J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Combinatorial auction-based allocation of virtual machine instances in clouds

Sharrukh Zaman, Daniel Grosu *

Department of Computer Science, Wayne State University, 5057 Woodward Avenue, Detroit, MI 48202, USA

ARTICLE INFO

Article history:
Received 4 March 2012
Received in revised form
25 September 2012
Accepted 10 December 2012
Available online xxxx

Keywords:
Cloud computing
Combinatorial auction
Virtual machine allocation
Resource allocation

ABSTRACT

Most of the current cloud computing providers allocate virtual machine instances to their users through fixed-price allocation mechanisms. We argue that combinatorial auction-based allocation mechanisms are especially efficient over the fixed-price mechanisms since the virtual machine instances are assigned to users having the highest valuation. We formulate the problem of virtual machine allocation in clouds as a combinatorial auction problem and propose two mechanisms to solve it. The proposed mechanisms are extensions of two existing combinatorial auction mechanisms. We perform extensive simulation experiments to compare the two proposed combinatorial auction-based mechanisms with the currently used fixed-price allocation mechanism. Our experiments reveal that the combinatorial auction-based mechanisms can significantly improve the allocation efficiency while generating higher revenue for the cloud providers.

1. Introduction

Cloud computing enables individuals and small to medium enterprises satisfy their computational needs with no or minimum upfront costs of acquiring hardware and software. On the other hand, cloud providers benefit by commercializing their huge computing resources through the cloud computing platform. A cloud computing platform abstracts the underlying physical resources from the users by providing them with the view of virtual machines (VMs). This enables easy management and pricing of the resources. Currently, the majority of cloud providers price their computing resources based on the 'size' of the VM instances offered. They define different types of VM instances by specifying the number and speed of processors, the memory size, the bandwidth allocation, etc. There are two ways to 'purchase' the VM instances: pay as you go and long term contract. In both cases users pay fixed prices per unit of time for using the resources; the only difference is that by committing to a long term contract they usually pay less per unit of time for using the same resource.

We argue that the currently used fixed-price schemes for allocation and pricing of resources have several drawbacks. First, they are not economically efficient [26], that is, they cannot guarantee that the user who values a bundle of VM instances the most gets it. Second, fixed prices do not necessarily reflect the equilibrium prices that arise from market demand and supply. This may lead to lower than optimal revenue for the service providers. Finally, since in cloud computing platforms resources are sold for a period of time, it is desirable that user requests be evenly distributed throughout the day. In general, the current fixed-price methods do not provide users with incentives for demand shaping (i.e., selecting their execution time-frames in such a way that the system load is balanced over time). It is possible to modify a fixed-price mechanism so that it provides such incentives by setting up different fixed prices at different times of the day based on historical demand. This needs statistical analysis and adjustment of prices as the demand pattern changes making it hard to achieve dynamic price adjustment.

The inefficiencies in solving the resource allocation problem in clouds mentioned above can be best addressed by employing auction-based mechanisms. Among different types of auctions, the *combinatorial auctions* are the most suitable for solving the VM pricing and allocation problem in clouds. In combinatorial auctions, the participants bid for bundles of items rather than individual items [10]. This enables bidders to express their valuations in a more meaningful way, especially when the items they require are complementary to each other. To illustrate this, let us consider the following example. A cloud service provider offers 'small' and 'large' VM instances. Suppose a user wants to deploy a three-tier web application on the cloud. The application needs a database server, an application server, and two web servers. The database and application servers are heavy weight and therefore the user prefers large instances for them. The web servers are light weight and can be hosted on two small instances. Thus, a user needs to run an application which requires two small and two large VM instances. It is more meaningful for her to be able to bid for the

* Corresponding author.
E-mail addresses: sharrukh@wayne.edu (S. Zaman), dgrosu@cs.wayne.edu (D. Grosu).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.12.006

entire bundle she needs rather than bidding for each VM instance separately. Bidding for each VM instance separately involves the risk of ending up acquiring just a subset of her required set of VM instances. The motivation behind our work is that by designing and deploying combinatorial auction-based mechanisms for allocating VM instances, the cloud providers can guarantee fairness to their users as well as enjoy higher revenues and a balanced load on their systems over time. Load balancing over time is actually a side-effect of using auctions for allocating VM instances. Users with lower valuations for the VM instances will choose a time-frame that does not conflict with that of 'high valuation users'. For example, if large businesses request resources during the daytime, individual users may consider that the nighttime slots are more suitable for them, thus balancing the load of the system over time.

Application of auctions, however, is not entirely new to the cloud computing community. After allocating computing resources for the long-term and on demand users, Amazon EC2 sells the remaining virtual machines (instances) through an auction called Spot Instances [2]. In this auction, the bidders specify their demand (i.e., the number and the type of instances) and the maximum price they are willing to pay. Amazon periodically runs the auction with active bidders to determine the current price and then users with bids higher than that price are provided with their desired instances. All users pay the same price per instance which is computed by the auction. A user getting the allocation may be terminated at a later point if the auction-determined price goes beyond her bid. This approach is different from combinatorial auctions because one single price is determined based on market supply and demand (i.e., equilibrium) and all bidders pay the same price per item regardless of how much they value the item. On the other hand, in combinatorial auctions, each winning bidder's payment is calculated based on her and other bidders' valuations. Another important difference is that the Spot Instances auction does not support bidding on bundle of instances, while combinatorial auctions were specifically designed to work with such bundles. From Amazon's initial effort of using auction-based allocation, it is reasonable to expect that cloud providers will be interested in more efficient allocation and pricing schemes in the near future. Combinatorial auctions will clearly be one of the most desirable allocation schemes in this regard. This is supported by their successful application in various fields ranging from selling wireless spectrum to transportation procurement for large industries [10].

1.1. Our contribution

We formulate the problem of allocating VM instances in clouds as a combinatorial auction problem. The objective of this problem is to efficiently allocate VM instances of several types to several users requesting a set of VM instances of different types. To solve this problem, we propose two combinatorial auction-based allocation mechanisms. These two mechanisms are obtained by extending the mechanisms proposed by Archer et al. [5] and Lehmann et al. [16]. The mechanism proposed by Archer et al. [5] considers a combinatorial auction problem where a user can include at most one item of a particular type in her requested bundle. We relax this condition to allow users requesting more than one item of a given type. Also, the mechanism proposed by Archer et al. [5] is suitable for combinatorial auctions with many types of items where each type of items has few instances. We extend the mechanism so that it can be applied to the VM allocation problem where there are few types of items and many instances of each type.

The other mechanism we propose is an extension of the greedy mechanism proposed by Lehmann et al. [16]. This mechanism determines the allocation based on the valuation of the users and the total number of items they request. We extend the mechanism proposed by Lehmann et al. [16] so that it considers the relative sizes of the VM instances and show that the properties of the original mechanism are maintained. We compare the two proposed combinatorial auction-based mechanisms with the fixed-price based allocation mechanism used by Microsoft in their Windows Azure platform [18]. We investigate the relative performance of these three allocation mechanisms by performing extensive simulation experiments. We also consider variants of the fixed-price mechanism in which the fixed prices are different at different times of the day. We compare the performance of these mechanisms with that obtained by our proposed mechanisms as well. The experiments show that the proposed combinatorial auction-based mechanisms clearly outperform the fixed-price mechanism in terms of resource utilization, generated revenue, and allocation efficiency. We analyze the results and provide recommendations on where to use the proposed mechanisms.

1.2. Related work

The use of auctions in computing dates back to 1968 when Sutherland [24] proposed allocating the processor time in a single computer via auctions. Gagliano [13] also investigated the allocation of computing resources through auctions, where the tasks themselves are provided enough intelligence to calculate the bid that is necessary to get the required resources. Recently, researchers investigated various market-based models for resource allocation in computational grids. Wolski et al. [28] compared commodities markets and auctions in grids in terms of price stability and market equilibrium. Gomoluch and Schroeder [15] simulated a double auction protocol for resource allocation in grids and showed that it outperforms the conventional round-robin approach. Das and Grosu [11] proposed a combinatorial auction-based protocol for resource allocation in grids. They considered a model where different grid providers can provide different types of computing resources. An 'external auctioneer' collects this information about the resources and runs a combinatorial auction-based allocation mechanism where users participate by requesting bundles of resources. The major difference between the present work and the one presented in [11] is that we are considering allocating VM instances of a single cloud provider whereas Das and Grosu [11] considered the problem of allocating different types of physical resources from multiple grid providers. Garg et al. [14] designed a double auction-based meta-scheduler for grids, which schedules grid jobs into different clusters that improves both user utility and system performance when compared to traditional meta-schedulers. The differences between the market-based mechanisms designed for grids and those designed for clouds are mainly related to their underlying resource allocation model. Clouds allocate resources in terms of VM instances while traditional grids allocate physical resources directly without involving virtualization. The market-based mechanisms are more suitable for clouds since they are designed to make profit by selling services while traditional grids were designed mainly for sharing resources and not for making profits by selling resources.

Recently, researchers investigated the economic aspects of cloud computing from different points of view. Wang et al. [27] studied different economic and system implications of pricing resources in clouds. Altmann et al. [1] proposed a marketplace for resources where the allocation and pricing are determined using an exchange market of computing resources. In this exchange, the service providers and the users both express their ask and bid prices and matching pairs are granted the allocation. Risch et al. [21] proposed a testbed for cloud services designed for testing different mechanisms. They deployed the exchange mechanism proposed by Altmann et al. [1] on this platform.

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

An exchange requires that providers and users submit asks and respectively bids to the exchange mechanism which uses them to match users to providers. In this paper we are considering a combinatorial auction mechanism run by a single cloud provider instead of an exchange mechanism run by a federation of clouds.

The tool CloudCmp [17] was developed to assist users in choosing the appropriate service providers based on the user's requirements. Buyya et al. [8] proposed an infrastructure for auction-based resource allocation across multiple clouds. Researchers also investigated whether cloud solutions are economically feasible for all kinds of computing needs. Walker et al. [25] proposed a model to determine the benefits of acquiring storage services from clouds. Chohan et al. [9] showed how to accelerate MapReduce jobs using Spot Instances. They also analyzed the performance gain and the cost effectiveness of this approach. Ben-Yehuda et al. [7] analyzed the pricing of Amazon EC2 and claimed that it is not market-driven. They showed that the prices are randomly generated considering a hidden reserve price that is not driven by supply and demand.

The complexity of solving the combinatorial auctions, specifically the winner determination problem, was first addressed by Rothkopf et al. [22]. Sandholm [23] proved that solving the winner determination problem is computationally hard. Rothkopf et al. [22] and Sandholm [23] used the technique of pruning the search tree to devise approximation algorithms. Andersson et al. [4] proposed an integer programming based solution to the winner determination problem.

Zurel and Nisan [30] also presented an efficient algorithm for combinatorial auctions. Lehmann et al. [16] studied combinatorial auctions with single-minded bidders and devised a greedy mechanism for combinatorial auctions. In this paper, we extend this mechanism in order to solve the problem of VM allocation and pricing in clouds. Archer et al. [5] considered another case of single-minded bidders where multiple identical copies are available for different types of items. They provided a mixed integer programming based algorithm for winner determination and showed theoretically that their solution performs better than generalized solutions for this special case. We extend this mechanism such that it can be used to solve the VM allocation and pricing problem we consider in this paper. A detailed survey on combinatorial auctions can be found in [12]. Cramton et al. [10] provides good foundational knowledge on this topic.

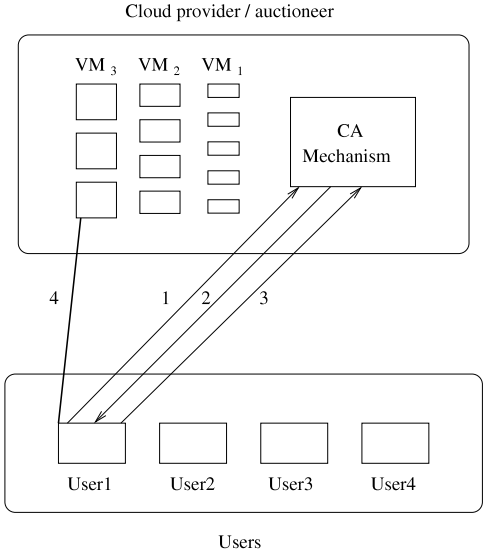
1.3. Organization

The rest of the paper is organized as follows. In Section 2, we formally define the VM instance allocation problem. In Section 3, we present the mechanisms we consider for solving the VM allocation problem. In Section 4, we describe the experimental results. In Section 5, we conclude the paper and present possible directions for future research.

2. Virtual machine allocation problem

The cloud providers set different configurations of VM instances that the users can request. A user requests VM instances of different types and pays the cloud provider for the time she uses them. Usually, the prices for different types of instances for short-term use are fixed by the cloud providers in advance. Another possibility is that a user sets up a long-term contract if she requires the resources for a long period of time, in which case she may obtain them for a lower price. Here we consider the problem of efficient allocation and pricing of VM instances for short-term use.

In Fig. 1, we provide a high-level representation of the VM instance allocation system we consider in this paper. The cloud provider has several VM instances of different types available for allocation and runs a combinatorial auction-based mechanism to allocate them to users. The auction mechanism consists of three steps. First, the mechanism collects 'bids' from the users, which comprise the number of different types of VM instances a user requests and the price she offers for that bundle. Then, the mechanism computes the allocation and the payment based on the collected bids and the availability of resources. Finally, users who get the allocation pay the cloud provider and obtain access to the resources they requested.



Step 1: Mechanism collects bids from all users
Step 2: Mechanism computes allocation and payment
Step 3: User pays the cloud provider
Step 4: User gets access to the resources requested

Fig. 1. VM instance allocation in clouds: system model.

We define the *Virtual Machine Allocation Problem (VMAP)* as follows. Assume that the allocation and prices are decided periodically by a given mechanism. Let the interval between two such decisions be 'one unit of time'. VMAP considers allocating the VM instances for one unit of time. Assume that a cloud provider has m different types of virtual machines $\mathrm{VM}_1, \ldots, \mathrm{VM}_m$. The relative computing capabilities (based on number and speed of CPUs, memory, etc.) of these VMs are characterized by a vector $\mathbf{w} = (w_1, \ldots, w_m)$, where $w_i \in \mathbb{R}_+$, $i = 1, \ldots, m$. We also assume that $w_1 = 1$ and $w_1 \leq w_2 \leq \cdots \leq w_m$. To illustrate this, we consider the types of instances currently offered by Microsoft Azure Platform: Small (CPU 1.6 GHz, Memory: 1.75 GB, Storage: 225 GB), Medium (CPU 2 $\times$ 1.6 GHz, Memory: 3.5 GB, Storage: 490 GB), Large (CPU 4 $\times$ 1.6 GHz, Memory: 7 GB, Storage: 1 TB), and Extra large (CPU 8 $\times$ 1.6 GHz, Memory: 14 GB, Storage: 2 TB). In this example, VM_1, VM_2, VM_3, VM_4 are the Small, Medium, Large, and respectively Extra large VM instances. The weight vector characterizing the VM instances is $\mathbf{w} = (1, 2, 4, 8)$.

Let us assume that k_i copies of VM_i instances are available for allocation at a given instance of time, $i = 1, \ldots, m$. There are n users $u_1, \ldots, u_n$, each requesting a set (bundle) of VM instances and revealing how much she values that particular set. That is, a user u_j is requesting VMs from the cloud provider by placing a bid $B_j = (r_1^j, r_2^j, \ldots, r_m^j, v_j)$, where $r_i^j \in \{0, 1, \ldots, k_i\}$ is the number of instances of type VM_i user u_j requires in her bundle and v_j is her valuation for this bundle, i.e., the maximum price she is willing to pay for using the requested VMs for one unit of time. Here we consider the users to be single-minded bidders. A single-minded bidder u_j desires only a specific bundle of items S_j, and values that

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

bundle at v_j. Thus, u_j has the following valuation function for a bundle S [16],

$$v(S) = \begin{cases} v_j & \text{if } S_j \subseteq S \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

We would like to mention that the assumption of single-minded bidders does not limit the users to express more flexible requirements. Our model assumes that auctions are run periodically and that bidders will request only one bundle in a given auction. Since the auctions are run periodically, a user may choose to revise her bid based on the previous auction outcome and her preference. For example, suppose that the time interval between consecutive auctions is one hour. If a user needs a particular bundle for five units of time and her deadline to complete the job is ten hours, she needs to win five auctions within ten hours. She may choose to bid the same value until her job is finished, or she may choose to start with a low bid and raise it when the deadline is approaching. Users executing parallel applications may want to request as many VM instances as possible to finish their jobs quickly. In this case, they could start by bidding for the largest possible bundle they can afford and if not successful, adjust the requested bundle size for the next auction. If a user must require continuous allocation of resources, she may continue bidding increasing values in order to increase her chances of winning every auction.

The goal of the VM allocation problem, given the set of users U and their bids, is to determine the set of winners $W \subseteq U$ and the price the winners have to pay to the cloud provider. User u_j is a winner (i.e., $u_j \in W$) if she receives her requested bundle of VM instances. The price user u_j pays to the cloud provider is denoted by p_j. We formally define the VM allocation problem as follows:

Virtual Machine Allocation Problem (VMAP)

Determine the set of winners, $W \subseteq U$, and payment p_j for each user $u_j, j = 1, \ldots, n$, such that

$$\sum_{j:u_j \in W} r_i^j \leq k_i \quad i = 1, \ldots, m \quad (2)$$

$$0 \leq p_j \leq v_j \quad \text{if } u_j \in W \quad (3)$$

$$p_j = 0 \quad \text{if } u_j \notin W. \quad (4)$$

The constraint in Eq. (2) ensures that the users are allocated at most k_i instances of VM_i. Eqs. (3) and (4) ensure that the winners pay at most their valuations and the losers do not pay at all.

Note that VMAP does not have an objective function. The most reasonable objective function would be to maximize the cloud provider's revenue, but very little is known about revenue maximization in the context of combinatorial auctions [20]. Combinatorial auctions are usually designed to maximize the sum of the bidders' valuations, i.e., $\max \sum_{j=1}^{n} v_j$. Since valuation is a measure of willingness to pay, maximizing the sum of the valuations usually generates more revenue for the resource provider than a fixed-price allocation does. On the other hand, given the prices of each type of VM instance, a fixed-price allocation mechanism does not have an objective function to maximize. Therefore, VMAP is formulated here as a feasibility problem with the constraints that are to be satisfied by all types of solutions. We shall introduce other constraints and/or objective functions when we discuss the proposed mechanisms for solving VMAP.

3. Virtual machine allocation mechanisms

In this section, we present three mechanisms that solve VMAP. The first, called FIXED-PRICE, is the fixed-price mechanism currently used by several cloud service providers [3,19]. The next two mechanisms are the proposed combinatorial auction-based mechanisms, CA-LP (Combinatorial Auction-Linear Programming) and CA-GREEDY (Combinatorial Auction-Greedy). CA-LP is an extended version of the mechanism proposed by Archer et al. [5]. The mechanism proposed by Archer et al. [5] solves a problem similar to VMAP by using linear programming relaxation and randomized rounding. We extend that mechanism so that it is able to solve VMAP. CA-GREEDY is an extension of the mechanism proposed by Lehmann et al. [16]. The mechanism proposed by Lehmann et al. [16] provides the best achievable approximate solution[1] for combinatorial auctions with single-minded bidders. However, this is a general purpose mechanism that does not assume any relative importance of the items being allocated. We extend this mechanism by incorporating the weights of different types of VMs as described in Section 2. We now describe each mechanism in detail.

Algorithm 1 FIXED-PRICE Mechanism

1: {*Phase 1: Receive requests from users*}
2: **for** $j = 1, \ldots, n$ **do**
3: Receive $(r_1^j, \ldots, r_m^j, v_j)$ from user u_j
4: **end for**
5: {*Phase 2: Allocation*}
6: Sort users according to their time of placing the request, from earliest to latest.
(Here we assume $u_1, u_2, \ldots, u_n$ as the order.)
7: Initialize $W \leftarrow \emptyset$
8: **for** $j = 1, \ldots, n$ **do**
9: $F_j \leftarrow \sum_{i=1}^{m} r_i^j f_i$
10: **if** $(v_j \geq F_j)$ **and**
11: $(r_i^j + \sum_{u_{j'} \in W} r_i^{j'} \leq k_i, i = 1, \ldots, m)$ **then**
12: $W \leftarrow W \cup \{u_j\}$
13: **end if**
14: **end for**
15: {*Phase 3: Payment*}
16: **if** $u_j \in W$ **then**
17: User u_j pays, $p_j = F_j$
18: **else**
19: User u_j pays, $p_j = 0$
20: **end if**

3.1. FIXED-PRICE mechanism

The FIXED-PRICE mechanism presented in Algorithm 1 defines a fixed-price vector $\mathbf{f} = \{f_1, \ldots, f_m\}$, where f_i is the price a user has to pay for using one instance of VM_i for one unit of time. The mechanism allocates VM instances to the users in a first-come, first-served basis until the resources are exhausted. It also makes sure that in order to get the requested bundle, the valuation of user u_j is at least F_j, where F_j is the sum of the fixed prices of each VM instance in her bundle (line 10). It also makes sure that the allocation does not exceed the number of available VM instances of each type (line 11). The set of users receiving the requested bundle is denoted by W. A user pays the sum of the fixed-prices of each VM instance in her allocated bundle.

3.2. Combinatorial auction-based mechanisms

The general combinatorial auction problem can be informally stated as determining the allocation and prices of bundle of items such that the sum of the user's valuations is maximized. In a combinatorial auction, user valuations are expressed on bundles of items rather than on individual items.

[1] Lehmann et al. [16] showed that the approximation ratio achieved by their proposed mechanism cannot be further improved unless NP = ZPP.

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

A desired property of a combinatorial auction mechanism is *truthfulness*. A mechanism is truthful if the participants benefit the most when they reveal their true valuations to the mechanism. A participant's benefit in a combinatorial auction is expressed by her *utility*, which is defined as the difference between the valuation she receives from the resource allocation and the price she pays to the mechanism. An ideal truthful mechanism determines the optimal allocation that maximizes the sum of the valuations and computes payments such that each participant maximizes her utility only by reporting her true valuation to the mechanism. A truthful mechanism helps the bidders in that they do not need to compute a complex strategy or assume other users' strategies while making their bids. They just need to bid their true valuations for the bundle since bidding any other value will not improve their utility.

The winner determination problem of combinatorial auctions is an NP-hard problem [23]. Therefore, research has been conducted to find approximate solutions to combinatorial auctions. In order to obtain a truthful approximation mechanism that solve the winner determination problem, few issues need to be addressed [6]. The approximation algorithm needs to be *monotone*. In a monotone allocation algorithm, a bidder can only increase her chance of getting her requested bundle by reporting a higher valuation or by requesting fewer items in her bundle. A monotone allocation algorithm allows finding the so called *critical value* of a winning bidder, which is the minimum she needs to bid in order to get her requested bundle. In a truthful mechanism a winning user has to pay her critical value to the mechanism. For some combinatorial auction problems, randomization is involved in the winner determination and/or the payment calculation algorithm. In that case, the goal of the resulting mechanism is to ensure that the participants maximize their expected utility by bidding their true values. Such mechanisms are *truthful in expectation*. We discuss the useful properties of our proposed mechanisms in the next subsections.

The proposed mechanisms are intended to be run periodically, each time considering the bids placed by the users during that period. It is assumed that users place their bids until they have been allocated their requested resources for enough units of time to execute their job to completion, or it becomes obvious that their job cannot be completed by a deadline. We also assume that the VM instances are statically provisioned, that is, the cloud provider has already provisioned a given number of VM instances of each type and only these instances are available for allocation.

3.2.1. CA-LP mechanism

Archer et al. [5] considered a combinatorial auction problem similar to VMAP. The difference is that in their case bidders can request at most one copy of each item type (i.e., $r_i^j \in \{0, 1\}$), whereas in the VMAP, users can request multiple copies of each type of item (i.e., $r_i^j \in \{0, 1, \ldots, k_i\}$). We modify the winner determination algorithm of the original mechanism such that it is able to solve VMAP. The algorithm for the calculation of payment is kept the same as in [5] because it maintains its properties when applied to VMAP with the modified winner determination algorithm. We present it here for completeness. The CA-LP mechanism is given in Algorithm 2.

Algorithm 2 CA-LP Mechanism

1: {*Phase 1: Collect Bids*}
2: **for** $j = 1, \ldots, n$ **do**
3: Collect bid $B_j = (r_1^j, \ldots, r_m^j, v_j)$ from user u_j
4: **end for**
5: {*Phase 2: Winner Determination*}
6: Set $k_i' \leftarrow (1 - \epsilon)k_i$, where $0 < \epsilon < 1, i = 1, \ldots, m$
7: Solve the following linear program

$$\max \sum_{j=1}^{n} x_j v_j \tag{5}$$

subject to

$$\sum_{j=1}^{n} x_j r_i^j \le k_i', \ i = 1, \ldots, m \tag{6}$$

$$0 \le x_j \le 1, \quad j = 1, \ldots, n \tag{7}$$

8: Initialize $W \leftarrow \emptyset$
9: **for** each user u_j, taken in descending order of x_j **do**
10: Generate a random number $y_j \in [0, 1]$
11: **if** $(y_j \le x_j)$ **and**
12: $(r_i^j + \sum_{j':u_{j'} \in W} r_i^{j'} \le k_i, i = 1, \ldots, m)$ **then**
13: $W \leftarrow W \cup \{u_j\}$
14: **end if**
15: **end for**
16: {*Phase 3: Payment* (same as in [4])}
17: **for** each user $u_j \in W$ **do**
18: Perform binary search for v_j' in the range $[0, v_j]$
(i) Set valuation of u_j as v_j' in Equation (5);
(ii) Solve the LP, let x_j' be the fractional allocation computed for u_j;
(iii) Until a v_j' is found such that, setting valuation of u_j less than v_j' generates $x_j' < y_j$ and setting the valuation greater than v_j' generates $x_j' > y_j$. This v_j' is the 'critical value'.
19: $p_j \leftarrow v_j'$
20: **end for**
21: **for** each user $u_j \notin W$ **do**
22: $p_j \leftarrow 0$
23: **end for**

CA-LP involves solving the linear program given by Eqs. (5)–(7). The objective of the linear program is to find a vector of 'fractional allocations' $\mathbf{x} = \{x_1, \ldots, x_m\}$ that maximizes the sum of the users' valuations (Eq. (5)). In line 6, the total number of available VM_i instances is reduced to k_i', which is then used in the constraint in Eq. (6). This constraint limits the allocation of VM_i instances to k_i'. Using k_i's instead of k_is in this constraint helps reducing the probability of over allocating the VMs during the randomized rounding performed in lines 9–15. This constraint is a modification of the constraint used in the mechanism presented in [5] by letting r_i^j take any value rather than only 0 and 1. The next constraint (Eq. (7)) bounds the fractional allocation values between 0 and 1.

Lines 9–15 implement the randomized rounding where user u_j is selected as a winner with probability x_j, if this allocation does not violate any constraint in Eq. (6). This operation is executed in order of decreasing x_j so that if there is a violation in the constraint, the user assigned a lower x_j is not included in the set of winners W. This step is another modification of the winner determination algorithm presented in [5]. In the original algorithm, users are first included in W with a probability of x_j and if constraint (6) is violated for any item, all users requesting that item are excluded from W. This method is suitable for auctions where many different types of items are sold and each type of item has only a few copies. But in the context of VMAP, this approach will significantly affect the allocation since each type of VM has many copies and there are only a few different types of VMs. For example, let a cloud provider offer four types of virtual machines, 500 instances of each type. Suppose that after rounding, VM_1 becomes over allocated. The mechanism proposed by Archer et al. [5] discards all the users that request any instance of VM_1 in her bundle. This results in 500 unsold VM instances. The other VMs requested by those users are also deallocated. This is the reason we cannot use the original winner determination algorithm proposed by Archer et al. [5] to solve VMAP.

The payment is calculated in lines 17–23. For each winning user u_j, CA-LP computes u_j's critical payment as follows. It performs a binary search in the range $[0, v_j]$, where v_j is the reported valuation of u_j. For each $v_j' \in [0, v_j]$, it solves the linear program given in

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

line 7 until it finds the minimum v_j' that yields $x_j \geq y_j$ (line 18). This v_j' is the critical value for user u_j because reporting a valuation less than v_j' will not allow her to win the bid, and therefore v_j' is what she has to pay. However, a losing user pays zero (lines 21 and 22). Note that the payment computation phase of CA-LP is the same as in the original mechanism.

We now summarize the changes we made to the original mechanism by Archer et al. in order to be able to solve VMAP. First, we relaxed the problem formulation to allow users to request multiple VM instances of the same type in their bundles. This is important in order to provide the user with more flexibility of bidding. The other modification is significant in terms of resource utilization. The original mechanism discards all bids that include a conflicting item. This approach is suitable only in the cases where the auction involves many item types where the number of each type of items is very small. The cloud providers usually offer only a few item types (VM instance types), and a large number of items of each type. Keeping the original approach would result in poor utilization of resources, and thus, we modified the allocation function to address this issue and at the same time maintain the truthfulness property.

Archer et al. [5] proved that the original mechanism is truthful in expectation. We claim that CA-LP maintains this property.

Theorem 1. *CA-LP mechanism is truthful in expectation.*

Proof. In order to prove that an approximation mechanism is truthful, we need to prove that its winner determination algorithm is monotone and that the payment calculated for a winning user is her 'critical payment', i.e., the minimum she needs to bid to obtain her requested bundle.

It is shown in [5] that the x_j values determined by solving the LP in line 7 are monotone with respect to the user valuations, i.e., a user u_j can increase her probability of winning by increasing her valuation. We now show that the randomized rounding step of CA-LP maintains the monotonicity of allocation. We can have two different cases in the randomized rounding step (lines 11–15),

$$\sum_{j:x_j>0} r_i^j \leq k_i, \quad \forall i \in \{1, \ldots, m\} \tag{8}$$

or

$$\sum_{j:x_j>0} r_i^j > k_i, \quad \exists i \in \{1, \ldots, m\}. \tag{9}$$

Eq. (8) represents the condition at which each user u_j having $y_j \leq x_j$ is guaranteed to get her requested bundle. Therefore, the probability of user u_j to be finally included in the set of winners is exactly x_j and the allocation is monotone.

On the other hand, when Eq. (9) holds for some i, we divide the users into two groups as follows. First, let us assume that $x_1, \ldots, x_n$ are in decreasing order. Now, let l be the largest index for which the following equation holds.

$$\sum_{j\in\{1\ldots l\},x_j>0} r_i^j \leq k_i, \quad \forall i \in \{1, \ldots, m\}. \tag{10}$$

Therefore, a user $u_j, j \leq l$, will be included in the winners list with probability x_j, which in turn is monotone with respect to her valuation.

Now, a user $u_j, l < j \leq n$, will get her allocation with probability x_j if

$$r_i^j + \sum_{j'<j,u_{j'}\in W} r_i^{j'} \leq k_i, \quad \forall i \in \{1, \ldots, m\} \tag{11}$$

Table 1
CA-LP example.

j	r_1^j	r_2^j	v_j	x_j	y_j	$u_j \in W$	p_j
1	0	4	0.74	0	0.43	*N*	0
2	3	4	7.62	0.85	0.32	*Y*	3.65
3	4	1	6.02	0.62	0.61	*N*	0
4	1	3	7.54	1	0.74	*Y*	2.01
5	2	1	5.94	1	0.14	*Y*	3.49
6	1	0	0.97	0	0.95	*N*	0

i.e., there are enough resources available to fulfill user u_j's request after determining the winners among $u_1, \ldots, u_{j-1}$. Therefore, the probability of user u_j winning her bundle is given by:

$$Pr\left[r_i^j + \sum_{j'<j,u_{j'}\in W} r_i^{j'} \leq k_i, \forall i \in \{1, \ldots, m\}\right] x_j. \tag{12}$$

The probability given by Eq. (12) decreases as j increases (i.e., x_j decreases). User u_j can increase her probability of winning by reporting a higher valuation. Therefore, the allocation is monotone with respect to her valuation, although it is not directly proportional to x_j.

Considering the above two cases, we claim that the allocation algorithm of CA-LP determines the set of winners with a probability that is monotone with respect to the user valuations.

The payment calculated by CA-LP is the critical value that is the minimum a user must bid to get her requested bundle allocated. Her reported valuation only helps decide whether she will be a winner, but she has to pay this critical value when she wins, no matter how large her valuation is. Because of these and following the results given in [5] the CA-LP mechanism is truthful in expectation. □

Example 1. We show the execution of CA-LP for a small VMAP instance illustrated in Table 1. In this VMAP instance, six users are placing their bids and the cloud provider has two types of VM instances with eight available copies for each type of instance. Each row of the table represents a user. The first four columns list the user index j, the requested number of VM instances of type 1 (r_1^j) and type 2 (r_2^j), and the user's valuation (v_j). For example, user u_1's bid is $B_1 = (0, 4, 0.74)$ specifying a request for zero instances of type VM_1 and four instances of type VM_2, and a valuation of 0.74 for this bundle. Column x_j shows the fractional allocation values for each user computed by the LP. The next column is the random value (y_j) used to decide the allocation. We see that users u_2, u_3, u_4, and u_5 have higher x_j than the corresponding y_js. But it is not possible to allocate the requested bundles to all these users, because that will exceed the number of available VMs of both types. Therefore, we first eliminate u_3 from the set of winners since x_3 is the minimum among these x_js. After this elimination, the set of winners satisfies all constraints. We show the final allocation decision in the column titled '$u_j \in W$', where '*Y*' means the bundle is allocated and '*N*' means the bundle is not allocated.

The values in the y_j column are also used in payment calculation. For example, the amount bidder u_4 will pay to the resource provider is determined by solving the LP with different valuations of u_4. Here, we perform a binary search between zero and 7.54 (i.e., v_4) to find out the valuation v_4' and solve the LP to find a new x_4', such that $x_4' < y_4$ (i.e., $x_4' < 0.74$) for valuations smaller than v_4' and $x_4' > y_4$ for valuations greater than v_4'. We find that for $v_4' = 2.0138$, $x_4' = 0.82 > 0.74$ and for $v_4' = 2.0129$, $x_4' = 0 < 0.74$. The search ends here by deciding the payment $p_4 = 2.0129$, which is shown rounded to two decimal digits in Table 1. We show the payment for all users in column p_j. Thus, users u_2, u_4 and u_5 obtain their requested bundles and pay 3.65, 2.01, and 3.49, respectively.

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

Algorithm 3 CA-GREEDY Mechanism

1: {*Phase 1: Collect Bids*}
2: **for** $j = 1, \ldots, n$ **do**
3: Collect bid $B_j = (r_1^j, \ldots, r_m^j, v_j)$ from user u_j
4: **end for**
5: {*Phase 2: Winner Determination*}
6: $W \leftarrow \emptyset$;
7: **for** $j = 1, \ldots, n$ **do**
8: $s_j \leftarrow \sum_{i=1}^{m} r_i^j w_i$
9: **end for**
10: re-order users such that $v_1/\sqrt{s_1} \geq v_2/\sqrt{s_2} \geq \ldots \geq v_n/\sqrt{s_n}$
11: **for** $j = 1, \ldots, n$ **do**
12: **if** for all $i = 1, \ldots, m$, $r_i^j + \sum_{u_{j'} \in W} r_i^{j'} \leq k_i$ **then**
13: $W \leftarrow W \cup u_j$
14: **end if**
15: **end for**
16: {*Phase 3: Payment*}
17: **for all** $u_j \in W$ **do**
18: $W_j' \leftarrow \{u_l : u_j \notin W \Rightarrow u_l \in W\}$
19: $l \leftarrow$ minimum index in W_j'
20: **if** $W_j' \neq \emptyset$ **then**
21: $p_j \leftarrow (v_l/\sqrt{s_l})\sqrt{s_j}$
22: **else**
23: $p_j \leftarrow 0$
24: **end if**
25: **end for**
26: **for all** $u_j \notin W$ **do**
27: $p_j \leftarrow 0$
28: **end for**

3.2.2. CA-GREEDY mechanism

Lehmann et al. [16] proposed a $\sqrt{M}$-approximation mechanism for combinatorial auctions with single-minded bidders where the total number of items that need to be allocated is M. We extend this mechanism by redefining M to be the weighted total number of VM instances, i.e., $M = \sum_{i=1}^{m} k_i w_i$. Here we define the 'size' s_j of the bundle in bid B_j requested by user u_j as $s_j = \sum_{i=1}^{m} w_i r_i^j$, while in the original mechanism, s_j is defined as $s_j = \sum_{i=1}^{m} r_i^j$, i.e., the total number of items requested in B_j. Our CA-GREEDY mechanism is given in Algorithm 3.

CA-GREEDY determines the winners by first ranking the users in decreasing order of their 'bid density' (i.e., $v_1/\sqrt{s_1}$) and then greedily allocating them starting from the top of the list. Before allocating a new bundle the mechanism verifies that the new allocation does not exceed the number of available VM instances of each type (lines 11–15). The payment p_j a winner u_j pays is calculated by multiplying $\sqrt{s_j}$ with the highest bid density among the losing bidders who would win if u_j would not be a winner (lines 17–24). That is, the winner pays the critical value.

Our mechanism differs from the mechanism proposed by Lehmann et al. [16] in the way the bid density is calculated. The original mechanism computes s_j as the total number of items in B_j, while in our case we consider s_j to be the weighted sum of the number of VM instances requested in B_j. Another difference is in the way our mechanism verifies if the capacity is exceeded for each type of VM instance (line 12). These two differences are significant because the original problem formulation assumes that each item is of different type and that different types of items do not have any relative importance to the auctioneer. In our setting a cloud provider allocates different types of VM instances, which have different characteristics and are valued differently by the cloud provider. Thus, we associate a weight to each VM instance type in order to reflect these differences. In line 12, the original mechanism needs to check whether there is a common item in the bundle of the user that is currently being allocated and the ones that are already in the set of winners. Since there are lots of VM instances of the same type, we changed this and check if the number of instances of each type allocated to the winning bidders does not exceed the number of available instances of each type. We claim that CA-GREEDY has the same approximation ratio as the original greedy mechanism and it is truthful as well.

Table 2
CA-GREEDY example.

j	r_1^j	r_2^j	v_j	s_j	$v_j/\sqrt{s_j}$	$u_j \in W$	p_j
1	0	4	0.74	8	0.26	*N*	0
2	3	4	7.62	11	2.3	*N*	0
3	4	1	6.02	6	2.46	*Y*	5.63
4	1	3	7.54	7	2.85	*Y*	0.69
5	2	1	5.94	4	2.97	*Y*	4.6
6	1	0	0.97	1	0.97	*Y*	0

Theorem 2. *CA-GREEDY is a truthful mechanism that computes a $\sqrt{M}$-approximate solution to VMAP, where $M = \sum_{i=1}^{m} k_i w_i$.*

Proof. The mechanism proposed by Lehmann et al. [16] is an $\sqrt{M}$-approximation mechanism that solves the general combinatorial auction problem, where M is the total number of items. In the case of VMAP, $M = \sum_{i=1}^{m} k_i w_i$. According to the definition of **w**, w_i is the number of VM_1 instances equivalent to one VM_i instance. Therefore, in VMAP, M is the total number of equivalent instances of VM_1 that are available. The mechanism proposed by Lehmann et al. [16] provides a $\sqrt{M}$-approximation solution when there are M items in total, therefore the CA-GREEDY mechanism also generates an $\sqrt{M}$-approximation solution to the VMAP.

Now, we show that the winner determination algorithm of CA-GREEDY is monotone and the payment calculated for a winner is the critical value. From line 10 of the mechanism, it is clear that a user can increase her chance of winning by increasing her bid. Also, a user can increase her chance to win by decreasing the weighted sum of the items. For example, a user requesting two small and two large VM instances will be higher in the order than a user requesting one small and three large instances for the same valuation, although the numbers of VMs requested are the same. Therefore, the winner determination algorithm of CA-GREEDY is monotone with respect to user bids considering the relative computing capacities of different types of VMs. Finally, a winning bidder u_j pays the minimum amount she has to bid to win her bundle, i.e., her critical value. This is done by finding the losing bidder u_l who would win if u_j would not participate in the auction. User u_j's minimum bid density has to be at least equal to the bid density of user u_l for winning her bundle. Therefore, her critical valuation is $(v_l/\sqrt{s_l})\sqrt{s_j}$, which is the payment calculated by CA-GREEDY. Thus, the CA-GREEDY mechanism has a monotone allocation algorithm and charges the winning bidders their critical payment. We conclude that CA-GREEDY is a truthful mechanism. □

Example 2. In Table 2, we show the allocation and payment computation obtained by CA-GREEDY for the same instance of VMAP we used in Example 1. Here we also assume that $w_1 = 1$ and $w_2 = 2$, i.e., one instance of VM_2 is two times more powerful than one instance of VM_1. There are eight available instances of each type of VM. In Table 2, the first four columns represent the user index, the number of VMs of each type in their bundle and their valuation for that bundle. The value in the column titled 's_j' is the weighted sum of the total number of VM instances in a bundle. The next column, titled '$v_j/\sqrt{s_j}$', gives the relative valuation of users with respect to the weighted bundle size, that is the 'bid density'.

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

To determine the set of winners, we include users in descending order of $v_j/\sqrt{s_j}$ in the set of winners unless the inclusion violates the constraint that only eight copies of each type of VM can be allocated. User u_5 is the first winner and we allocate two copies of type VM_1 and one copy of type VM_2 to her. The next user to get an allocation is u_4, thus three instances of type VM_1 and four instances of VM_2 are allocated so far. Next, u_3 is selected for allocation, raising the total VM allocation to seven for VM_1 and five for VM_2. We see that the next user in the order is u_2, but allocating u_2 requires three instances of type VM_1 and four of type VM_2, whereas there is only one instance of type VM_1 and three instances of type VM_2 remaining. Therefore, u_2 is not included in the set of winners. User u_6, is next in the order not violating the constraints, thus she is included in the set of winners. So far, eight instances of VM_1 and five instances of VM_2 are allocated, leaving only three VM_2 instances not allocated. The last user, u_1, cannot obtain her allocation since she requests four instances of VM_2.

We show the payment calculation for user u_4 as an example. If u_4 is not a winner, there will be one and six instances of VM_1 and VM_2 to be allocated. The first non-winning user with respect to the order is u_2, but the number of VM instances available is not enough to allocate u_2. But the other remaining user, u_1's request can be fulfilled when u_4 is not a winner. Therefore, u_4's payment is calculated by multiplying u_1's bid density value by $\sqrt{7}$. Since no such user could be found for u_6, u_6's payment is zero.

Here, we note that the total revenue generated by CA-LP is 9.15 and that generated by CA-GREEDY is 10.92. However, it is not guaranteed that CA-GREEDY will always generate higher revenue than CA-LP. The payment of the CA-LP mechanism depends on both the random variables y_j generated during the allocation phase and the competition among the bidders. On the other hand, the payment determined by the CA-GREEDY mechanism depends only on the competition among the bidders. In the example, we see that the highest four bid densities are between 2.3 and 2.97, where 2.3 is the bid density of user u_2, which is highest among the losing bids. Since this value is close to the winning bids, the winning bidders need to pay more to win their bundles. However, in a different scenario the CA-LP mechanism may generate higher revenues.

4. Experimental results

We perform simulation experiments with different instances of VMAP. We solve these problems by employing the three mechanisms presented above. We compare the results and discuss the applicability of these mechanisms under different scenarios.

4.1. Experimental setup

The simulation for one instance of VMAP runs for five simulation days. During each simulation, a maximum of $N = 100,000$ users are generated. Groups of users are created five times an hour, i.e., every twelve minutes. Therefore, an average of about 167 users are generated every twelve minutes. We add a deviation randomly chosen from $[-20\%, +20\%]$ to this number to determine the actual number of users generated at a particular time. We invoke all three mechanisms every hour, with all the users generated during that hour and all users from previous time slots that are still active. An active user is one whose task has not been finished or the task deadline has not been reached. Each mechanism computes the allocation and pricing for the next one hour time-frame and keeps track of the users' status separately. We would like to emphasize here that each run of the mechanisms computes the allocation and payment of a given user for only one time slot and not for all the time slots required for the user's task to complete execution. The user will need to participate in and win several auctions in order to complete her task.

Users are of three categories: type-1, type-2, and type-3. Type-1 users are the most demanding, type-3 the least, and type-2 users fall in between. User demands are characterized by four factors: number of requested VMs, valuation, duration for which the bundle is requested, and a deadline by which the task has to be finished. For example, type-1 users request more VMs than the other two types of users, request the VMs for longer periods of time, have the highest valuations, and have stricter deadlines than the others. Also, each category of users are generated at particular times of the day. A simulation day is divided into three periods: peak (8am–4pm), off-peak (4pm-midnight), and night (midnight-8am). Type-1 users are generated (and hence submit their bids) during the peak hours only. Type-2 users submit bids only during peak and off-peak hours while type-3 users submit bids at any time of the day. To compare with real life scenarios, we can roughly consider that type-1 users are the big corporations, type-2 are the large and medium businesses, and type-3 are the small businesses and individual users.

We assume that the cloud provider offers four types of VM instances: small, medium, large, and huge (VM_1, VM_2, VM_3, and VM_4). We set their relative weights to $\mathbf{w} = (1, 2, 4, 8)$ and their fixed prices to $\mathbf{f} = (0.12, 0.24, 0.48, 0.96)$. This corresponds to the fixed-price model used in Microsoft's Windows Azure Platform [18]. We call this vectors a linear price vector since $f_i = 0.12 \cdot w_i$, for $i = 1, \ldots, 4$. Each user u_j's bid is a 5-tuple $(r_1^j, r_2^j, r_3^j, r_4^j, v_j)$, where r_i^j is the number of requested instances of VM_i and v_j is her valuation. User u_j's task is characterized by the tuple (t_j, d_j), where t_j is the duration for which the resources are requested and d_j is the time by which u_j's job needs to be completed.

To generate user bids, first the type of the user is randomly chosen from the user distribution. Then, random numbers are generated from the ranges $[r^{\min}, r^{\max}]$, $[0, v^{\max}]$, and $[t^{\min}, t^{\max}]$ and assigned to r_i^j, v_j, and t_j, respectively. These values are then scaled with a factor associated with the category of user. For example, the scale factors for r_i^js are given by the vector $\boldsymbol{\rho}$. Therefore, after generating r_i^j values from the given range, they are multiplied by ρ_1 to determine the actual value when the user is of type-1. To illustrate this, suppose we generate some $r_i^j = 5$ for user u_j and $\boldsymbol{\rho} = (2, 1.5, 1)$. Now, the actual r_i^j value of users of type-1, type-2, and type-3 will be $5 \times 2 = 10$, $5 \times 1.5 \approx 8$, and 5, respectively. Similarly, the elements of vector $\boldsymbol{\lambda}$ give the scaling factors for valuation of different types of users. After generating a random number within the range $[0, v^{\max}]$, we multiply it with the entry in λ corresponding to the type of the generated user. Similarly, vectors $\boldsymbol{\tau}$ and $\boldsymbol{\delta}$ denote the factors for scaling the time required and the deadline. The deadline is determined by selecting a random number, scaling it, and then adding the result to t_j. We list all simulation parameters in Table 3.

To create different instances of VMAP, we vary the parameters that affect the user distribution, demand, and payment resulting from the allocation. Thus, we choose four different distributions of type-1, type-2, and type-3 users given by the following tuples: (10%, 40%, 50%), (20%, 30%, 50%), (20%, 40%, 40%), and (30%, 30%, 40%). We consider four values of $v^{\max}$, 1, 2, 5, and 10, which give four ranges of valuations (0–1), (0–2), (0–5), and (0–10). We also vary the number of available VM instances and the factors that distinguish bids of different types of users. Table 3 lists the parameters, their description, and the range of values they take. Combining all these values with each other, our simulation experiment simulated 768 different instances of VMAP.

In addition to the above set of experiments, we perform six sets of experiments with 768 VMAP instances each—by varying only one of the parameters listed above. We create two sets of such experiments by setting N, the maximum number of users

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

Table 3
Simulation parameters.

Parameter	Description	Value(s)
m	Types of VMs	4
$k_1, \ldots, k_m$	Available VMs of each type	500, 1000, 2000
$\mathbf{w}$	Relative weight of VMs	(1, 2, 4, 8)
$\mathbf{f}$	Fixed-price vector	(.12, .24, .48, .96) (.12, .22, .39, .70) (.12, .26, .58, 1.28)
$\boldsymbol{\phi}$	Fixed-price factor vector	(1, 1, 1) (3, 2, 1) (4, 2, 1)
N	Maximum number of users	100,000, 50,000, 10,000
n	Number of users in an auction	Varies
$r^{\min}$, $r^{\max}$	Min. & Max. VM instances of each type in a bundle	0, 5
$v^{\max}$	Maximum valuation	1, 2, 5, 10
$t^{\min}$, $t^{\max}$	Min. & Max. execution time	1, 10
$d^{\min}$, $d^{\max}$	Min. & Max. deadline	2, 10
$\boldsymbol{\pi}$	Distribution of users	(10%, 40%, 50%), (20%, 30%, 50%), (20%, 40%, 40%), (30%, 30%, 40%)
$\boldsymbol{\rho}$	Scale factor for bundle size	(2, 1.5, 1), (3, 2, 1)
$\boldsymbol{\lambda}$	Scale factor for valuation	(2, 1.5, 1), (3, 2, 1)
$\boldsymbol{\tau}$	Scale factor for execution time	(2, 1.5, 1), (3, 2, 1)
$\boldsymbol{\delta}$	Scale factor for deadline	(0.5, 0.67, 1), (0.33, 0.5, 1)

to 50,000 and 10,000, respectively. From these experiments we try to evaluate the VM allocation mechanisms for various degrees of user demands. In the next two sets of experiments, we set $N = 100,000$ and consider two different fixed-price vectors $\mathbf{f}$ as follows. A sublinear price vector with prices for instance VM_i given by $f_i = 0.12 \cdot (w_i)^{0.85}$, which corresponds approximately to $\mathbf{f} = (0.12, 0.22, 0.39, 0.70)$; and a superlinear price vector with prices for instance VM_i given by $f_i = 0.12 \cdot (w_i)^{1.15}$, which corresponds to $\mathbf{f} = (0.12, 0.26, 0.58, 1.28)$. Since the FIXED-PRICE mechanism heavily depends on the fixed prices of the VM instances, these experiments let us determine whether the fixed-price vector affect the performance of the proposed mechanisms.

Finally, we vary the fixed-price vectors during the peak and off-peak hours of the day to examine whether they can generate higher revenue by capturing the higher demands during these times. This is accomplished by introducing the *fixed-price factor vector* $\boldsymbol{\phi}$. This is a 3-vector containing factors that are used as multipliers for the fixed-price vector during different hours of the day. For example, $\boldsymbol{\phi} = (3, 2, 1)$ indicates the fixed prices of each type of VM instance will be multiplied by 3 during the peak hours, by 2 during the off-peak hours, and by 1 during night hours. If the fixed price-vector is $\mathbf{f} = (0.12, 0.22, 0.39, 0.70)$ then the prices for the four types of VM instances during peak hours are given by (0.36, 0.66, 1.17, 2.1). In the regular case, $\boldsymbol{\phi} = (1, 1, 1)$, that is, the prices for VM instances are the same for all periods of the day. In our experiments we use two price factor vectors (3, 2, 1) and (3, 2, 1), that is, we consider that during peak hours the prices are four, and respectively three times higher than during the night hours, while the prices during off-peak hours are two times higher than the prices during night hours. This will allow us to investigate the effect of taking into account the demand when establishing prices for the fixed-price mechanisms. Table 3 lists these price vectors.

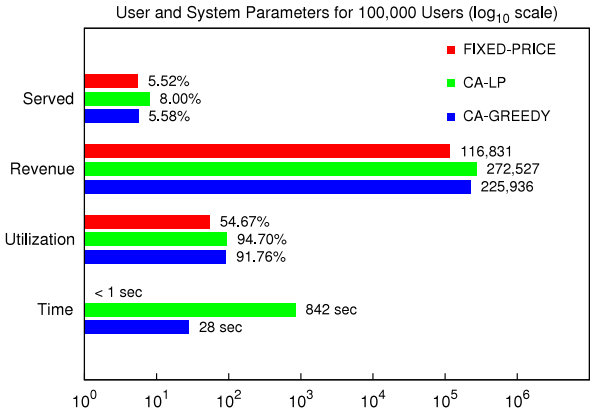


Fig. 2. Overall performance of the mechanisms with linear fixed-price vector (.12, .24, .48, .96), fixed-price factor vector $\boldsymbol{\phi} = (1, 1, 1)$, and 100,000 users. The plot is drawn at $\log_{10}$ scale.

4.2. Analysis of results

The experimental results show that the proposed combinatorial auction-based mechanisms have clear advantages over the fixed-price mechanism for solving the VMAP. Here we discuss their overall performance and then investigate the effect of different parameters on various performance metrics such as generated revenue, utilization, runtime, and the number of users served by the system.

First, we present the average performance of the mechanisms in Figs. 2–5. All the plots in these figures are represented using a logarithmic scale. The fixed price mechanism used in these experiments assumes the same fixed price for all the periods of the day, that is, the fixed price factor vector $\boldsymbol{\phi} = (1, 1, 1)$. In Fig. 2 we present the summary of the experiments with 100,000 users and the linear price vector. We see that CA-LP outperforms the other two mechanisms in all the metrics except the running time. Here the running time is the average time needed to run one auction simulation. About 8% of the 100,000 users could complete their tasks while running the CA-LP mechanism. We also see that the overall utilization of the resources and the revenue generated are the best for CA-LP. This is because the linear program has as objective maximizing the sum of the valuations, which eventually generates higher revenue by utilizing as many machines as possible while satisfying the constraint given in Eq. (6). Utilizing more machines allocates more users and therefore more users can finish their tasks. On the other hand, the CA-GREEDY mechanism allocates users based on their relative valuation. Therefore, it cannot always utilize resources as much as CA-LP can. But the running time of the CA-LP is prohibitively high because the payment calculation involves repeated solving of the linear program. The FIXED-PRICE mechanism obviously has the lowest running time because it only allocates users on a first-come, first-served basis. The CA-GREEDY mechanism has very low running time compared to CA-LP since its only major computation is to sort the list of users.

In Figs. 3 and 4, we show the summary of the results for experiments with 50,000 and 10,000 users and linear price vector. First, we observe that each of the mechanisms serves higher percentage of users, generates lower revenue, and utilizes less resources as the number of users decreases. This trend with decreasing demand is natural for any allocation mechanism. We further observe that the rank of the mechanisms in terms of all the metrics remain the same regardless of the total number of participants. Also note that compared to the FIXED-PRICE mechanism, the increase in served users is much higher for CA-LP and CA-GREEDY. This is due to the fact that FIXED-PRICE only

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

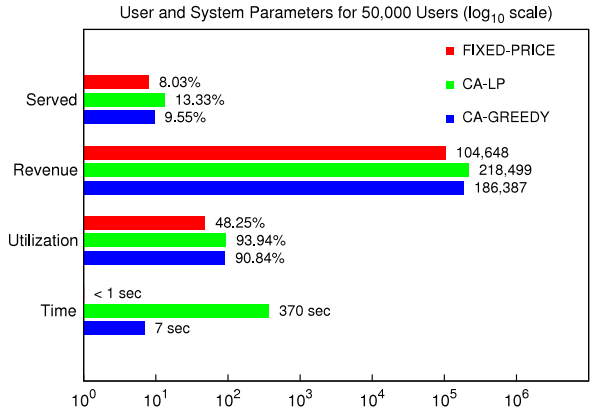


Fig. 3. Overall performance of the mechanisms with linear fixed-price vector (.12, .24, .48, .96), fixed-price factor vector $\boldsymbol{\phi} = (1, 1, 1)$, and 50,000 users. The plot is drawn at $\log_{10}$ scale.

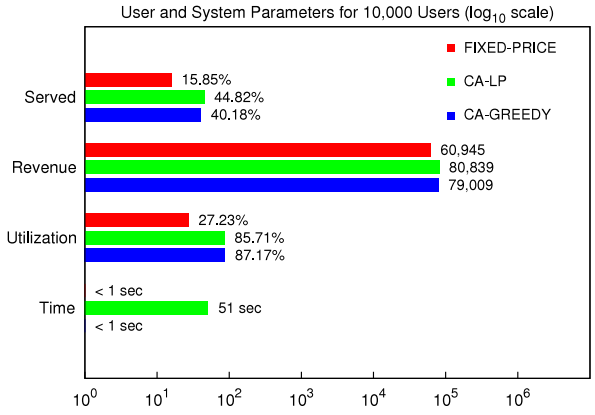


Fig. 4. Overall performance of the mechanisms with linear fixed-price vector (.12, .24, .48, .96), fixed-price factor vector $\boldsymbol{\phi} = (1, 1, 1)$, and 10,000 users. The plot is drawn at $\log_{10}$ scale.

considers those users who bid at least the fixed value, while the auctions determine allocations based on the market demand and supply. For the same reason, the utilization of the machines decreases at a slower rate in the case of combinatorial auction-based mechanisms than in the case of the fixed-price mechanism. However, the gap between the total revenue generated reduces when there are less participants, as the auction-based mechanisms generate less revenue when there is less competition.

In Fig. 5a and b, we summarize the results of the experiments with 100,000 users and sublinear and superlinear fixed-price vectors, respectively. By comparing them with the results in Fig. 2, we see that the only mechanism affected is FIXED-PRICE, which can serve more users and utilize more resources when the price vector is sublinear. However, in this case the total revenue decreases as users pay less than what they pay in the case of a linear price vector. Naturally, we see the opposite trend for the superlinear price vector. We can conclude that we cannot improve the overall quality of the allocation generated by the FIXED-PRICE mechanism. By changing the price vector we can only improve one metric while sacrificing another.

We now investigate different performance metrics by varying other simulation parameters, while setting the total number of users to 100,000 and using the linear price vector. In Fig. 6a, we show the revenue generated for different ranges of user valuations. We see that low user valuations most adversely affect the FIXED-PRICE mechanism. This is because it does not allocate the requested bundles to users having valuations below the fixed-price range. On the other hand the combinatorial auction mechanisms can generate higher revenues because they determine the payments from the user valuations. The revenue increases at the same rate from valuation ranges (0–1) to (0–5). Then, for the valuation range (0–10), we see a sharp rise in revenue generated by the auction mechanisms, while the FIXED-PRICE mechanism's revenue does not increase that much. This is because the price for an average-sized bundle is 4.5 according to the fixed prices we set. FIXED-PRICE mechanism's revenue is bounded by the fixed prices, therefore it cannot take advantage of higher user valuations. As shown in Fig. 6b, our experiments reveal that the rate of resource utilization obtained by the auction-based mechanisms is not affected by the valuation ranges. The utilization obtained by the FIXED-PRICE mechanism increases as the valuation range increases, and it is lower than that obtained by combinatorial auction mechanisms for all the ranges of valuations except for the (1–10) range.

In Fig. 7, we show the average revenue and resource utilization generated by the mechanisms when different values of the scale factors for valuation ($\boldsymbol{\lambda}$) and deadline ($\boldsymbol{\delta}$) are used. As a reminder, a scale factor for valuation (or, the price factor) represents how much more a bundle is valued by a types 1 and 2 user than a type 3 user. For example, $\boldsymbol{\lambda} = (2, 1.5, 1)$ denotes the case where on average a type-1 user bids twice the value than a type-3 user and a type-2 user bids around 1.5 times higher than a type-3 user. When $\boldsymbol{\lambda} = (3, 2, 1)$, these multiplication factors become 3 and 2, respectively and meaning that those users' demands are even higher than those of type-3 users. Similarly, a deadline factor says how strict is the deadline of type 1 and type 2 users compared to that of type 3 users. We consider four possible combinations of these two factors, which we denote as Ratio 1, . . ., Ratio 4 in Fig. 7. We show the

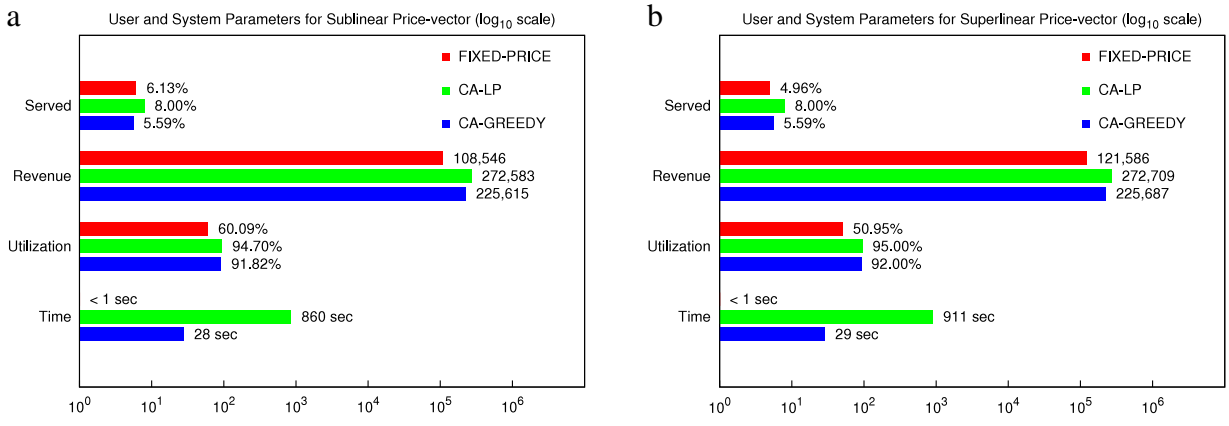


Fig. 5. Overall performance of the mechanisms with 100,000 users and (a) sublinear fixed-price vector (.12, .22, .39, .70); (b) superlinear fixed-price vector (.12, .26, .58, 1.28). The fixed-price factor vector $\boldsymbol{\phi} = (1, 1, 1)$. The plot is drawn at $\log_{10}$ scale.

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

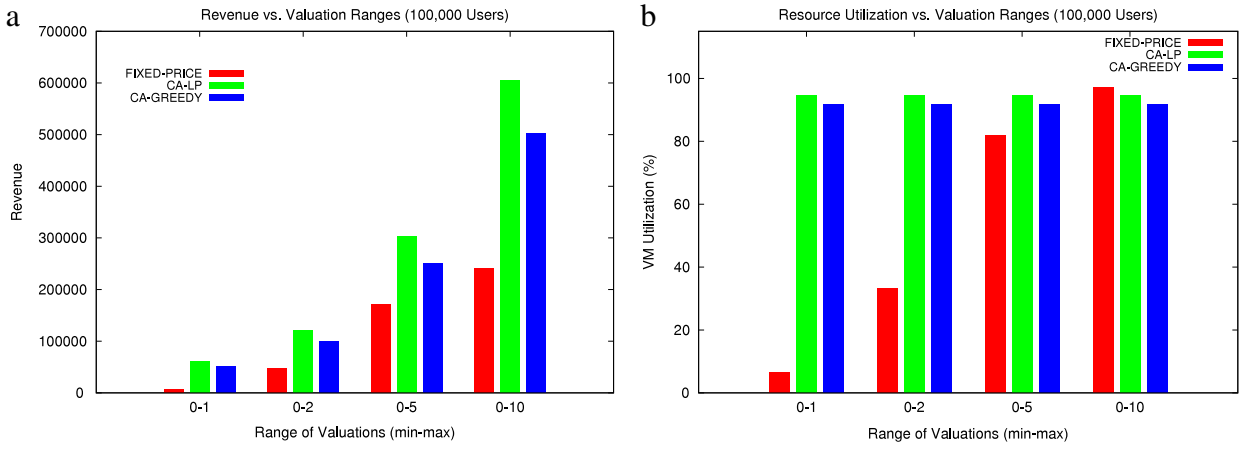


Fig. 6. Effect of valuation ranges (with 100,000 users) on (a) revenue; (b) VM utilization.

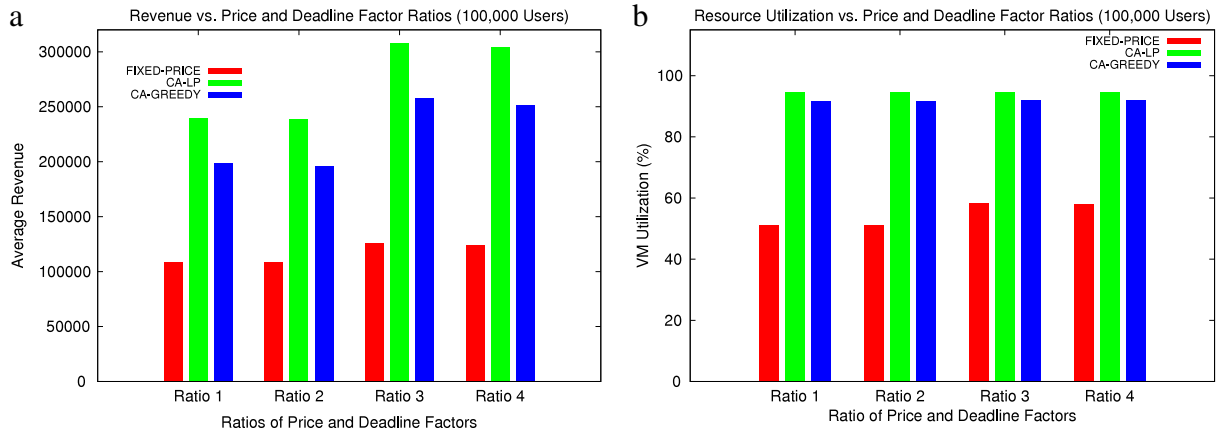


Fig. 7. (a) Revenue and (b) VM utilization vs. ratios of price and deadline factors. Ratio is defined as a set of ((price-factor), (deadline-factor)) values. Ratio 1 = ((2, 1.5, 1), (.33, .5, 1)), Ratio 2 = ((2, 1.5, 1), (.5, .67, 1)), Ratio 3 = ((3, 2, 1), (.33, .5, 1)), Ratio 4 = ((3, 2, 1), (.5, .67, 1)).

revenue generated in different such scenarios in Fig. 7a. Ratios 1 and 2 are for the price factors (2, 1.5, 1) and Ratios 3 and 4 are for the price factors (3, 2, 1). We see that the combinatorial auction-based mechanisms are capable of generating higher revenues when the types 1 and 2 bidders bid more, but the fixed-price mechanism cannot increase the generated revenue that much. However, we see that deadline factors do not have much effect on the outcome, as evident from similar values shown for different deadline factors but the same valuation factor (e.g., Ratios 1 and 2). From Fig. 7b we see that these factors have almost no effect on machine utilization achieved by the combinatorial auctions. But utilization is increased a little for the FIXED-PRICE mechanism with higher valuation factors.

We now examine how the three mechanisms deal with different types of users. Recall that type-1 users are the most demanding and type-3 users are the least demanding. First, we show the percentage of users who could complete their tasks in Figs. 8–10. We refer to these users as the served users. In these figures we show the results from three different sets of experiments, with 100,000, 50,000, and 10,000 users and the linear price vector. In Fig. 8, where the total number of users is 100,000, we observe that the FIXED-PRICE mechanism serves type-3 users the most. This is because it only considers the order in which users arrive. Type-1 and type-2 users have shorter deadlines and therefore leave the system if they do not get the allocation within a few allocation events. On the other hand, type-3 users have longer deadlines, and therefore they are active longer and eventually get the allocation once the users that entered the system earlier finish their tasks.

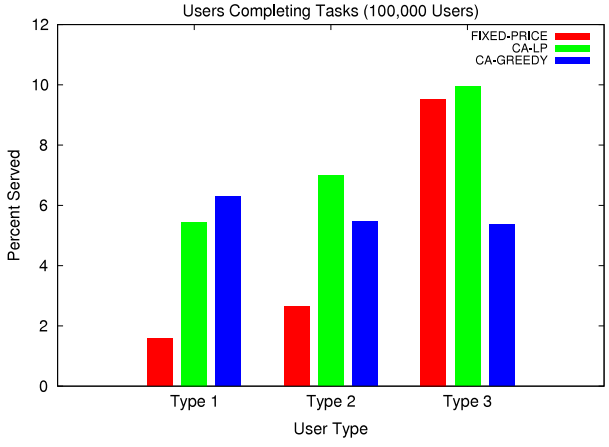


Fig. 8. Percentage of served users for simulations with 100,000 users.

CA-LP also served more users of type-3 than users of other types, yet it served more users compared to the FIXED-PRICE mechanism in every category. Here we see a nice property of CA-GREEDY that is, it serves more type-1 users than the other mechanisms. It also serves more users of type-1 than other user types. This is because CA-GREEDY makes decisions based on the bid densities, which are on average the highest for type-1 users.

Now comparing Fig. 8 with Figs. 9 and 10, we see that for 50,000 users the CA-GREEDY mechanism maintains its feature of serving a higher percentage of more demanding users than less demanding

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

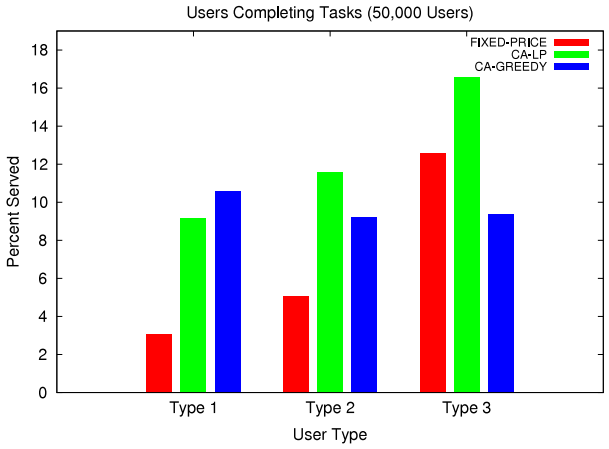


Fig. 9. Percentage of served users for simulations with 50,000 users.

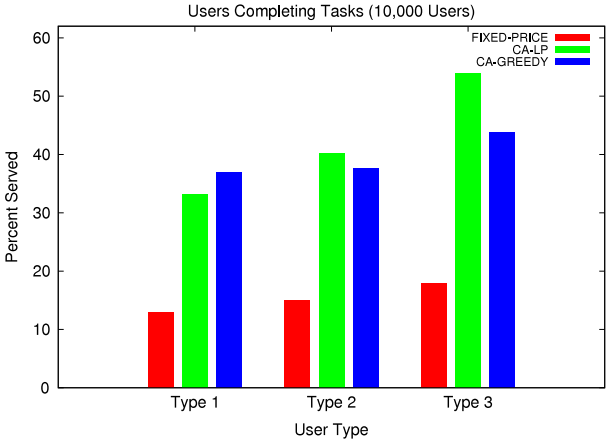


Fig. 10. Percentage of served users for simulations with 10,000 users.

users. But for 10,000 users, the percentage of served users of type 3 is higher than the ones corresponding to the other two types of users. This is because for the reduced demand, type 1 and type 2 users cannot occupy most of the resources as they do for the cases with higher number of users. Also, recall that type 1 and type 2 users request larger bundles and are active during peak hours and off-peak hours only. On the other hand, type 3 users are generated any time of the day. Therefore, the type 3 users get more space for occupying the resources facing less competition from the other users. Also, their bundle size is smaller compared to the other users and therefore a higher number of users can be served using the same amount of resources. On the other hand, in the case of CA-LP, we see almost the same trend (although at a different scale) in terms of serving the three types of users. Hence, we can conclude that CA-GREEDY is a better choice in terms of fairness and the handling of demand and supply in the market.

In Figs. 11–13 we plot the percentage of users that could only partially complete their tasks. First, we observe that the FIXED-PRICE mechanism has the least number of partially served users in all three cases. This is because of its inherent first-come, first-served policy. These plots also reveal that the percentage of partially served users increases in the case of CA-LP and CA-GREEDY when fewer users participate. Such effect is natural to auction mechanisms; a user must be denied the resources once another user with a higher bid arrives in an already saturated market for resources. However, if the consequences of having jobs partially completed is very important in some systems, it may be better to consider FIXED-PRICE as the allocation mechanism. The combinatorial auction-based mechanisms we propose can also be

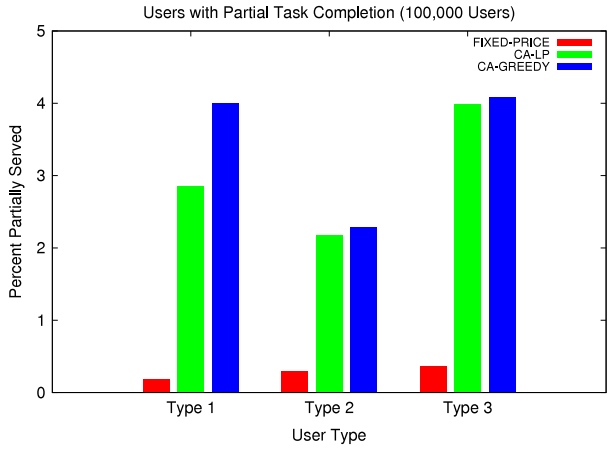


Fig. 11. Percentage of partially served users for simulations with 100,000 users.

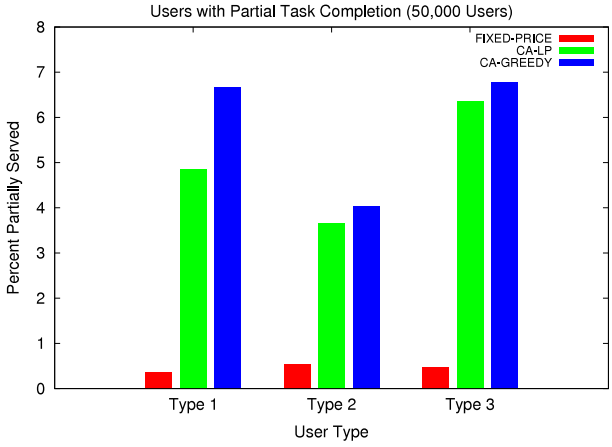


Fig. 12. Percentage of partially served users for simulations with 50,000 users.

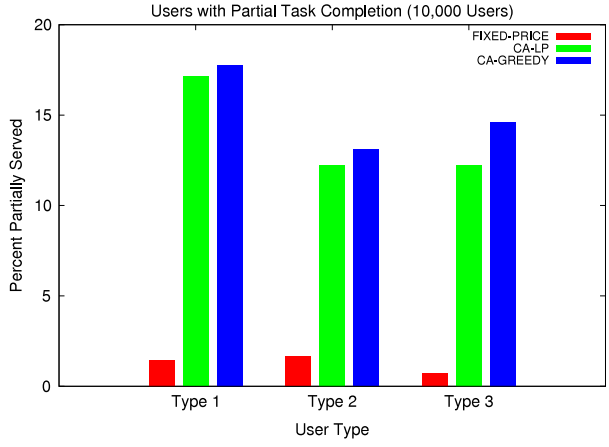


Fig. 13. Percentage of partially served users for simulations with 10,000 users.

improved by incorporating some penalty for partially finished jobs. In the simulations we consider that the bids are generated once and a user submits the same bid until she gets her job done or her deadline is reached. In practice, a user is an interactive entity and can adapt her bid depending on the value and urgency of her job and the current market demand. Creating an automated bidding agent to participate in the combinatorial auctions could also be an interesting research direction that could eventually decrease the number of partially served users.

Now we present the average resource utilization obtained by the three mechanisms during different periods of time of the day.

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

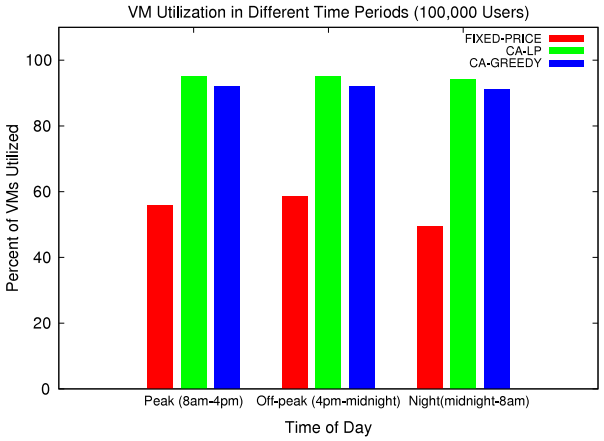


Fig. 14. Utilization of resources during different periods of time (100,000 users).

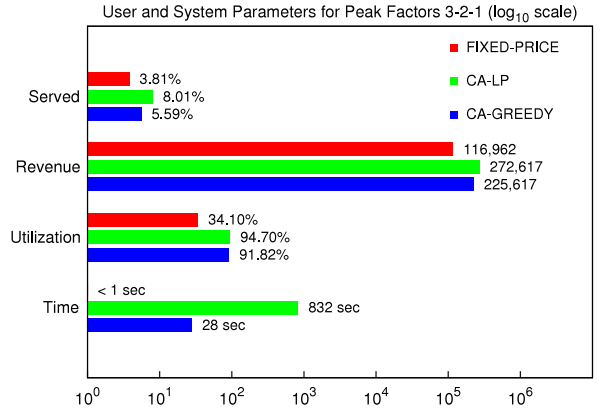


Fig. 15. Overall performance of the mechanisms with fixed-price factor vector $\boldsymbol{\phi} = (3, 2, 1)$ and 100,000 users. The plot is drawn at $\log_{10}$ scale.

Recall that in the experiments, we divided a day between peak (8am–4pm), off-peak (4pm-midnight), and night (midnight-8am) hours. Also, Type-1 users are generated during the peak hours, Type-2 users during the peak and off-peak hours, and Type-3 users can place their bids any time of the day. In Fig. 14, we see that the resource utilization is around 95% for CA-LP for each period of the day. The utilization achieved by CA-GREEDY is very close to that of CA-LP for all three periods. The proposed mechanisms are able to effectively balance the load of the system over time. The utilization obtained by FIXED-PRICE is about 56% and 59% during peak and off-peak hours and it falls below 50% during night. Since FIXED-PRICE is a first-come-first-served mechanism, it cannot free up resources that are being used by Type-3 users when in the morning Type-1 users start placing their requests. Since Type-1 users have shorter deadlines, by the time some resources are freed up, some users have already left the system. Therefore, the utilization obtained by FIXED-PRICE is far below that obtained by CA-LP and CA-GREEDY. At night, the utilization further drops because only the Type-3 users request computing resources.

All the above experiments considered the FIXED-PRICE mechanism with a fixed-price factor vector $\boldsymbol{\phi} = (1, 1, 1)$. We now show the results obtained by considering different fixed-price factor vectors, $\boldsymbol{\phi}$. This is equivalent to considering different prices at different times of the day and it will allow us to investigate the effect of increasing the prices during high demand hours on the performance of the mechanisms. The combinatorial auction-based mechanisms dynamically determine the prices of the VM instances. Since demand varies during peak, off-peak, and night hours, we multiply the fixed prices with different values based on the time of the day. The results presented in Fig. 15, are obtained when we multiply

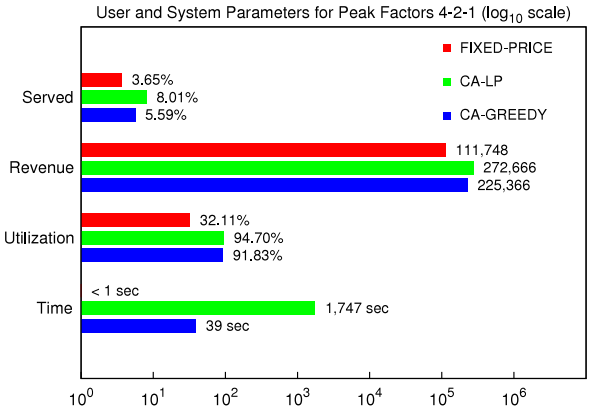


Fig. 16. Overall performance of the mechanisms with fixed-price factor vector $\boldsymbol{\phi} = (4, 2, 1)$ and 100,000 users. The plot is drawn at $\log_{10}$ scale.

the fixed-price vector by 3 for the peak hours, by 2 for the off-peak hours, and by 1 for the night hours. This correspond to a fixed-price factor vector $\boldsymbol{\phi} = (3, 2, 1)$. We see that when compared to the results presented in Fig. 2, the percentage of served users when employing the FIXED-PRICE mechanism has decreased from 5.5% to 3.8% and the utilization of VM instances has decreased from 54% to 34%. This is expected, since more users are being rejected allocation during peak and off-peak hours due to the increase in the fixed-prices. However, the revenue generated by the FIXED-PRICE mechanism remains at almost the same level. This shows that adjusting the fixed-price vector in anticipation of higher demand may not improve much the overall efficiency. At higher prices, fewer users are served and resources are under-utilized leading to no significant impact on the generated revenue. Fig. 16 shows the results obtained when we changed the fixed-price factor vector to $\boldsymbol{\phi} = (4, 2, 1)$, i.e., we multiply the fixed-price vector by 4 during the peak hours, by 2 during the off-peak hours and by 1 during night hours. Here we observe that while serving fewer users and utilizing less resources, the FIXED-PRICE mechanism also obtains lower revenue. We conclude that it is possible to control the behavior of fixed-price mechanisms by updating the fixed-price vector based on observation or statistical data about the demands. But combinatorial auction-based mechanisms compute the price dynamically, therefore no matter how the demand changes, they are able to obtain an efficient allocation and pricing.

In summary, we can conclude that combinatorial auction-based allocation and pricing mechanisms are more desirable over the fixed-price based ones currently employed by cloud providers. CA-LP is a better choice when the objective is to obtain higher revenue and higher utilization of resources. However, we have to limit the application of CA-LP to systems with small number of users, because otherwise the execution time will be prohibitive. This is because the CA-LP involves solving a linear program whose number of unknowns increases with the number of users participating in the auction. In addition to this, in order to compute the user's payments, CA-LP needs to solve one linear program for each user, thus the execution time increases very fast with the number of users. CA-LP can be a good choice when auctions are run at longer intervals. On the other hand, CA-GREEDY can be applied to cloud systems with any number of users being able to generate high revenue and resource utilization with very low execution time. CA-GREEDY is a better choice when the objective of VMAP is to maximize the social welfare. It is also worth mentioning that the CA-LP mechanism is designed for bidders with known bundles (i.e., bundles that are known to the auctioneer) [5]. Therefore, this mechanism is vulnerable to manipulation by users who bid for unknown bundles in the hope of obtaining a better allocation or price. On the other hand, CA-GREEDY is a truthful mechanism with

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006

respect to both valuations and the bundles requested. Considering all the above aspects, we recommend using the CA-GREEDY mechanism for solving general purpose VM allocation problems in clouds.

5. Conclusion

We investigated the applicability of combinatorial auction-based mechanisms for allocation and pricing of VM instances in cloud computing platforms. We proposed two combinatorial auction-based mechanisms for solving the problem of allocating VM instances in clouds. We compare their performance with that obtained by a currently used fixed-price mechanism. We perform extensive simulation experiments and conclude that combinatorial auction-based mechanisms are clearly a better choice for VM allocation in clouds. Based on experimental data and on the theoretical properties of the mechanisms, we also make recommendations that the CA-GREEDY mechanism should be the choice for general purpose VM instance allocation problems while the CA-LP mechanism can be reserved for special scenarios. Future work includes the deployment of the proposed mechanisms on an experimental cloud computing testbed and the development of other market-based VM allocation protocols.

Acknowledgments

This paper is a revised and extended version of [29] presented at the 2nd IEEE International Conference on Cloud Computing Technology and Science (IEEE CloudCom 2010). The authors wish to express their thanks to the editor and the anonymous referees for their helpful and constructive suggestions, which considerably improved the quality of the paper. This work was supported in part by NSF grants DGE-0654014 and CNS-1116787.

References

[1] J. Altmann, C. Courcoubetis, G.D. Stamoulis, M. Dramitinos, T. Rayna, M. Risch, C. Bannink, GridEcon: a market place for computing resources, in: Proc. Workshop on Grid Economics and Business Models, 2008, pp. 185–196.
[2] Amazon, Amazon EC2 spot instances, URL http://aws.amazon.com/ec2/spot-instances/.
[3] Amazon, Amazon EC2 pricing, URL http://aws.amazon.com/ec2/pricing/.
[4] A. Andersson, M. Tenhunen, F. Ygge, Integer programming for combinatorial auction winner determination, in: Proc. Fourth Int'l Conf. on Multi-Agent Systems, 2000, pp. 39–46.
[5] A. Archer, C. Papadimitriou, K. Talwar, E. Tardos, An approximate truthful mechanism for combinatorial auctions with single parameter agents, Internet Mathematics 1 (2) (2005) 129–150.
[6] A. Archer, E. Tardos, Truthful mechanisms for one-parameter agents, in: Proc. 42nd IEEE Symp. on Foundations of Computer Science, 2001, pp. 482–491.
[7] O.A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, D. Tsafrir, Deconstructing amazon EC2 spot instance pricing, in: Proc. 3rd IEEE Int'l Conf. on Cloud Computing Technology and Science, CloudCom 2011, 2011.
[8] R. Buyya, R. Ranjan, R.N. Calheiros, InterCloud: utility-oriented federation of cloud computing environments for scaling of application services, in: Proc. 10th Int'l Conf. on Algorithms and Architectures for Parallel Processing, 2010, pp. 13–31.
[9] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, C. Krintz, See spot run: using spot instances for MapReduce workflows, in: Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, 2010.
[10] P. Cramton, Y. Shoham, R. Steinberg, Combinatorial Auctions, The MIT Press, 2005.
[11] A. Das, D. Grosu, Combinatorial auction-based protocols for resource allocation in grids, in: Proc. 19th Int'l Parallel and Distributed Processing Symp., 6th Workshop on Parallel and Distributed Scientific and Engineering Computing, 2005.
[12] S. de Vries, R.V. Vohra, Combinatorial auctions: a survey, INFORMS Journal on Computing 15 (3) (2003) 284–309.
[13] R.A. Gagliano, M.D. Fraser, M.E. Schaefer, Auction allocation of computing resources, Communications of the ACM 38 (6) (1995) 88–102.
[14] S.K. Garg, S. Venugopal, J. Broberg, R. Buyya, Double auction-inspired meta-scheduling of parallel applications on global grids, Journal of Parallel and Distributed Computing (2012) (submitted for publication).
[15] J. Gomoluch, M. Schroeder, Market-based resource allocation for grid computing: a model and simulation, in: Proc. 1st Int'l Workshop on Middleware for Grid Computing, 2003, pp. 211–218.
[16] D. Lehmann, L.I. O'Callaghan, Y. Shoham, Truth revelation in approximately efficient combinatorial auctions, Journal of the ACM 49 (5) (2002) 577–602.
[17] A. Li, X. Yang, S. Kandula, M. Zhang, CloudCmp: shopping for a cloud made easy, in: Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, 2010.
[18] Microsoft, Windows Azure FAQ. URL http://www.microsoft.com/windowsazure/faq/.
[19] Microsoft, Windows Azure offers. URL http://www.microsoft.com/windowsazure/offers/.
[20] N. Nisan, T. Roughgarden, E. Tardos, V.V. Vazirani, Algorithmic Game Theory, Cambridge University Press, 2007.
[21] M. Risch, J. Altmann, L. Guo, A. Fleming, 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.
[22] M.H. Rothkopf, A. Pekec, R.M. Harstad, Computationally manageable combinatorial auctions, Management Science 44 (8) (1998) 1131–1147.
[23] T. Sandholm, Algorithm for optimal winner determination in combinatorial auctions, Artificial Intelligence 135 (1–2) (2002) 1–54.
[24] I.E. Sutherland, A futures market in computing, Communications of the ACM 11 (6) (1986) 449–451.
[25] E. Walker, W. Brisken, J. Romney, To lease or not to lease from storage clouds, IEEE Computer 43 (4) (2010) 44–50.
[26] R. Wang, Auctions versus posted-price selling, The American Economic Review 83 (4) (1993) 838–851.
[27] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, L. Zhou, Distributed systems meet economics: Pricing in the cloud, in: Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, 2010.
[28] R. Wolski, J.S. Plank, J. Brevik, T. Bryan, Analyzing market-based resource allocation strategies for the computational grid, The International Journal of High Performance Computing Applications 15 (3) (2001) 258–281.
[29] S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, in: Proc. 2nd IEEE Int'l Conf. on Cloud Computing Technology and Science, CloudCom 2010, 2010, pp. 127–134.
[30] E. Zurel, N. Nisan, An efficient approximate allocation algorithm for combinatorial auctions, in: Proc. 3rd ACM Conf. on Electronic Commerce, 2001, pp. 125–136.

Sharrukh Zaman received his Bachelor of Computer Science and Engineering degree from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh. He is currently a Ph.D. candidate in the Department of Computer Science, Wayne State University, Detroit, Michigan. His research interests include cloud computing, distributed systems, game theory and mechanism design. He is a student member of the IEEE.

Daniel Grosu received his Diploma in engineering (automatic control and industrial informatics) from the Technical University of Iasi, Romania, in 1994 and his M.Sc. and Ph.D. degrees in computer science from the University of Texas at San Antonio in 2002 and 2003, respectively. Currently, he is an associate professor in the Department of Computer Science, Wayne State University, Detroit. His research interests include distributed systems and algorithms, resource allocation, computer security and topics at the border of computer science, game theory and economics. He has published more than 70 peer-reviewed papers in the above areas. He has served on the program and steering committees of several international meetings in parallel and distributed computing. He is a senior member of the ACM, the IEEE and the IEEE Computer Society.

Please cite this article in press as: S. Zaman, D. Grosu, Combinatorial auction-based allocation of virtual machine instances in clouds, J. Parallel Distrib. Comput. (2013), doi:10.1016/j.jpdc.2012.12.006