tbody>
</table>

The closest work to ours is by Zaman and Grosu [17, 16] who proposed truthful approximation mechanisms for combinatorial auction-based allocation of VM instances in clouds in static and dynamic settings. However, these mechanisms are simple approximation mechanisms and not PTAS mechanisms. Briest et al. [5] proposed a truthful FPTAS (1+)-approximation mechanism for the knapsack problem. This mechanism is designed for single-minded multi-unit auctions. They also proposed a PTAS mechanism for a speciﬁc class of Generalized Assignment Problem (GAP) where all bins have the same capacity. We extend their technique to design a PTAS mechanism for solving the problem of VM instance provisioning and allocation in clouds. The reader is referred to [13] for a comprehensive introduction to mechanism design.

1.3 Organization

The rest of the paper is organized as follows. In Section 2, we describe the VM provision and allocation problem in clouds. In Section 3, we present the proposed mechanism and characterize its properties. In Section 4, we evaluate the mechanism by extensive simulation experiments. In Section 5, we summarize our results and present possible directions for future research.

2. VM PROVISIONING AND ALLOCATION PROBLEM

We consider a cloud provider offering R types of resources, \( R = \{1, \ldots, R\} \), to users in the form of VM instances. These resources include cores, memory, storage, etc. The cloud provider has restricted capacity, \( C_i \), on each resource \( r \in R \) available for allocation. The cloud provider offers these resources in the form of M types of VMs, \( \forall M = \{1, \ldots, M\} \), where each VM of type \( m \in \forall M \) provides a specific amount of each type of resource \( r \in R \). The amount of resources of type \( r \) that one VM instance of type \( m \) provides is denoted by \( w_{mr} \). As an example, in Table 1, we present the four types of VM instances offered by Amazon EC2 at the time of writing this paper. If we consider that CPU represents the type 1 resource, memory the type 2 resource, and storage the type 3 resource, we can characterize, for example, the Small instance (\( m = 1 \)) by: \( w_{11} = 1, w_{12} = 1.7 \) GB, and \( w_{13} = 160 \) GB.

We consider a set of N users requesting a set of VMs. User \( i, i = 1, \ldots, N \), requests a bundle \( S_i \leq k_{i1}, k_{i2}, \ldots, k_{iM} \geq 0 \) of M types of VM instances, where \( k_{im} \) is the number of requested VM instances of type \( m \in \forall M \). 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 instance, \( \langle 2, 1, 4, 3 \rangle, \$10 \)
represents a user requesting 2 small VM instances, 1 medium
VM instance, 4 large VM instances, and 3 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,
\[ V = \sum_{i \in U} v_i(S_i) \cdot x_i \]
where $x_i$, $i = 1, \ldots, N$, are indicator variables defined as follows:
\[ x_i = \begin{cases} 1 & \text{if bundle } S_i \text{ is allocated to user } i, \\ 0 & \text{otherwise} \end{cases} \]

Table 2 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 mecha-
nism design objective, that is, maximization of $V$, the sum
of the users’ valuations [13]. Since the valuation of a user
represents a user requesting 2 small VM instances, 1 medium
VM instance, 4 large VM instances, and 3 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,
\[ V = \sum_{i \in U} v_i(S_i) \cdot x_i \]
where $x_i$, $i = 1, \ldots, N$, are indicator variables defined as follows:
\[ x_i = \begin{cases} 1 & \text{if bundle } S_i \text{ is allocated to user } i, \\ 0 & \text{otherwise} \end{cases} \]

Table 2 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 mecha-
nism design objective, that is, maximization of $V$, the sum
of the users’ valuations [13]. Since the valuation of a user
represents a user requesting 2 small VM instances, 1 medium
VM instance, 4 large VM instances, and 3 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,
\[ V = \sum_{i \in U} v_i(S_i) \cdot x_i \]
where $x_i$, $i = 1, \ldots, N$, are indicator variables defined as follows:
\[ x_i = \begin{cases} 1 & \text{if bundle } S_i \text{ is allocated to user } i, \\ 0 & \text{otherwise} \end{cases} \]

Table 2 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 mecha-
nism design objective, that is, maximization of $V$, the sum
of the users’ valuations [13]. Since the valuation of a user
represents a user requesting 2 small VM instances, 1 medium
VM instance, 4 large VM instances, and 3 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,
\[ V = \sum_{i \in U} v_i(S_i) \cdot x_i \]
where $x_i$, $i = 1, \ldots, N$, are indicator variables defined as follows:
\[ x_i = \begin{cases} 1 & \text{if bundle } S_i \text{ is allocated to user } i, \\ 0 & \text{otherwise} \end{cases} \]

Table 2 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 mecha-
nism design objective, that is, maximization of $V$, the sum
of the users’ valuations [13]. Since the valuation of a user
represents a user requesting 2 small VM instances, 1 medium
VM instance, 4 large VM instances, and 3 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,
\[ V = \sum_{i \in U} v_i(S_i) \cdot x_i \]
where $x_i$, $i = 1, \ldots, N$, are indicator variables defined as follows:
\[ x_i = \begin{cases} 1 & \text{if bundle } S_i \text{ is allocated to user } i, \\ 0 & \text{otherwise} \end{cases} \]

The solution to this problem is a vector $x = (x_1, x_2, \ldots, x_n)$
maximizing the social welfare. Constraints (4) ensure that
the allocation of each resource type does not exceed the
available capacity of that resource. Constraints (5) repres-
ent 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 [9]. The objective is to select
a subset of items for the multidimensional knapsack maxi-
mizing the total value.

3. TRUTHFUL MECHANISMS FOR VM
PROVISIONING AND ALLOCATION

In this section, we first present the basic concepts of mech-
anism design and propose a VCG-based exact mechanism
that solves VMPAC. We then propose a truthful PTAS me-
chanism, called PTAS-VMPAC, that solves the VMPAC
problem.

3.1 Preliminaries

A mechanism $\mathcal{M} = (A, P)$ consists of an allocation func-
tion $A = (A_1, \ldots, A_N)$ and a payment rule $P = (P_1, \ldots, P_N)$.The allocation function determines which users receive their
requested bundles, and the payment rule determines the
amount that each user must pay.

Table 2: Notation

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

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, a 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:
\[ v_i(S_i) = \begin{cases} b_i & \text{if } S_i \subseteq \hat{S}_i \\ 0 & \text{otherwise} \end{cases} \]

The goal is to design a truthful mechanism that maximizes
the social welfare $V = \sum_{i \in U} v_i(S_i)$, where $\hat{S}_i$ is the bundle
allocated to user $i$.

We denote by $\theta = (\theta_1, \ldots, \theta_N)$ the vector of types of all
users. $\theta_i$ is the vector of all types except user $i$’s type
(i.e., $\theta_i = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_N)$). User $i$ has a
utility function $u_i(\theta) = v_i(A_i(\theta)) - P_i(\theta)$, where $P_i(\theta)$ is
the payment for user $i$ that the mechanism calculates based on
the payment rule $P$. Each user’s type is private knowledge.
The users may declare different types from their true types. We
denote by $\hat{\theta}_i = (S_i, b_i)$ 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 mecha-
nism 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 an inefficient allocation of resources and ultimately
will reduce the revenue obtained by the cloud provider. We
want to prevent such manipulations by designing a truthful
mechanism for solving VMPAC. A mechanism is truthful
if all users have incentives to reveal their true types.

Definition 1. A mechanism $\mathcal{M}$ is truthful (or incentive
compatible) if for every user $i$, for every type declaration
of the other users $\theta_{-i}$, a true type declaration $\theta_i$ and any
other declaration $\hat{\theta}_i$ of user $i$, we have that $\theta_i, \hat{\theta}_i \geq
u_i(\theta_i, \theta_{-i})$.

In other words, a mechanism is truthful if truthful report-
ing 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 $A$ must be monotone
and the payment rule must be based on the critical value [12].
To define monotonicity, we need to introduce a preference
relation $\succeq$ on the set of types as follows: $\theta_i' \succeq \theta_i$ if $S_i' \subseteq \hat{S}_i$ and $b_i' \geq b_i$ for user $i$. That means the type $\theta_i'$ is
more preferred than $\theta_i$ if user $i$ requests a subset of her current
bundle and/or submits a higher bid.
Definition 2. An allocation function \( A \) is monotone if it allocates the resources to user \( i \) with \( \theta_i \) as her declared type, then it also allocates the resources to user \( i \) with \( \theta_i' \), where \( \theta_i' \succeq \theta_i \).

Any winning user who receives her requested bundle by declaring a type \( \theta_i \) will still be a winner if she requests a smaller bundle and submits a higher bid.

Definition 3. Let \( A \) be a monotone allocation function, then for every \( \theta_i \), there exist a unique value \( v_i' \), called critical value, such that \( \forall \theta_i \geq (S_i, v_i') \), \( \theta_i \) is a winning declaration, and \( \forall \theta_i < (S_i, v_i') \) 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 \( A \) and the payments using the payment rule \( P \). The payment rule \( P \) is based on the critical value and is defined as follows:

\[
P_i(\hat{\theta}) = \begin{cases} 
  v_i' & \text{if } i \text{ wins} \\
  0 & \text{otherwise}
\end{cases}
\]

(7) where \( v_i' \) is the critical value of user \( i \).

Definition 4. A monotone allocation function \( A \) is bitonic if for any user \( i \):

- if \( A \) allocates the resources to the user \( i \) with \( \theta_i \) as her declared type, then \( v_i(A(\theta_i', \theta_{-i})) \geq v_i(A(\theta_i, \theta_{-i})) \), where \( \theta_i' \succeq \theta_i \).

- if \( A \) does not allocate the resources to the user \( i \) with \( \theta_i \) as her declared type, then \( v_i(A(\theta_i', \theta_{-i})) \geq v_i(A(\theta_i, \theta_{-i})) \), where \( \theta_i' \succeq \theta_i \).

\( A \) is bitonic with respect to \( v_i \). This requires that the welfare does not increase with \( v_i \) when user \( i \) loses \( (v_i < v_i') \), and it does increase with \( v_i \) when user \( i \) wins \( (v_i > v_i') \).

3.2 Truthful Exact Mechanism

We introduce a VCG-based truthful exact mechanism. VCG requires an optimal allocation algorithm implementing the allocation function \( A \) [10]. A VCG mechanism is defined as follows:

Definition 5. A mechanism \( \mathcal{M} = (A, P) \) is a Vickrey-Clarke-Groves (VCG) mechanism if \( A \) maximizes the social welfare, and

\[
P_i(\hat{\theta}) = \sum_{j \in A(\theta_{-i}), j \neq i} \hat{v}_j - \sum_{j \in A(\theta_{-i}), j \neq i} \hat{v}_j,
\]

(8) where \( \sum_{j \in A(\theta_{-i}), j \neq i} \hat{v}_j \) is the optimal social welfare that would have been obtained had user \( i \) not participated, and \( \sum_{j \in A(\theta_{-i}), j \neq i} \hat{v}_j \) is the sum of all users valuations except user \( i \)’s.

In order to design a VCG-based mechanism for VMPAC we need to design an algorithm that provides the optimal solution to VMPAC. The algorithm, called DP-VMPAC, is based on a dynamic programming approach, and it is given in Algorithm 1. The DP-VMPAC algorithm has two input parameters, the vector of users declared types \( \hat{\theta} \) and the vector of resource capacities \( C = (C_1, \ldots, C_R) \). The algorithm has two output parameters: \( V^* \), the optimal social welfare, and \( x^* \), the optimal allocation of VM instances to the users.

### Algorithm 1 DP-VMPAC: Exact Allocation Algorithm

1: Input: \( \hat{\theta} = (\theta_1, \ldots, \theta_P) \); vector of types (bundle, bid)
2: Input: \( C = (C_1, \ldots, C_R) \); vector of resource capacities
3: for all \( i \in U \) do
4: for all \( r \in R \) do
5: \( a_{ir} = \sum_{m \in \Omega_{VM}} k_{ir} \text{cost}_{ir} \)
6: \( A_i = (a_{i1}, \ldots, a_{iK}) \)
7: \( C = C \)
8: if \( a_{ir} \leq C_r \), \( \forall r \in R \) then
9: \( V(1, C) = v_1 \)
10: \( C = C - A_1 \)
11: else
12: \( V(1, C) = 0 \)
13: for all \( j = 2, \ldots, N \) do
14: \( V(j, C) = \max\{V(j - 1, C - A_{j}), V(j - 1, C) + v_j\} \)
15: \( V^* = V(N, C) \)
16: Find \( x^* \) by looking backward at \( V(j, C) \)
17: Output: \( V^*, x^* \)

The recurrence considers two cases, not allocating the bundle to \( j \) and allocating it to \( j \). If allocating the requested bundle of the \( j \)th user increases the value \( V(j - 1, C) \), the algorithm allocates the bundle to the \( j \)th user. The maximum between \( V(j - 1, C) \) and \( V(j - 1, C - A_{j}) + v_j \) gives the optimal value of \( V(j, C) \). Once the final value \( V(N, C) \) is determined, the algorithm finds \( x^* \), the optimal allocation of VM instances, by looking backward at \( V(j, C) \).

The DP-VMPAC algorithm finds the optimal solution to the VMPAC problem. Showing that the dynamic programming approach in this case provides an optimal solution is trivial and we will not provide a proof for it here. VMPAC is equivalent to the multidimensional knapsack problem (MKP) which is strongly NP-hard [9]. Thus, the VMPAC is also strongly NP-hard. DP-VMPAC solves VMPAC optimally in time \( O(\max N C^2) \), where \( \max C = \max_{C_r} 0 \{C_r\} \). This is due to the fact that the dynamic programming builds a \((R + 1)\)-dimensional table, where the first dimension corresponds to the number of users and the other \( R \) dimensions correspond to the \( R \) types of resources.

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

Definition 6. The VCG-VMPAC mechanism consists of the allocation algorithm DP-VMPAC and the payment function VCG-PAY defined by the VCG payment rule given in equation (8).

The VCG-VMPAC mechanism is given in Algorithm 2. The mechanism is run periodically by the cloud provider. It collects the requests from the users, expressed as types,
and determines the allocation by calling the DP-VMPAC allocation algorithm. Once the allocation is determined, the mechanism provisions the required number and types of VM instances and determines the payments by calling the VCG-PAY function. The users are then charged the amount determined by the mechanism.

The VCG-PAY function is given in Algorithm 3. VCG-PAY has four input parameters, the vector of users declared types \( \mathbf{\theta} \), the vector of resource capacities \( \mathbf{C} \), the optimal welfare \( V^* \), and the optimal allocation given by \( \mathbf{x}^* \). It has one output parameter: \( \mathbf{P} \), the payment vector for the users. VCG-PAY calls DP-VMPAC to find the allocation and welfare obtained without user \( i \)'s participation (line 6). Based on the optimal allocation to the users with and without user \( i \)'s participation, VCG-PAY finds the payment for user \( i \), where \( \text{sum}_1 \) is the sum of all values with user \( i \)'s participation in the mechanism, and \( \text{sum}_2 \) is the sum of all except user \( i \)'s value in the optimal case (lines 7-12).

The VCG-VMPAC mechanism is truthful and determines the optimal allocation, but its execution time becomes prohibitive for large instances of VMPAC. More than this, the problem is strongly NP-hard and there is no Fully Polynomial Time Approximation Scheme (FPTAS) for solving it, unless \( P = NP \) [9]. Thus, the best we can do is to design a PTAS mechanism for solving it. In the next section, we will design such a PTAS mechanism for VMPAC.

### 3.3 Truthful PTAS Mechanism

We now introduce our proposed truthful PTAS mechanism, VMPAC-PTAS. The design of the VMPAC-PTAS allocation algorithm is based on an idea proposed by Brust et al. [5] for the design of a monotone allocation algorithm for the Generalized Assignment Problem (GAP). They designed a monotone allocation algorithm for GAP where all bins have the same capacity. The idea is to determine partial assignments of \( k \) items to bins, then round the sizes of the unallocated items and use an optimal allocation algorithm to allocate those items to the remaining capacity (i.e., the capacity left after partial assignments).

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

**Definition 7.** The PTAS-VMPAC mechanism consists of the allocation algorithm PTAS-VMPAC and the payment function C-PAY.

Our monotone PTAS allocation algorithm, called PTAS-VMPAC, is given Algorithm 4. PTAS-VMPAC has three input parameters: the vector of users declared types \( \mathbf{\theta} \), the vector of resource capacities \( \mathbf{C} \), and an integer \( q \), where \( q \leq N \). The parameter \( q \) controls how close the solution determined by PTAS-VMPAC is to the optimal solution. The PTAS-VMPAC algorithm has two output parameters: \( V^* \), the total social welfare and \( \mathbf{\mathbf{x}}^* \), the allocation of VM instances to the users.

The PTAS-VMPAC algorithm iterates over all subsets \( \mathbf{\hat{U}} \) of at most \( q \) users (lines 5-38). For each such subset the algorithm finds a feasible partial allocation \( \mathbf{\hat{x}} \) of at most \( q \) users (lines 17-19) and rounds the amount of requested resources by the unallo-
The remaining unallocated capacities, the corresponding VM instance allocation \( \mathbf{x} \) obtained over all iterations (lines 34-38).

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

\[
V(j, d) = \max\{V(j-1, d), V(j-1, d - \mathbf{A}_j) + v_j\} \quad (10)
\]

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

**Theorem 1.** PTAS-VMPAC is monotone.

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

First, we show that the solution is within \((1 - \epsilon)\) of the optimal solution. Let \( \mathbf{x}^* \) be the optimal allocation of the requested bundles, and \( V^* \) be the corresponding optimal value. Assume that PTAS-VMPAC determines an allocation \( \mathbf{x} \) and a value \( V \). Let \( \mathbf{x} \) be the optimal allocation when we consider only \( q \) users with the highest declared values in the first step. The second step of allocation is allocating the remaining resources given by \( d \) to the users who were not selected in the first step. The rounding procedure for the remaining users, in the second step, increases the size of the requested bundles of those users for each resource type. This may lead to an infeasible allocation of the bundles based on the new rounded sizes. Based on the rounding, the total increase in the size of the requested bundles for each resource is less than \( d_r/N \). In order to make the allocation feasible, we can remove a requested bundle such that it satisfies the capacity constraints for each resource type while decreasing the least amount of value from the objective function.

We find those allocated bundles in the second step where for all resource types their size is larger than \( d_r/N \). Among those, we choose the bundle \( S_r \) with the smallest size. Since in the first step, we chose the \( q \) bundles with the highest values, the bundle \( S_r \) can be the \( q+1 \) most valuable bundle. Therefore, user \( r \) valuation for this bundle is \( v_r(S_r) \leq 1/(q+1)V^* \). Removing bundle \( S_r \) makes the obtained objective function between \((1 - 1/(q+1))V^* \) and \( V^* \). Therefore, we have \((1 - \epsilon)V^* < \mathbf{V} \leq V^* \) where \( \epsilon = 1/(q+1) \).

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

The PTAS-VMPAC mechanism is given in Algorithm 5. The mechanism is run periodically by the cloud provider. It collects the requests from the user expressed as types and determines the allocation by calling the PTAS-VMPAC allocation algorithm. Once the allocation is determined the mechanism provisions and allocates VM instances according to $\mathbf{x}^*$. The payments are based on the critical vector of users declared types ($\hat{\mathbf{x}}$), the vector of resource capacities $\mathbf{C}$, the optimal allocation given by $\mathbf{x}^*$, and the integer $q$. It has one output parameter: $\mathcal{P}$, the payment vector for the users. The payments are based on the critical types of the winning users. The payment of winning user $i$ is $v_i^*$, where $v_i^*$ is the critical value of user $i$, if $i$ wins and zero if $i$ loses. Finding the critical value is done by a binary search over values less than the declared value.

We now show that the proposed mechanism is truthful.

**Theorem 3.** The PTAS-VMPAC mechanism is truthful.

**Proof.** The allocation algorithm PTAS-VMPAC is monotone (Theorem 1) and the payment is the critical value payment (implemented by C-PAY), therefore the PTAS-VMPAC mechanism is truthful. 

In the next section we evaluate the proposed mechanism by simulation experiments.

### 4. EXPERIMENTAL RESULTS

We perform two sets of simulation experiments which allows us to investigate the properties of PTAS-VMPAC. In the first set of experiments, we compare the performance of PTAS-VMPAC with that of VCG-VMPAC for a case with eight users. We also investigate the effects of untruthful declaration of types by a user. Since VMPAC is strongly NP-hard, obtaining optimal solutions is feasible only for small size instances of VMPAC. In the second set of experiments, we investigate the performance of PTAS-VMPAC for larger VMPAC problems. We also conduct a sensitivity analysis study on several parameters such as number of users and $\epsilon$, where $\epsilon = \frac{1}{qn}$. VCG-VMPAC and PTAS-VMPAC mechanisms are implemented in C++ and the experiments are conducted on Intel 2.93GHz Quad Core 16GB RAM which are part of the Wayne State Grid System.

#### 4.1 Experimental Setup

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 the same types of VM instances available to users as those offered by Amazon EC2. Each of these VM instances has specific resource demands with respect to two available resource types: cores and storage. 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 Compute Cloud (EC2) (See Table 1).

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, 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 $\sum_{m=1}^{M} 2^{m-1} k_{im}$. The parameters and their generated values for the experiments are listed in Table 3.

### 4.2 Analysis of Results

We first compare the performance of our mechanism, PTAS-VMPAC, with that obtained by VCG-VMPAC for eight users. In order for VCG-VMPAC to be able to solve the VMPAC problem, we consider smaller capacities of the two resources as follows: 100 cores, and 1800 (10GB) of storage.

Fig. 1 shows the social welfare of eight users based on the parameters and their generated values for the experiments. For example, for $\epsilon = 0.5$, the social welfare is 223 and the optimal social welfare is 230 satisfying: $1 - 0.5 \times 230 = 115 < 223 < 230$. For smaller $\epsilon$, PTAS-VMPAC obtained the optimal social welfare. Fig. 2 shows the execution time of PTAS-VMPAC for the same selected $\epsilon$, and the execution time of VCG-VMPAC. The results show that PTAS-VMPAC is able to find a near optimal social welfare in much shorter time. This is due to the fact the PTAS-VMPAC is a polynomial time approximation scheme.

In addition, we investigate the effects of untruthful declarations by a user. In this set of experiments, our goal is to show that our proposed mechanism, PTAS-VMPAC, is robust against manipulation by a user. The true types of the eight users are shown in Table 4. The PTAS-VMPAC ($\epsilon = 0.33$) allocates resources to user 1, 2, 3, 7 and 8 in the case that all users declare their true types. The payments of the winning users based on C-PAY are 3, 3, 18, 0, and 10, respectively.

Table 4: Users' true types

<table>
<thead>
<tr>
<th>User</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_{i1}$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>$k_{i2}$</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$k_{i3}$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$v_i$</td>
<td>30</td>
<td>18</td>
<td>95</td>
<td>10</td>
<td>5</td>
<td>15</td>
<td>7</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 5: Different scenarios for user 8's type declaration

<table>
<thead>
<tr>
<th>Case</th>
<th>$S_8$</th>
<th>$v_8$</th>
<th>Scenario</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$&lt; 3, 2, 1, 1 &gt;$</td>
<td>$\hat{v}_8 = v_8, S_8 = S_8$</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>II</td>
<td>$&lt; 3, 2, 1, 1 &gt;$</td>
<td>$\hat{v}_8 &lt; v_8, S_8 = S_8$</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>III</td>
<td>$&lt; 3, 2, 1, 1 &gt;$</td>
<td>$\hat{v}_8 &lt; v_8, S_8 &gt; S_8$</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>IV</td>
<td>$&lt; 3, 2, 1, 1 &gt;$</td>
<td>$\hat{v}_8 &lt; v_8, S_8 &gt; S_8$</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>$&lt; 3, 2, 1, 3 &gt;$</td>
<td>$\hat{v}_8 = v_8, S_8 &gt; S_8$</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>VI</td>
<td>$&lt; 3, 2, 1, 5 &gt;$</td>
<td>$\hat{v}_8 = v_8, S_8 &gt; S_8$</td>
<td>L</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: PTAS-VMPAC vs. VCG-VMPAC: Social welfare.

Figure 2: PTAS-VMPAC vs. VCG-VMPAC: Execution time.

Figure 3: PTAS-VMPAC: Effect of untruthful declarations.

We assume that user 8 lies about her type $\hat{\theta}_8$. The consequence of such a declaration depends on her reported value $v_8$ and the bundle $S_8$. We consider different scenarios as shown in Table 5, where user 8 does not reveal her true type. Fig. 3 shows the payment and utility of the user for all the cases. Case I is when the user declares her true type. In case II, when user 8 reports a value greater than her true type, she is still a winner and the mechanism determines the same payment for her as in case I. In case III, user 8 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 8 reports a value below her true value. In this case, she will not get her requested bundle. In case V, she declares a larger bundle and still obtains the bundle due to available capacities. In case VI, she declares a larger bundle...
but becomes a loser since the cloud provider does not have enough resources to fulfill her request. In all cases, the user can not increase her utility by declaring a type other than her true type.

We now analyze the performance of PTAS-VMPAC for different number of users. The results are presented for an average of 10 cases for each number of users. Fig. 4 shows the social welfare for 24 to 64 users where $\epsilon$ is 0.5 and 0.33. This figure shows that for each number of users, the social welfare increases as $\epsilon$ decreases. For example, for 24 users by decreasing $\epsilon$ from 0.5 to 0.33, PTAS-VMPAC obtains a higher social welfare. In addition, the social welfare increases by increasing the number of users since the cloud provider is able to allocate more VMs, and obtain higher social welfare. Fig. 5 shows the execution time of PTAS-VMPAC for different number of users, where $\epsilon$ is 0.5 and 0.33. This figure shows that by increasing the number of users, the execution time of PTAS-VMPAC increases. However, this increase is polynomial in the number of users. In addition, by decreasing $\epsilon$, the execution time of PTAS-VMPAC increases. This is the case for any PTAS algorithm.

Fig. 6 and Fig. 7 show the utilization of cores and storage, respectively. Here the utilization is defined as the percentage of the available resources that are allocated by the mechanism. The results show that the utilization of the cloud resources using PTAS-VMPAC is increasing by decreasing $\epsilon$. This is due to the fact that the allocations achieved by PTAS-VMPAC by decreasing $\epsilon$ gets closer to the optimal allocation which utilizes more effectively the resources.

5. CONCLUSION

We proposed a truthful PTAS mechanism for autonomic resource allocation in clouds that provides incentives to the users to reveal their true valuations for the requested bundles of VM instances. We also designed a truthful VCG based mechanism using a dynamic programming approach. We investigated the properties of our proposed PTAS mechanism by performing extensive simulation experiments. The results showed that the proposed mechanism determines near optimal allocations while giving the users incentives to report their true valuations for the bundles of VM instances. We plan to perform more experiments and implement the mechanism as part of an integrated solution for autonomic management of resources in an experimental cloud computing system.

Acknowledgment

This research was supported in part by NSF grants DGE-0654014 and CNS-1116787.
6. REFERENCES


