

An Online Mechanism for Dynamic VM Provisioning and Allocation in Clouds

Sharrukh Zaman
Department of Computer Science
Wayne State University
Detroit, Michigan, USA
Email: sharrukh@wayne.edu

Daniel Grosu
Department of Computer Science
Wayne State University
Detroit, Michigan, USA
Email: dgrosu@cs.wayne.edu

Abstract—Current cloud computing providers allocate their virtual machine (VM) instances via fixed price-based or auction-like mechanisms. However, these mechanisms have one limitation, they are all offline mechanisms, therefore they need to collect information and be invoked periodically. In this paper, we address this limitation by designing an online mechanism for dynamic provisioning and allocation of VM instances in clouds. Our proposed mechanism, MOVMPA, is invoked as soon as a user places a request or some VM instances already allocated become available again. When invoked, the mechanism selects users who would be allocated VM instances for the period they requested for, and ensures that those users will continue using those VMs for the entire period requested. We prove that the mechanism is incentive compatible and also investigate its performance through extensive simulation experiments.

Index Terms—cloud computing; online mechanisms; combinatorial auctions; allocation; scheduling;

I. INTRODUCTION

Virtualization technologies have created convenient ways for the cloud providers to allocate their computing resources. They define different configurations of virtual machines (VMs) and ‘sell’ the resources in units of VM instances. Currently, the cloud providers use fixed price-based mechanisms to allocate and sell VM instances (e.g., Windows Azure [1], Amazon EC2 [2]) or auction-based mechanisms to sell resources that are not utilized after the fixed-price based selling [3].

There are certain features of the problem of VM allocation in clouds that makes online mechanisms suitable for solving it. First of all, it is quite natural for users to submit bids for bundles of VMs rather than for individual VMs. For example, if a user requires one ‘small’ and one ‘large’ VM instance for a particular application, she would prefer to get both of them together, otherwise she would prefer nothing to getting only one of the VMs. Therefore, the allocation mechanism must allow users to express their preferences for a bundle of VMs and not only for individual VMs. On the other hand, the users submit their requests (or arrive at the system) continuously and would prefer to have their allocation decision as soon as possible, at the best possible price. Once they obtain the requested VM bundles, they would like to complete their entire task on the acquired resources.

Another consideration is that the cloud provider would prefer a mechanism that supports dynamic provisioning so that

they can decide on the number of instances of different types of VMs based on the market demand. The cloud provider is also interested in maximizing its revenue or profit. Considering all the desired properties mentioned above, in this paper, we design an online mechanism that provisions computing resources into VM instances, allocates them to users, and determines the payment for each user. Online mechanisms solve allocation problems without having all the information available. An online mechanism calculates allocation and payment as the participants arrive at the system and place their requests. This is particularly useful to guarantee allocation efficiency in systems where users arrive continuously and items being allocated are expiring items [4]. For the problem of VM instance allocation in clouds, the resources can be considered as expiring since the allocation is dependent on time: a resource not allocated at time t loses its utilization for that particular moment.

We design an online mechanism for provisioning and allocating VM instances in clouds. We design the bidding protocol so that a user requests a bundle of VMs expressing that she is only interested in the whole bundle and not a subset of it. Upon receiving such bids, the mechanism calculates the allocation and payment online. Also, the allocation cannot be preempted: a user receiving an allocation at time t continues to hold the resources for the time period she requested the resources for, while submitting her bid. We also provide theoretical results proving the properties of the proposed mechanism.

Related work. The economic aspects of cloud computing have been investigated by several researchers. Wang *et al.* [5] studied different economic and system implications of pricing resources in clouds. They performed experiments on Amazon EC2 and on their own testbed concluding that the pricing scheme used by Amazon is unfair to the users. Lin *et al.* [6] proposed the use of a simple $(k + 1)^{th}$ price auction for allocating cloud resources. They showed by statistical analysis that when there is a large number of resources and users, the auction can obtain an efficient allocation and a reasonable revenue for the cloud provider. Risch *et al.* [7] designed a cloud services testbed to test cloud mechanisms. Altmann *et al.* [8] designed an exchange market for grid resources, which was tested on the platform designed by Risch *et al.* [7].

Such a marketplace for clouds is not easy to build because of interoperability issues, therefore the current focus on the cloud markets is mostly on the single provider and multiple users model.

Recently, several researchers investigated the design of online mechanisms in different contexts. Parkes and Singh [9] designed a Markov Decision Process-based online mechanism and later on provided an approximate solution for the same model for large problems [10]. Hajiaghay *et al.* [11] proposed the idea of automated online mechanism design. Fundamentals of online mechanism design are covered in [4]. Carroll and Grosu [12] designed an online mechanism for scheduling malleable parallel jobs on parallel systems. They considered preemption of jobs in their model in order to provide for jobs with higher valuation that are submitted later than the currently allocated jobs.

This paper is an outcome of our ongoing effort on improving the efficiency of VM allocation and provisioning in clouds. In our previous research, we designed three combinatorial auction-based mechanisms for VM allocation CA-LP, CA-GREEDY [13], and CA-PROVISION [14]. CA-GREEDY and CA-LP solve the VM allocation problem, where resources are assumed to be pre-provisioned as a fixed number of VM instances of different types. On the other hand, CA-PROVISION extends this model and allows dynamic provisioning and allocation of VM instances. However, these mechanisms work with full information about all user requests accumulated over a period of time, that is, they are offline mechanisms. They are run periodically to allocate resources to the users. In this paper, we address the problem of online mechanism design for VM provisioning and allocation in clouds.

Our contribution. We formulate the online VM provisioning and allocation problem and design an online mechanism to solve it. To the best of our knowledge, this is the first work that proposes an online mechanism for VM provisioning and allocation in clouds. We provide theoretical results that prove that the proposed mechanism is incentive compatible and runs in a reasonable amount of time. We perform extensive experiments and show that this mechanism improves the efficiency of allocation of VM instances in clouds.

Organization. The rest of the paper is organized as follows. In Section II, we formulate the problem of online VM provisioning and allocation. In Section III, we introduce the basic concepts of mechanism design in general and online mechanism design in particular in the context of the problem we are addressing. In Section IV, we present our proposed online mechanism for VM provisioning and allocation. In Section V, we characterize the properties of the proposed mechanism. In Section VI, we investigate the performance of the proposed mechanism by extensive simulation experiments. In Section VII, we conclude the paper.

II. VM INSTANCE ALLOCATION PROBLEM

We consider a cloud provider that provisions its computing resources into m different types of VM instances. We denote

these types of VM instances by $VM_1, \dots, VM_m$. With each type VM_i we associate a ‘weight’ $w_i \in \mathbb{R}_+^*$ which specifies the relative computing power of VM_i with respect to the most powerful VM instance offered by the cloud provider (i.e., the instance of type VM_1). We consider, without loss of generality, that $w_1 = 1$ and $w_1 \leq \dots \leq w_m$. The total power of the computing resources available for allocation is denoted by M , and is defined as the equivalent power of M instances of type VM_1 . For example, let us consider a cloud provider that offers VM instances of three types: small (VM_1), medium (VM_2), and large (VM_3). The configurations of these VMs are as follows: $VM_1 \equiv$ (one 2GHz processor, 4GB memory, 1TB hard drive), $VM_2 \equiv$ (two 2GHz processors, 8GB memory, 2TB hard drive), and $VM_3 \equiv$ (four 2GHz processors, 16GB memory, 4TB hard drive). For this setting, the weights of the VMs would be $w_1 = 1$, $w_2 = 2$, and $w_3 = 4$. If, for example, the cloud provider has enough resources to provision 100,000 VM_1 instances, then $M = 100,000$. The goal of the cloud provider is to dynamically provision its available resources into VM instances and allocate them to users efficiently and at the same time maximize its profit.

There are n users $j = 1, \dots, n$ who submit their requests (or ‘bids’) to the cloud provider in order to secure bundles of VM instances for certain amount of time in order to execute their applications (jobs) on the cloud. A user j is characterized by her ‘type’ $\theta_j = (r_1^j, \dots, r_m^j, a_j, l_j, d_j, v_j) \in \Theta_j$, where Θ_j is her type space. Here, $(r_1^j, \dots, r_m^j)$ is the bundle of VM instances requested by user j , where $r_i^j \in \mathbb{N}$ is the number of VM_i instances requested by user j . We also denote the bundle by S_j and its total weight by $s_j = \sum_{i=1}^m w_i r_i^j$. The other components of user j ’s type θ_j are: $a_j \in \mathbb{N}$, the arrival time (i.e., the time at which user j submits her bid); $l_j \in \mathbb{N}$, the amount of time the requested bundle must be allocated for; and, $d_j \in \mathbb{N}$, the deadline for user j ’s job completion. We denote by δ_j , the time by which the bundle must be allocated to the user in order for her job to complete its execution. That is, $\delta_j = d_j - l_j$. The last component of θ_j is $v_j \geq 0$, the value user j receives if her requested bundle of VMs is allocated within time δ_j . We assume that the users are *single-minded*, i.e., each user j requires that either her requested bundle S_j be allocated to her and she pays for it, or she does not get any allocation and pays nothing.

In the following we give an example to clarify the meaning of the components of the user type. Suppose user j requires two VM_1 and one VM_3 instances for five hours to complete a particular application. Hence, she requests the bundle $S_j = (2, 0, 1)$ with a total weight $s_j = 2 + 4 = 6$. She submits her bid at time $t = 0$ and must get her job done by time $t = 10$. Hence, $a_j = 0$, $l_j = 5$, and $d_j = 10$. There is no use of the requested bundle if she gets it after $t = 5$, therefore $\delta_j = 5$. Finally, suppose the successful completion of the application yields a value of \$50 to her, hence the valuation of her bundle is $v_j = 50$. Altogether, user j ’s type is $\theta_j = (2, 0, 1, 0, 5, 10, 50)$. We also call it her ‘true bid’, because she may choose to report a different type from her type space (i.e., all possible combinations of the values) to the cloud provider

if she benefits by doing so.

The cloud provider allocates bundles of VM instances to users by dynamically provisioning the available resources and charges them money. The provider must decide the allocation of a bundle S_j within the time interval $[a_j, \delta_j]$. If a user is not granted the allocation within this interval, her request is declined and she withdraws her request. Formally, the cloud provider computes an allocation set $A \subseteq \Theta \times \mathbb{N}$ and a payment set $P \subseteq \Theta \times \mathbb{R}_+$. A tuple $(\theta_j, t_j) \in A$ represents that bundle S_j of user j of type θ_j has been allocated at time t_j , where $a_j \leq t_j \leq \delta_j$. A tuple $(\theta_j, p_j) \in P$ says that user j of type θ_j has to pay p_j to the cloud provider. For the above user j , if she is granted the allocation at time $t = 3$ and charged \$30, we would have $(\theta_j, 3) \in A$ and $(\theta_j, 30) \in P$. Users who do not get the requested allocation pay zero.

We also consider that the cloud provider decides about the provisioning and allocation online, i.e., whenever some users and resources are available. Once a bundle S_j is allocated to a user j , it will not be reclaimed until l_j units of time after the allocation. Due to the limited resources, the cloud provider cannot allocate the VM bundles to all users at any given time. Also, unallocated users are going to potentially leave the system when their respective δ_j time has passed and they will not contribute to the provider's revenue. Therefore, the challenge of the cloud provider is to make provisioning and allocation decisions dynamically while trying to maximize its profit.

Since very little is known about revenue maximization in the mechanism design literature, mechanisms are usually designed with the goal of maximizing the sum of the valuations of the users [15]. Thus a reasonable goal is to allocate the VM instances so that the sum of the valuations of the users who receive their bundles is maximized. We formulate the *Online VM Provisioning and Allocation Problem (OVMPA)* as follows.

$$\max \sum_{j: (\theta_j, t_j) \in A} v_j \quad (1)$$

subject to:

$$\sum_{j: \theta_j \in \tilde{N}^{(t)}} s_j \leq M \quad (2)$$

Equation (1) is the objective of the problem, that is, the cloud provider maximizes the sum of the values of the users who obtain their requested bundles. The constraint in equation (2) says that at any given time t the allocation is limited to M , the total amount of resources available. Here,

$$\tilde{N}^{(t)} = \{j \mid (\exists t_j \leq t : (\theta_j, t_j) \in A) \wedge (t_j + l_j > t)\}$$

is the set of users who have been allocated prior to or at t and have yet to complete their allotted time slot at time t .

A straight-forward solution to the above problem would be to select users with the highest values, and then charge them their reported values. However, the users are rational and they may misreport their types if it benefits them to do so. For example, a user may report a lower valuation to pay less or a higher valuation to enhance her chance of winning. If the cloud

provider prefers jobs with an earlier deadline while breaking a tie for values, users may choose to report a deadline that is earlier than their actual deadline. Since this information is private, the provider needs to employ a mechanism to compute the allocation and payment based on the users' reported values in such a way that the system-wide goals set by the provider are achieved.

Therefore, the problem of designing an online mechanism for solving the OVMPA is as follows. Design an online mechanism that computes an allocation set A and payment set P on the problem space $\Theta = \Theta_1 \times \dots \times \Theta_n$ to maximize the objective function in Equation (1) satisfying the constraint given in Equation (2). The information reported by the users to the mechanism is denoted as $\hat{\theta} = (\hat{\theta}_1, \dots, \hat{\theta}_n) \in \Theta$, where $\hat{\theta}_j \in \Theta_j$. The goal of the mechanism is to compute an efficient allocation even if $\hat{\theta}_j \neq \theta_j$ and calculate payments in a way so that it provides incentives to the users to report their true types.

III. ONLINE MECHANISM DESIGN FRAMEWORK

A mechanism $\mathcal{M} = \{\mathcal{A}, \mathcal{P}\}$ is a set of functions $\mathcal{A}$ for computing the allocation and $\mathcal{P}$ for computing the payment for each user. Here, $\mathcal{A} : \Theta \rightarrow A$ and $\mathcal{P} = (p_1(\cdot), \dots, p_n(\cdot))$, where $p_j : \Theta \rightarrow \mathbb{R}$ for $j = 1, \dots, n$. In the context of OVMPA, the allocation function $\mathcal{A}$ computes the allocation set $A \subseteq \Theta \times \mathbb{N}$ from the bids reported by the users. The allocation set A is the set of the tuples (θ_j, t_j) , where j is the user receiving her requested bundle at time t_j . Function $p_j(\cdot)$ determines user j 's payment based on the bids of all users.

Each user j is characterized by a *valuation function* V_j defined as follows:

$$V_j(\mathcal{A}(\hat{\theta}), \theta_j) = \begin{cases} v_j & \text{if } (\theta_j, t_j) \in A \wedge t_j \leq d'_j \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

This means that the user receives the value v_j if she secures the requested bundle, and no value, otherwise. We quantify user j 's benefit through a *utility function* defined as the difference between the value she receives from the mechanism and the payment charged to her:

$$U_j(\mathcal{A}(\hat{\theta}), \theta_j) = V_j(\mathcal{A}(\hat{\theta}), \theta_j) - p_j(\hat{\theta}) \quad (4)$$

In the example presented above user j derives a utility of \$20 if she values the bundle at \$50, receives her bundle at time $t \leq 5$, and pays \$30 for it (as shown in Section II). If the mechanism decides not to allocate her the requested bundle, she receives a value of zero. In that case, if the mechanism does not charge her any payment, her utility will be zero as well.

We are interested in designing mechanisms which have two important properties, *incentive compatibility* and *individual rationality*. In the following, we denote by $\hat{\theta}_{-j} = (\hat{\theta}_1, \dots, \hat{\theta}_{j-1}, \hat{\theta}_{j+1}, \dots, \hat{\theta}_n) \in \Theta_{-j}$ the bids of all users except j . Recall that the true type of user j is θ_j and her reported type (or bid) $\hat{\theta}_j$ may not be equal to θ_j .

Definition 1 (Incentive compatibility): A mechanism $\mathcal{M}$ is *incentive compatible* if for all j , $\hat{\theta}_j \in \Theta_j$, and $\hat{\theta}_{-j} \in \Theta_{-j}$,

$$U_j(\mathcal{A}(\theta_j, \hat{\theta}_{-j}), \theta_j) \geq U_j(\mathcal{A}(\hat{\theta}_j, \hat{\theta}_{-j}), \theta_j)$$

That is, the users maximize their utilities by reporting their true types to the mechanism, irrespective of the other users' bids. This is a very important property, because if satisfied, the users participating in the mechanism will not have incentives to report other types than their true types to the mechanism. That is, truthful reporting is their best strategy.

Definition 2 (Individual rationality): A mechanism is *individually rational* if a user never incurs a loss by reporting her true type. Formally, for all j , true type $\theta_j \in \Theta_j$, and $\hat{\theta}_{-j} \in \Theta_{-j}$,

$$U_j(\mathcal{A}(\theta_j, \hat{\theta}_{-j}), \theta_j) \geq 0$$

That is, regardless of other users' bids, a user reporting her type truthfully will never obtain a negative utility by participating in the mechanism. This is a very important property of a mechanism since it encourages users to voluntarily participate in the mechanism.

In order to obtain an incentive compatible mechanism the allocation function $\mathcal{A}$ must be *monotone* and the payment function $\mathcal{P}$ must implement the *critical value payment* [16].

We introduce a preference relation, $\succeq$, on the set of user bids. For example, $\hat{\theta}'_j \succeq \hat{\theta}_j$ means that bid $\hat{\theta}'_j$ is more preferred to the mechanism than bid $\hat{\theta}_j$. In our context $\hat{\theta}'_j \succeq \hat{\theta}_j$ if $\hat{s}'_j \leq \hat{s}_j$, $\hat{a}'_j \leq \hat{a}_j$, $\hat{l}'_j \leq \hat{l}_j$, $\hat{d}'_j \geq \hat{d}_j$ and $\hat{v}'_j \geq \hat{v}_j$.

Definition 3 (Monotone allocation function): An allocation function $\mathcal{A}$ is monotone if for all j , $\hat{\theta}_j, \hat{\theta}'_j \in \Theta_j$, and $\hat{\theta}_{-j} \in \Theta_{-j}$,

$$(\hat{\theta}'_j \succeq \hat{\theta}_j) \wedge (\exists t_j : (\hat{\theta}_j, t_j) \in \mathcal{A}(\hat{\theta}_j, \hat{\theta}_{-j})) \\ \Rightarrow \exists t'_j : (\hat{\theta}'_j, t'_j) \in \mathcal{A}(\hat{\theta}'_j, \hat{\theta}_{-j})$$

This means that if user j gets her requested bundle by declaring type $\hat{\theta}_j$, she will also get the resources by declaring type $\hat{\theta}'_j$, where type $\hat{\theta}'_j$ is more preferred over $\hat{\theta}_j$.

Definition 4 (Critical value payment): Critical value v_j^c for user j is defined as

$$v_j^c = \arg \min_{v'_j \geq 0} (\exists t_j : \\ (\hat{\theta}_j = (r_1^j, \dots, r_m^j, a_j, l_j, d_j, v'_j), t_j) \in \mathcal{A}(\hat{\theta}_j, \hat{\theta}_{-j}))$$

This means that the critical value for a user is the minimum amount she needs to report to the mechanism in order to receive her requested bundle. The payment function should charge a user her critical value in order to obtain an incentive compatible mechanism. In the following section, we design an online mechanism that solves OVMPA.

IV. ONLINE MECHANISM FOR VM ALLOCATION

We now present our mechanism for Online VM Provisioning and Allocation (MOVMPA) that solves the OVMPA problem.

Algorithm 1 Mechanism MOVMPA

Input: $Event, A, P$

Returns: A, P

```

1:  $t \leftarrow$  Current time
2:  $N^{(t)} \leftarrow \{\hat{\theta}_j \in N \mid \hat{a}_j \leq t \wedge \neg \exists t_j < t : (\hat{\theta}_j, t_j) \in A\}$ 
3:  $\tilde{N}^{(t)} \leftarrow \{\hat{\theta}_j \mid \exists t_j < t : (\hat{\theta}_j, t_j) \in A \wedge t_j + \hat{l}_j > t\}$ 
4:  $M^{(t)} \leftarrow M - \sum_{j: \hat{\theta}_j \in \tilde{N}^{(t)}} \hat{s}_j$ 
5: if  $M^{(t)} = 0 \vee N^{(t)} = \emptyset$  then return
6:  $R \leftarrow M^{(t)}$ 
7:  $A^{(t)} \leftarrow \text{MOVMPA-ALLOCATE}(t, N^{(t)}, R)$ 
8:  $A \leftarrow A \cup A^{(t)}$ 
9:  $P^{(t)} \leftarrow \{(\hat{\theta}_j, \hat{v}_j) \mid (\hat{\theta}_j, t_j) \in A^{(t)}\}$ 
10:  $P \leftarrow P \cup P^{(t)}$ 
11:  $P \leftarrow \text{MOVMPA-PAYMENT}(t, P, N^{(t)}, \tilde{N}^{(t)}, M^{(t)})$ 
12:  $N^P \leftarrow \{\hat{\theta}_j \in N \mid t' < \delta_j \leq t\}$ 
13: for each  $\hat{\theta}_j \in N^P$  do
14:   if  $\exists p_j : (\hat{\theta}_j, p_j) \in P$  then
15:     user  $j$  pays  $p_j$ 
16:   else
17:     user  $j$  pays 0
18:   end if
19: end for
20:  $t' \leftarrow t$ 
21: return  $A, P$ 
```

A. Mechanism MOVMPA

The mechanism MOVMPA is structured as an event handler, which is invoked when a new bid arrives or a user completes her time for the allocated bundle and releases the VM instances to the provider. We assume that the information about the users and the resources is made available to the mechanism via some standard protocol. MOVMPA uses this information to determine the set of users and resources available for allocation at the current time and calls the allocation and the payment functions. We present the MOVMPA mechanism in Algorithm 1.

MOVMPA takes as input the event, the current allocation set and the payment set. An event is either a release of resource or an arrival of a user request. We assume that the system stores these two sets and passes them to MOVMPA when it is invoked. MOVMPA updates the sets and returns back to the system. In lines 1-4, MOVMPA sets the current time to t and initializes three variables as follows: $N^{(t)}$, the set of bids of the users that have not been allocated so far; $\tilde{N}^{(t)}$, the set of bids of the users that have been allocated in the past and have not finished their time of allocation; and, $M^{(t)}$, the total weight of the resources that are available for allocation at time t . The mechanism proceeds only if resources and users are available. It calls the allocation function MOVMPA-ALLOCATE with the user bids that have not been allocated yet and the resources that are available at time t . MOVMPA-ALLOCATE returns set $A^{(t)}$, the set of users who would receive their requested bundles at time t (line 7).

Next, MOVMPA updates the overall allocation set A to $A^{(t)}$. The bids in $A^{(t)}$ are also inserted into the payment set, with $p_j = \hat{v}_j$ as their initial payment. However, this payment is updated by calling the payment function MOVMPA-PAYMENT

Algorithm 2 MOVMPA-ALLOCATE

Input: $t, N^{(t)}, R$
Returns: $A^{(t)}$

```

1:  $A^{(t)} \leftarrow \emptyset$ 
2: Sort all  $\hat{\theta}_j \in N^{(t)}$  in non-increasing order of  $\rho_j$ 
3: for each  $\hat{\theta}_j \in N^{(t)}$  in sorted order do
4:   if  $s_j \leq R$  then
5:      $A^{(t)} \leftarrow A^{(t)} \cup (\hat{\theta}_j, t)$ 
6:      $R \leftarrow R - \hat{s}_j$ 
7:   end if
8: end for
9: return  $A^{(t)}$ 

```

(line 11). In fact, the payment of user j is going to be updated several times until $t = \delta_j$, i.e., until the time instance the user must receive allocation of the bundles she requires. Thus, MOVMPA-PAYMENT calculates the payments for these users and updates the payment set P . The next step for MOVMPA is to determine the set N^P of the bids of users j for whom the current time has gone past their respective δ_j times (line 12). However, this set only includes the users whose δ_j has passed after t' , where t' is the time of the last invocation of MOVMPA (line 12). If user j has been already provided with her bundle, her payment will no longer change and the payment that is up to date at the current time will be her final payment charged to her. If user j has not received her requested bundle until t and $t > \delta_j$, she will not be successful in getting an allocation to get her job done. In this case, user j will pay $p_j = 0$ (lines 14-19). The computation of the prices will be presented when we discuss the MOVMPA-PAYMENT function.

B. Allocation function

The allocation function MOVMPA-ALLOCATE is given in Algorithm 2. In order to describe this function we define a new parameter called ‘bid density’, $\rho_j = \frac{\hat{v}_j}{\hat{s}_j \times \hat{l}_j}$. The intuition behind bid density is as follows. We can reformulate the VM allocation problem as the problem of allocating rectangles in a two-dimensional space of VM weight and time. The bid by user j for a bundle of VM instances of weight $\hat{s}_j$ for time $\hat{l}_j$ can be interpreted as requesting a rectangle with area $\hat{s}_j \times \hat{l}_j$ in that two-dimensional space, and user j values this area at $\hat{v}_j$. Hence, ρ_j is how much user j values a ‘unit area’ of the rectangular space. Obviously, the cloud provider is interested in users who want to pay more per unit of their resources per unit time. MOVMPA-ALLOCATE uses ρ_j to determine the relative values of the bids.

First, MOVMPA-ALLOCATE sorts all bids in non-increasing order of ρ_j s. Ties are broken in the following order: prefer earlier δ_j , smaller $\hat{l}_j$, and then smaller $\hat{s}_j$. Further ties are broken arbitrarily. Then the algorithm allocates bundles requested by these users while resources last. Finally, it returns the set $A^{(t)}$ of users who are selected for allocation at time instance t .

MOVMPA-ALLOCATE simply tries to maximize the sum of the reported valuations of the users who would be granted their requested bundles. In the case of a tie, by giving priority

Algorithm 3 MOVMPA-PAYMENT

Input: $t, P, N^{(t)}, \tilde{N}^{(t)}, M^{(t)}$
Returns: P

```

1: sort all  $\hat{\theta}_j \in N^{(t)}$  in non-increasing order of  $\rho_j$ 
2: for each  $(\hat{\theta}_j, p_j) \in P : \delta_j \leq t$  do
3:    $N'^{(t)} \leftarrow N^{(t)} \setminus \hat{\theta}_j$ 
4:   if  $\hat{\theta}_j \in \tilde{N}^{(t)}$  then
5:      $R' \leftarrow M^{(t)} + \hat{s}_j$ 
6:   else
7:      $R' \leftarrow M^{(t)}$ 
8:   end if
9:   for each  $\hat{\theta}_{j'} \in N^{(t)}$  do
10:     $\bar{v}_j \leftarrow \rho_{j'} \times \hat{s}_j \times \hat{l}_j$ 
11:     $\bar{\theta}_j \leftarrow (r_1^j, \dots, r_m^j, t, \hat{l}_j, \hat{d}_j, \bar{v}_j)$ 
12:     $A' \leftarrow \text{MOVMPA-ALLOCATE}(t, N'^{(t)} \cup \{\bar{\theta}_j\}, R')$ 
13:    if  $(\bar{\theta}_j, t) \in A'$  then
14:       $p_j \leftarrow \min(p_j, \bar{v}_j)$ 
15:    end if
16:  end for
17:   $\bar{\theta}_j \leftarrow (r_1^j, \dots, r_m^j, t, \hat{l}_j, \hat{d}_j, 0)$ 
18:   $A' \leftarrow \text{MOVMPA-ALLOCATE}(t, N'^{(t)} \cup \{\bar{\theta}_j\}, R')$ 
19:  if  $(\bar{\theta}_j, t) \in A'$  then
20:     $p_j \leftarrow 0$ 
21:  end if
22: end for
23: return  $P$ 

```

to users with a smaller δ_j , (i.e., users who need to leave the system earlier if they don’t get their bundles), it also makes sure that the highest possible number of users are served. For the same reason, in case of a tie with δ_j , priority is given to users who request the resources for a smaller amount of time.

C. Payment function

We give the payment function MOVMPA-PAYMENT in Algorithm 3. The payment function requires as input the current time t , the payment set P , the amount of resources available before calling the allocation function $M^{(t)}$, the set of users who were considered at the allocation function $N^{(t)}$, and, the set of users who are occupying resources at the current time t , $\tilde{N}^{(t)}$. It is worth mentioning that users that were granted the requested bundle at time t also belong to the set $N^{(t)}$.

The main idea of MOVMPA-PAYMENT is to calculate the critical payment of each user j with $\delta_j \leq t$ as if their time of arrival is t . By repeatedly calling this function at each event, MOVMPA ensures that the critical payment of a user j is calculated every time an event occurs between $\hat{a}_j$ and δ_j . Formally, MOVMPA-PAYMENT calculates the critical value

$$v_j^{ct} = \arg \min_{v_j' \geq 0} (\exists t_j : (\hat{\theta}_j' = (\hat{r}_1^j, \dots, \hat{r}_m^j, t, \hat{l}_j, \hat{d}_j, v_j'), t_j) \in \mathcal{A}(\hat{\theta}_j, \theta_{-j})) \quad (5)$$

at time t , for all users j with $\delta_j \geq t$. Based on this critical value the MOVMPA mechanism computes the critical value as

$$v_j^c = \min_{t \in [\hat{a}_j, \delta_j]} v_j^{ct} \quad (6)$$

That is, v_j^c is the minimum value user j must report to get her requested bundle for any arrival time $a'_j \in [\hat{a}_j, d'_j]$.

MOVMPA-PAYMENT considers users from the payment set that have $\delta_j \leq t$. For each user j , the arrival time component of her type is set to t and her value is set to the values of each user to find the minimum value to be reported by user j in order to get her requested bundle (lines 2-16). If no such minimum value is found, the payment is set to zero (lines 17-21). One can also set this value to a predefined reserve price. Finally, the updated set P is returned to the mechanism.

The mechanism keeps updating P by calling MOVMPA-PAYMENT and charges the updated payment at time t to users j for which $\delta_j < t$. Users who were not allocated any resources pay zero.

V. PROPERTIES OF MOVMPA

In this section, we prove that the MOVMPA mechanism is incentive compatible and individually rational. We also perform competitive analysis and determine the runtime complexity of the mechanism. Due to space limitation we will only provide the sketches of the proofs.

Theorem 1: MOVMPA is an incentive compatible mechanism.

Proof: (Sketch) To prove this theorem we need to show that the allocation function is monotone and that the payment function implements the critical value payment. First, we show that the allocation is monotone. Clearly, if user j reports $\hat{v}'_j \geq \hat{v}_j$, her bid $\hat{\theta}'_j$ will be allocated if $\hat{\theta}_j$ is also allocated. Similarly, if a user gets the allocation by reporting $\hat{d}_j$, she will also get it by reporting $\hat{d}'_j \geq \hat{d}_j$. Proofs for the other parameters follow the same logic, based on the priorities set for ordering the reported types in MOVMPA-ALLOCATE (Subsection IV-B). Thus, the allocation function MOVMPA-ALLOCATE is monotone. Since we compute the minimum value that the users must report to get the allocation, the minimum value is the same for both $\hat{\theta}_j$ and $\hat{\theta}'_j$, given that the other users' reported types remain the same. Hence, j pays the same amount for reporting both types. This is the critical value payment that we calculate in MOVMPA-PAYMENT. Hence, MOVMPA is incentive compatible. ■

Theorem 2: The time complexity of MOVMPA is $O(n \log n)$.

Proof: The costliest operation in MOVMPA-ALLOCATE is sorting the users in line 2. This requires a runtime of $O(n \log n)$. However, n is the total number of users and each time the mechanism encounters only a fraction of them.

MOVMPA-PAYMENT seems rather complicated because it repeatedly calls MOVMPA-ALLOCATE. But in an actual implementation, we do not need to call the allocation function to determine the critical payment. For each user j that is a winner, we need to consider only the users j' who lose at time t , in non-increasing order of their bids, and check whether user j' can win her bid if j does not participate. This check is done in a single comparison of resources occupied by j and the resources requested by j' . When we find first such

j' , user j 's payment is $\hat{v}_{j'}$. We need to sort the losing bids only once at the beginning of the algorithm, which again has a running time of $O(n \log n)$, where n is a much larger number than what that algorithm encounters in practice. We present the payment algorithm in this fashion in order to show the underlying concepts. ■

Now we prove the competitive ratio of MOVMPA. The *competitive ratio* of an online algorithm is $c > 1$ if the ratio of its performance to an optimum offline algorithm is $1/c$. We prove the competitive ratio of MOVMPA by choosing an input that produces the worst-case scenario for MOVMPA.

Theorem 3: MOVMPA mechanism has a competitive ratio of M .

Proof: Consider two bids θ_1 and θ_2 where $s_1 = 1$, $s_2 = M$, $(a_1, l_1, d_1, v_1) = (0, l, l, v)$, and $(a_2, l_2, d_2, v_2) = (1, l, l + 1, Mv + \epsilon)$, where $l > 1$. MOVMPA will allocate user 1 her requested bundle that she will release at time $l > 1$. Now, since for θ_2 , $a_2 + l_2 = 1 + l = d_2$, user 2 must get the allocation at time $a_2 = 1$, otherwise she withdraws her bid because she cannot satisfy her deadline. Therefore, MOVMPA will allocate VM instances only to user 1 and obtain a value of v (Equation (1)). On the other hand, an optimum offline algorithm will allocate resources to user 2 and obtain a value of $Mv + \epsilon > M \times v$. Hence, the competitive ratio of MOVMPA is M . ■

VI. EXPERIMENTAL RESULTS

In this section, we evaluate MOVMPA through simulation experiments. We compare MOVMPA with a good offline mechanism to identify its strengths and weaknesses.

A. Experimental Setup

We choose the offline mechanism CA-PROVISION from our previous work [14] to compare with MOVMPA. CA-PROVISION solves the dynamic VM provisioning and allocation problem in clouds. It is invoked at regular intervals, (e.g., hourly). When it is invoked, it considers all the bids collected during the past interval and runs a combinatorial auction to determine the set of winners, their payments, and the number of different VM instances to provision. The VM instances are allocated for one time interval and users requiring subsequent access must continue to bid until their time requirement is fulfilled. A bundle allocated to a user at current period may be preempted later if the demand increases.

The input data we use are collected from The Parallel Workload Archive [17]. We show the information about the workload logs and some statistics associated with them in Table I. The table shows the name of workload logs, the duration for which the logs were collected, the number of jobs, and the total log hours in the first four columns. The table also shows the total number of processors of the system the logs were generated from. We assume that the weight of a VM instance corresponds to the number of processors allocated to it. Hence, the total number of processors of a system represents the total weight of the computing resources M . We denote the total computing resources allocated to a workload ω by

TABLE I
WORKLOAD LOGS

Logfile (ω)	Collection period	Total Jobs	Total hours	Total no. of Procs. (M_ω)	Jobs / hour (J_ω)	Avg. runtime in hour (T_ω)	Avg procs. per job (P_ω)	Normalized load (η_ω)
DAS2-fs0-2003	12 months	225,711	8744	144	25.81	1.09	10.27	2.01
DAS2-fs1-2003	12 months	40,315	8633	64	4.67	1.23	8.38	0.75
DAS2-fs2-2003	12 months	66,429	8760	64	7.58	1.29	9.45	1.44
DAS2-fs3-2003	12 months	66,737	8712	64	7.66	1.17	4.96	0.69
DAS2-fs4-2003	11 months	33,795	7963	64	4.24	1.67	3.66	0.40
LLNL-uBGL-2006	7 months	112,611	5339	2,048	21.09	1.25	575.79	7.41
LPC-EGEE-2004	9 months	234,889	5728	140	41.00	1.80	1	0.53

TABLE II
SIMULATION PARAMETERS

Parameter	Notation	Values
Average bundle weight	s_{avg}	from workload (P_ω)
Average time per job	l_{avg}	from workload (T_ω)
Average deadline factor	d_{avg}	3
Average valuation	v_{avg}	5, 10, 20
Cost of running and idle VM	c_R, c_I	1, 0.5

M_ω . The next four columns in Table I are statistical data and are defined as follows. Jobs/hour (J_ω) is the average number of jobs submitted to the system per hour. The average runtime (T_ω) and the average number of processors per job (P_ω) are calculated over all records of the workload log. The last column shows η_ω , the ‘normalized load’ of workload ω , which is computed as follows

$$\eta_\omega = \frac{J_\omega \times T_\omega \times P_\omega}{M_\omega}.$$

Normalized load measures the average amount of resources requested against each unit of computing resources available. It helps us in ranking the otherwise heterogeneous workloads and also explaining some of the experimental results.

We setup the simulation experiments as follows. We assume that a cloud provider provisions its resources into four types of VM instances, $VM_1, \dots, VM_4$ with weights $\mathbf{w} = (1, 2, 4, 8)$. User’s bids ($\theta_j = r_1^j, \dots, r_m^j, a_j, l_j, d_j, v_j$) are generated using a workload file ω and its pre-computed statistics that are shown above. The time of arrival a_j of user j is taken from the log file. $s_j = \sum_{i=1}^m w_i r_i^j$, i.e., the total weight of the requested bundle is chosen from an exponential distribution with mean $s_{avg} = P_\omega$, the average number of processors per job in workload ω . We arbitrarily set the values of r_i^j s once we determine s_j . The required time for allocation, l_j , is also determined using exponential distribution with mean l_{avg} , which is derived from T_ω , average runtime of the jobs in workload ω . Deadline d_j and valuation v_j are computed as $d_j = a_j + l_j \times \exp(d_{avg})$ and $v_j = \exp(v_{avg})$. We show the values chosen for d_{avg} and v_{avg} in Table II. We bring more variation in the input data by randomly choosing about 50% users and multiplying any of their bid parameters by 2.

CA-PROVISION computes a reserve price $v_{res} = c_R - c_I$ from the cost parameters associated with a running or idling unit VM instance. Although we did not present MOVMPA with a reserve price, it can be easily incorporated by discarding users below the reserve price and then charging a winning user

the reserve price instead of zero at line 20 of Algorithm 3. In our experiments, the reserve price for MOVMPA is the same as that used for CA-PROVISION. We show all the parameter values in Table II. We run 18 experiments per workload log with different combinations of parameters.

B. Analysis of Results

We summarize the results per workload in Figure 1. In this figure, we show the workload logs with their normalized load on the horizontal axis and the percent of users served, the average revenue per served user, and the average utility per served user on the vertical axis. In Figure 1a, we observe that MOVMPA serves a higher percentage of users than CA-PROVISION. The most significant gain is with logs LLNL-uBGL-2006 (normalized load 7.41) and DAS2-fs0-2003 (normalized load 2.01), which are the highest among the workloads used here. The percentage of served users is nearly doubled for the case of DAS2-fs0-2003 workload and it increases more than six-fold for LPC-EGEE-2004 workload. This is because in CA-PROVISION, even if there are available resources, a user must wait until the next auction time. By that time more users may arrive and she may lose the auction. On the other hand, preemption also makes some users leave even if they receive the allocation in some auction but still require the resources for additional time to complete their task. Since the above two workloads generate high bid density, this leads to an increase in the number of auctions and MOVMPA could accommodate more bids, thanks to its online design.

On the other hand, in Figure 1b, we see a decline in the average revenue generated from each served user. As we discussed in Section V, online mechanisms cannot obtain an optimal outcome because they determine the allocation based on incomplete information. This leads to a suboptimal value of the social welfare given by Equation (1). An intuitive explanation is that by deciding about an allocation as soon as possible, MOVMPA helps a user avoid facing competition with the future bidders. With CA-PROVISION, users must compete with other users that bid during the same period and hence the price of the items increases. But the increased number of served users offsets some of this revenue loss.

However, we see in Figure 1c that MOVMPA produces comparable results for the average utility of the served users. Although it is expected that by paying less, users would gain higher utility, but that will happen if both auctions select the same set of users for allocation. This is not possible

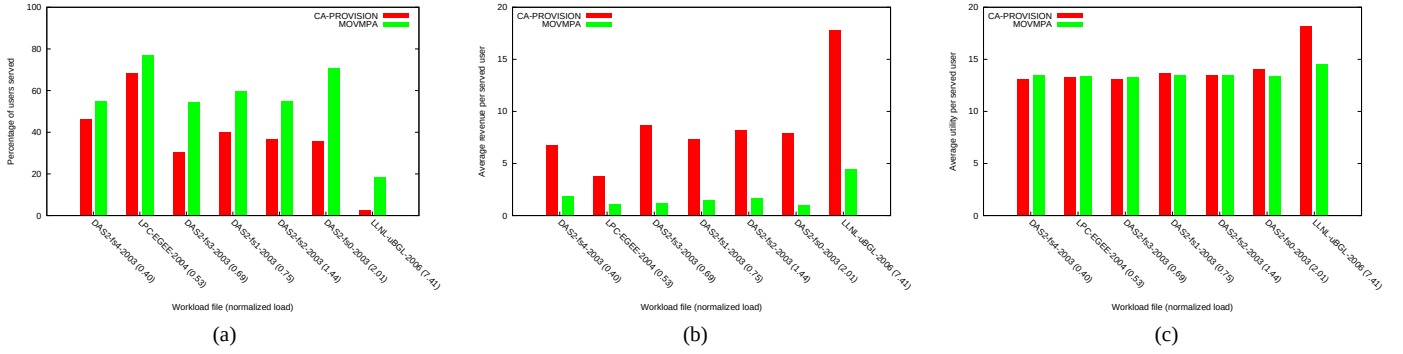


Fig. 1. Overall results comparing CA-PROVISION and MOVMPA: (a) percent of users served; (b) average revenue per served user; and (c) average utility per served user vs. the workload logs.

because of two reasons: CA-PROVISION preempts a user with low valuation for one with higher valuation, but MOVMPA allocates VMs to a user for the entire period she requested for them. This may lead to losing a high valued user who arrived at a time when there is no resources available. On the other hand, MOVMPA allocates more users than CA-PROVISION, which means it accommodates users with low valuations and these users contribute to the low average utility.

In summary, we claim that MOVMPA improves the overall cloud experience for both the users and the providers. The cloud provider benefits by serving more users, gaining user satisfaction, and eventually increasing its overall revenue.

VII. CONCLUSION

In this paper, we designed an online mechanism for VM provisioning and allocation in clouds. The mechanism provisions and allocates VM instances whenever enough resources and matching bids are available. We proved that the mechanism is incentive compatible, has a competitive ratio of M and runs in polynomial time. We performed extensive experiments to determine the strength and weaknesses of the mechanism. The mechanism increases the number of users served but at the cost of decreased average revenue. However, the loss in average revenue may be offset by the increased number of served users. We conclude that the proposed mechanism is a good choice for provisioning and allocation of VM instances in clouds.

ACKNOWLEDGMENT

This work was supported in part by NSF grants DGE-0654014 and CNS-1116787.

REFERENCES

- [1] Microsoft, "Purchase options - pricing - Windows Azure," <http://www.microsoft.com/windowsazure/offers/>.
- [2] Amazon, "Amazon EC2 pricing," <http://aws.amazon.com/ec2/pricing/>.
- [3] —, "Amazon EC2 spot instances," <http://aws.amazon.com/ec2/spot-instances/>.
- [4] D. C. Parkes, "Online mechanisms," in *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, Éva Tardos, and V. V. Vazirani, Eds. Cambridge University Press, 2007, ch. 16.
- [5] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: Pricing in the cloud," in *Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing*, 2010.

- [6] W.-Y. Lin, G.-Y. Lin, and H.-Y. Wei, "Dynamic auction mechanism for cloud resource allocation," in *Proc. 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 591–592.
- [7] M. Risch, J. Altmann, L. Guo, A. Fleming, and C. Courcoubetis, "The GridEcon platform: A business scenario testbed for commercial cloud services," in *Proc. Workshop on Grid Economics and Business Models*, 2009, pp. 46–59.
- [8] J. Altmann, C. Courcoubetis, G. D. Stamoulis, M. Dramitinos, T. Rayna, M. Risch, and C. Bannink, "GridEcon: A market place for computing resources," in *Proc. Workshop on Grid Economics and Business Models*, 2008, pp. 185–196.
- [9] D. C. Parkes and S. Singh, "An mdp-based approach to online mechanism design," in *Proc. 17th Annual Conference on Neural Information Processing Systems*, 2003.
- [10] D. C. Parkes, S. P. Singh, and D. Yanovsky, "Approximately efficient online mechanism design," in *Proc. 18th Annual Conference on Neural Information Processing Systems*, 2004.
- [11] M. Hajiaghayi, R. Kleinberg, and T. Sandholm, "Automated online mechanism design and prophet inequalities," in *Proc. National Conference on Artificial Intelligence*, 2007.
- [12] T. E. Carroll and D. Grosu, "Incentive-compatible online scheduling of malleable parallel jobs with individual deadlines," in *Proc. 39th Intl. Conf. on Parallel Processing*, 2010, pp. 418–425.
- [13] S. Zaman and D. Grosu, "Combinatorial auction-based allocation of virtual machine instances in clouds," in *Proc. 2nd IEEE Intl. Conf. on Cloud Computing Technology and Science*, 2010, pp. 127–134.
- [14] —, "Combinatorial auction-based dynamic VM provisioning and allocation in clouds," in *Proc. 3rd IEEE Intl. Conf. on Cloud Computing Technology and Science*, 2011, pp. 107–114.
- [15] N. Nisan, T. Roughgarden, Éva Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge University Press, 2007.
- [16] A. Mu'alem and N. Nisan, "Truthful approximation mechanisms for restricted combinatorial auctions," in *Proc. 18th National Conference on Artificial Intelligence*, 2002, pp. 379–384.
- [17] D. G. Feitelson, "Parallel Workloads Archives: Logs," <http://www.cs.huji.ac.il/labs/parallel/workload/logs.html>.