2011 Third IEEE International Conference on Coud Computing Technology and Science

Combinatorial Auction-Based Dynamic VM Provisioning and Allocation in Clouds

Sharrukh Zaman
Department of Computer Science
Wayne State University
Detroit, MI 48202
Email: sharrukh@wayne.edu

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

***Abstract*—Efficient Virtual Machine (VM) provisioning and allocation allows the cloud providers to effectively utilize their available resources and obtain higher profits. Existing combinatorial auction-based mechanisms assume that the VM instances are already provisioned, that is they assume static VM provisioning. A better solution would be to take into account the users' demand when provisioning VM instances. We design an auction-based mechanism for dynamic VM provisioning and allocation that takes into account the user demand for VMs when making VM provisioning decisions. We perform extensive simulation experiments using real workload traces and show that the proposed mechanism can improve the utilization, increase the efficiency of allocation, and yield higher revenue for the cloud provider.**

***Index Terms*—cloud computing; VM provisioning; combinatorial auctions;**

I. Introduction

Cloud computing systems provide the next computing infrastructure enabling users to provision remote resources for their computational needs, eliminating the upfront costs of setting up their own systems. Clouds give users the illusion of an infinite computing resource available on demand and allow them to acquire and pay for resources on a short term basis. Examples of cloud computing systems include both commercial (e.g., Microsoft Azure [1], Amazon EC2 [2]) and open source ones (e.g., Eucalyptus [3]). The usage model of cloud computing involves virtualization of computing resources. The cloud providers provision their resources into different types of virtual machine (VM) instances. These instances are then 'sold' to the users for specific periods of time.

The current allocation mechanisms used by the cloud providers are fixed-price mechanisms. It is evident from economics literature that a fixed-price mechanism cannot ensure efficient allocation of resources [4]. In particular, since each user pays a fixed price for an item, such mechanisms cannot guarantee that the user who values an item the most gets it. An auction-based mechanism can achieve the economic efficiency because it allocates items based on the perceived values of the users. The nature of allocation requests for cloud resources suggests that a combinatorial auction-based mechanism is best suited for the VM allocation problem in clouds. However, we have to overcome certain challenges while using combinatorial auction-based mechanisms for VM provisioning and allocation in clouds. The winner determination problem of a combinatorial auction is NP-complete [5], therefore we need an approximation algorithm to solve it. We designed two combinatorial auction-based approximation mechanisms in our previous work [6]. Although these mechanisms are able to increase the allocation efficiency of VM instances and also increase the cloud provider's revenue, they assume static provisioning of VM instances. That is, they require that the VM instances are already provisioned and would not change. Static provisioning leads to inefficiencies due to under-utilization of resources if it cannot accurately predict the user demand. Since a regular auction computes the price of the items based on user demands, a very low demand may require the auctioneer to set a reserve price to prevent losses.

In this paper, we address the VM provisioning and allocation problem by designing a combinatorial auction-based mechanism that produces an efficient allocation of resources and high profits for the cloud provider. The mechanism extends one of the mechanisms we proposed in [6] to include dynamic configuration of virtual machine instances and reserve prices. The proposed mechanism, called CA-PROVISION, treats the set of available computing resource as 'liquid' resources that can be configured into different numbers and types of VM instances depending on the requests of the users. The mechanism determines the allocation based on the users' valuations until all resources are allocated. It involves a reserve price determined by the operating cost of the resources. The reserve price ensures that a user has to pay a minimum amount to the cloud provider so that the provider does not suffer any losses from the VM provisioning and allocation.

We would like to mention that our previous work [6] showed that the VM allocation problem can be best solved by a combinatorial auction-based mechanism. Our focus was to evaluate the combinatorial auctions against fixed-price mechanisms in solving the VM allocation problem with static provisioning in clouds. In this paper, we design a combinatorial auction-based mechanism that dynamically provisions and allocates VM instances.

Related Work. Researchers approached the problem of VM provisioning in clouds from different points of view. Shivam *et al.*[7] presented two systems called Shirako and NIMO that

978-0-7695-4622-3/11 $26.00 © 2011 IEEE
DOI 10.1109/CloudCom.2011.24

complement each other to obtain on-demand provisioning of VMs for database applications. Shirako does the actual provisioning and NIMO guides it through active learning models. The CA-PROVISION mechanism we present in this paper performs both demand tracking and provisioning via a combinatorial auction. Dornemann *et al.* [8] proposed on-demand resource provisioning for the Business Process Execution Language (BPEL). Their work extends the BPEL engine so that it can support scientific workflows by dynamically provisioning resources from Amazon EC2, when the demand surpasses the capacity of the BPEL host.

Dynamic provisioning of computing resources was investigated in [9] where a decentralized online clustering algorithm for VM provisioning based on the workload characteristics was proposed. The authors proposed a model-based approach to generate workload estimates on a long-term basis. Our proposed mechanism provisions the VMs dynamically and it does not require to predict the workload behaviors, rather, the current demand for VMs is captured and the provisioning is decided by a combinatorial auction-based mechanism. Van *et al.* [10] proposed an autonomic resource management system that decouples VM allocation from the physical mapping of VM to resources. They showed that their approach can simultaneously satisfy both service level agreement and resource utilization criteria.

Recently, researchers investigated economic models for resource allocation in computational grids. Wolski *et al.* [11] compared commodities markets and auctions in grids in terms of price stability and market equilibrium. Gomoluch and Schroeder [12] simulated the double auction protocol for resource allocation in grids and showed that it outperforms the conventional round-robin approach. Das and Grosu [13] 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.

Altmann *et al.* [14] 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 and removed from the system. In [15], a testbed for cloud services was designed to be able to test different mechanisms on clouds. The authors deployed the exchange mechanism described in [14] on this platform. In this paper, we consider designing a combinatorial auction mechanism with reserve price instead of an exchange. In this case, instead of specifying an asking price, the cloud provider determines a reserve price that is based on its cost parameters. Also, the outcome of the auction determines the configuration of VM instances that needs to be provisioned.

A detailed survey on combinatorial auctions can be found in [16]. The book by Cramton *et al.* [17] provides good foundational knowledge on this topic. Lehmann *et al.* [18] studied combinatorial auctions with single-minded bidders and devised a greedy mechanism for combinatorial auctions. In our previous work [6], we extended this mechanism and developed CA-GREEDY, a combinatorial auction-based mechanism to allocate VM instances in clouds. We showed that CA-GREEDY can efficiently allocate VM instances in clouds generating higher revenue than the currently used fixed-price mechanisms. However, CA-GREEDY requires that the VMs are provisioned in advance, that is, they require static provisioning. The mechanism we propose in this paper is different from CA-GREEDY in that it selects the set of VM instances in a dynamic fashion which reflects the market demand at the time when the mechanism is executed.

Our Contribution. We formulate the dynamic VM provisioning and allocation problem and provide a combinatorial auction-based mechanism to solve it. Our mechanism ensures a higher profit for the cloud provider, as well as better utilization of resources. We analyze the cost and benefit of running this new mechanism and provide implementation guidelines. Using the proposed mechanism, it is possible to achieve higher utilization of resources, higher profit for the cloud provider, along with yielding an efficient allocation of resources. We evaluate our mechanism by performing simulation experiments using traces of real workload from Parallel Workload Archive [19].

Organization. The rest of the paper is organized as follows. In Section II, we formulate the problem of dynamic VM provisioning and allocation in clouds. In Section III, we present our proposed mechanism for solving the VM provisioning and allocation problem and characterize its theoretical properties. In Section IV, we perform extensive simulations on real workload traces to investigate the properties of our proposed mechanism. In Section V, we conclude the paper and discuss possible future research directions.

II. Dynamic VM Provisioning and Allocation Problem

Virtualization technology allows the cloud computing providers to configure computational resources into virtually any combination of different types of VMs. Hence, it is possible to determine the best combination of VM instances through a combinatorial auction and then dynamically provision them. This will ensure that the number of VM instances of different types are determined based on the market demand and then allocated efficiently to the users. We formulate the Dynamic VM Provisioning and Allocation Problem (DVMPA) as follows.

A cloud provider offers computing services to users through m different types of VM instances, $VM_1, \ldots, VM_m$. The computing power of a VM instance of type VM_i, $i = 1, \ldots, m$ is w_i, where $w_1 = 1$ and $w_1 < w_2 < \ldots < w_m$. We denote by $\mathbf{w} = (w_1, w_2, \ldots, w_m)$ the vector of computing powers of the m VMs. In the rest of the paper we will refer to this vector as the 'weight vector'. As an example of how we use this vector, let us consider a cloud provider offering three types of VM instances: VM_1, consisting of

one 2 GHz processor, 4 GB memory, and 500 GB storage; VM_2, consisting of two 2 GHz processors, 8 GB memory, and 1 TB storage; and VM_3, consisting of four 2 GHz processors, 16 GB memory, and 2 TB storage. The weight vector characterizing the three types of VM instances is thus, $\mathbf{w} = (1, 2, 4)$. We assume that the cloud provider has enough resources to create a maximum of M VM instances of the least powerful type, VM_1. The cloud provider can provision the VM instances in several ways according to the specified types given by $VM_1, \ldots, VM_m$. Let's denote by k_i the number of VM_i instances provisioned by the cloud provider. The provider can provision any combination of instances given by the vector $(k_1, k_2, \ldots, k_m)$ as long as $\sum_{i=1}^{m} w_i k_i \leq M$.

We consider n users $u_1, \ldots, u_n$ who request computing resources from the cloud provider specified as bundles of VM instances. A user u_j requests VM instances by submitting a bid $B_j = (r_1^j, \ldots, r_m^j, v_j)$ to the cloud provider, where r_i^j is the number of instances of type VM_i requested and v_j is the price user u_j is willing to pay to use the requested bundle of VMs for a unit of time. An example of a bid submitted by a user to a cloud provider that offers three types of VMs can be $B_j = (2, 1, 4, 10)$. This means that the user is bidding ten units of currency for using two instances of type VM_1, one instance of type VM_2, and four instances of type VM_3 for one unit of time. The provider runs a mechanism, in our case an auction, periodically (e.g., once an hour) to provision and allocate the VM instances such that its profit is maximized. In order to define the profit obtained by the cloud provider we need to introduce additional notation. Let's denote by p_j the amount paid by user u_j for using her requested bundle of VMs. Note that depending on the pricing and allocation mechanism used by the cloud provider p_j and v_j can have different values, usually $p_j \leq v_j$.

Let us assume that the time interval between two consecutive auctions is one unit of time. Let c_R and c_I be the costs associated with running, respectively idling a VM_1 instance for one unit of time. Obviously, $c_R > c_I$. The cloud provider's cost of running all available resources (i.e., all M VM_1 instances) is $M \cdot c_R$ while the cost of keeping all the available resources idle is $M \cdot c_I$. We denote by $x = (x_1, x_2, \ldots, x_n)$ the allocation vector, where $x_i = 1$ if the bundle requested by user u_j was allocated to her, and $x_i = 0$, otherwise. Given a particular allocation vector and payments, the cloud provider's profit is given by

$$P = \sum_{j=1}^{n} x_j p_j - c_R \sum_{j=1}^{n} x_j s_j - c_I \left(M - \sum_{j=1}^{n} x_j s_j \right) \quad (1)$$

where $s_j = \sum_{i=1}^{m} w_i r_i^j$, that is, the amount of 'unit' computing resources requested by user u_j. The 'unit' computing resource is equivalent to one VM instance of type VM_1 (i.e., the least powerful instance offered). The first term of the equation gives the revenue, the second term gives the running cost of the VM instances that are allocated to the users, and the third term gives the cost of keeping the remaining resources idle.

The problem of Dynamic VM Provisioning and Allocation (DVMPA) in clouds is defined as follows

$$\max P \quad (2)$$

subject to: (i) $\sum_{j=1}^{n} s_j \leq M$; (ii) $x_j \in \{0, 1\}$; and (iii) $0 \leq p_j \leq v_j$. The solution to this problem consists of allocation x_j and price p_j for each user u_j who requested the bundle $(r_1^j, \ldots, r_m^j)$, $j = 1, \ldots, n$. The allocation will determine the number of VMs of each type that needs to be provisioned as follows. We compute $k_i = \sum_{j=1}^{n} x_j r_i^j$, for each type VM_i and provision k_i VM instances of type VM_i.

Current cloud service providers use a fixed-price mechanism to allocate the VM instances and rely on statistical data to provision the VMs in a static manner. In previous work [6], we have shown that combinatorial auction-based mechanisms can efficiently allocate VM instances in clouds generating higher revenue than the currently used fixed-price mechanisms. However, the combinatorial auction-based mechanisms we explored in [6] require that the VMs are provisioned in advance, that is, they require static provisioning. We argue that the overall performance of the system can be increased by carefully selecting the set of VM instances in a dynamic fashion which reflects the market demand at the time when an auction is executed. In the next section we propose a combinatorial auction-based mechanism that solves the DVMPA problem by determining the allocation, pricing, and the best configuration of VMs that need to be provisioned by the cloud provider in order to obtain higher profits. Since very little is know about profit maximizing combinatorial auctions [5], we cannot provide theoretical guarantees that our auction-based mechanism maximizes the profit. The only guarantee we can provide is that the mechanism maximizes the sum of the users' valuations. In designing our mechanism we also use reserve prices which are known to increase the revenue of the auctioneer, in our case, the revenue of the cloud provider.

III. Combinatorial Auction-Based Dynamic VM Provisioning and Allocation Mechanism

We present a combinatorial auction-based mechanism, called CA-PROVISION, that computes an approximate solution to the DVMPA problem. That is, it determines the prices the winning users have to pay, and the set of VM instances that need to be provisioned to meet the winning users' demand. The mechanism also ensures that the maximum possible number of resources are allocated and no VM instance is allocated for less than a reserve price. The design of the mechanism is based on the ideas presented in [18].

CA-PROVISION uses a reserve price to guarantee that users pay at least a given amount determined by the cloud provider. Thus, the cloud provider needs to set the reserve price, denoted by v_{res}, to a value which depends on its costs associated with running the VMs. To do that we observe that the reserve price should be the break-even point between c_R and c_I, which is given by $c_R - c_I$. This is because if a unit resource is not allocated, it incurs a loss of c_I. Again, if this resource is allocated for a price $c_R - c_I$, the loss is $c_R - (c_R - c_I) = c_I$.

Algorithm 1 CA-PROVISION Mechanism

Input: M; m; $w_j : j = 1, \ldots, n$; c_R; c_I;
Returns: W; $p_j : j = 1, \ldots, n$; $k_i : i = 1, \ldots, m$;
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 and provisioning*}
6: $W \leftarrow \emptyset$ {set of winners}
7: $v_{res} \leftarrow c_R - c_I$
8: add dummy user u_0 with bid $B_0 = (1, 0, 0, \ldots, 0, v_{res})$
9: **for** $j = 0, \ldots, n$ **do**
10: $s_j \leftarrow \sum_{i=1}^{m} r_i^j w_i$
11: $d_j \leftarrow v_j / s_j$ {'bid density'}
12: **end for**
13: re-order users $u_1, \ldots, u_n$ such that
$d_1 \geq d_2 \geq \ldots \geq d_n$
14: let l be the index such that
$d_j \geq d_0$ if $j \leq l$, and
$d_j < d_0$ otherwise
15: discard users $u_{l+1}, \ldots, u_n$
16: rename user u_0 as u_{l+1}
17: set $n \leftarrow l + 1$
18: $R \leftarrow M$
19: **for** $j = 1, \ldots, n-1$ **do** {leave out dummy user}
20: **if** $s_j \leq R$ **then**
21: $W \leftarrow W \cup u_j$
22: $R \leftarrow R - s_j$
23: **end if**
24: **end for**
25: **for** $i = 1, \ldots, m$ **do** {determine VM configuration}
26: $k_i \leftarrow \sum_{j:u_j \in W} r_i^j$
27: **end for**
28: {*Phase 3: Payment*}
29: **for all** $u_j \in W$ **do**
30: $W_j' \leftarrow \{u_l : u_j \notin W \Rightarrow u_l \in W\}$
31: $l \leftarrow$ lowest index in W_j'
32: $p_j \leftarrow d_l s_j$
33: **end for**
34: **for all** $u_j \notin W$ **do**
35: $p_j \leftarrow 0$
36: **end for**
37: **return** $(W, \mathbf{p}, \mathbf{k})$

In other words, the minimum price a user has to pay for using the least powerful VM for a unit of time is equal to the difference between the cost of running and the cost of keeping the resource idle. An auction with reserve price v_{res} can be modeled by an auction without reserve price in which we artificially introduce a dummy bidder u_0 having as its valuation the reserve price, i.e., $v_0 = v_{res}$. The dummy user u_0 bids $B_0 = (1, 0, \ldots, 0, v_{res})$, i.e., $r_1^0 = 1$, $r_i^0 = 0$ for all $i = 2, \ldots, m$, and $v_0 = v_{res}$. CA-PROVISION uses the density of the bids to determine the allocation. User u_j's *bid density* is $d_j = v_j / s_j$, where $s_j = \sum_{i=1}^{m} w_i r_i^j$, $j = 0, \ldots, n$. The bid density is a measure of how much a user bids per unit of allocation. In our case the unit of allocation corresponds to one VM instance of type VM_1. To guarantee that the users are paying at least the reserve price the mechanism will discard all users for which $d_j < d_o$.

CA-PROVISION is given in Algorithm 1. The mechanism requires some information from the system such as the total amount of computing resources M expressed as the total number of VMs of type VM_1 that can be provisioned by the cloud provider. The mechanism also requires as input the number of available VM types, m, and their weight vector $\mathbf{w}$. It also needs to know c_R, the cost of running a VM instance of type VM_1, and c_I, the cost of keeping idle a VM instance of type VM_1.

The mechanism works in three phases. In Phase 1, it collects the users' bids B_j (lines 1 to 4). In Phase 2, the mechanism determines the winning bidders and the VM configuration that needs to be provisioned by the cloud provider as follows. It adds a dummy user u_0 with a bid that contains only one instance of VM_1 and has a valuation of $v_{res} = c_R - c_I$ (line 8). This dummy user is only used to model the auction with reserve price and will not receive any allocation. It then computes the bundle size s_j and bid density d_j of all users (lines 9 to 12). Then, all users except the dummy user are ordered in decreasing order of their bid densities and all users u_j with $d_j < d_0$ are discarded (lines 13–15). We then move the dummy user u_0 to the end of the list of the remaining users, since the dummy user has the lowest density in the current set of users and reassign n to be the total number of users under consideration, including the dummy user (lines 16–17).

Next, the mechanism determines the winning users in a greedy fashion. It allocates the requested bundles to users in decreasing order of their bid density, as long as there are resources available. However, the dummy user is not considered for allocation. Once the winners are determined, the mechanism determines the VM configuration that needs to be provisioned by aggregating the bundles requested by the winning users (lines 25 to 27).

In Phase 3, the mechanism determines the payment for all users. For each winning bidder u_j, it finds the losing bidder u_l who would win if u_j would not participate. User u_j's payment is then calculated by multiplying her bundle size s_j with the bid density d_l of u_l. All losing bidders pay zero. This type of payment is known in the mechanism design literature as the *critical payment* [18].

We now investigate the properties of the proposed mechanism. One useful property is *truthfulness*. In our context a mechanism is truthful if the users maximize their utilities by bidding their true valuation for the bundle of VMs. Here, the utility of a user u_j is the difference between v_j, the valuation of user u_j for the bundle and p_j, the payment handed to the mechanism for using the bundle. This property is very important since the users participating in a truthful allocation mechanism do not have to employ sophisticated bidding strategies to maximize their utilities. They just need to bid their true valuation for the bundle of VMs. Truthfulness was well investigated and characterized in the mechanism design literature [5]. One such useful characterization gives the condition under which an allocation mechanism is truthful. Stated informally, an allocation mechanism is truthful if the allocation function is monotone and the payments are the critical payments. This characterization allows us prove the

following theorem.

Theorem 1: CA-PROVISION is truthful.

Proof: (Sketch) We first show that the CA-PROVISION allocation is monotone. A user can increase the chance of winning her requested bundle by increasing the density of her bid. The density increases with the reported valuation and decreases if the bundle size increases. As a result, a user can only bid higher or request a smaller bundle if she wants to go up in the order used by CA-PROVISION to decide the allocation. Therefore, the CA-PROVISION allocation is monotone. Next, the way the CA-PROVISION payments are designed, a user needs to pay a price for her bundle such that the average price per unit of VM instance is equal to her critical payment. She only pays the minimum amount she would require to bid for winning her bundle. Monotone allocation and critical payment guarantees that CA-PROVISION is truthful. The reserve prices do not affect the truthfulness of the mechanism since they are basically bids put out by the dummy user controlled by the cloud provider and truthful bidding is still a dominant strategy for the users. ■

We next investigate the complexity of CA-PROVISION. The loops in lines 19-24 and lines 29-33 constitute the major computational load of Algorithm 1. The first loop has a worst case complexity of $O(M)$, i.e., when all winning bidders bid for bundles containing exactly one unit of VM_1 instances. The total execution time of the loop in lines 29-33 is $O(n)$. This is because it iterates over the set of winning bidders and the search is performed on the losing bidders. Since the bidders are already sorted, the search for a critical payment for a winner u_{j+1} actually starts from the 'critical payment bidder' u_l of u_j (without loss of generality, we assume both u_j and u_{j+1} are winners in this case). Hence, the overall worst case complexity of this loop is $O(n)$, whereas the sorting in line 13 costs $O(n \log n)$. Thus, that the complexity of CA-PROVISION is $O(M + n \log n)$.

IV. Experimental Results

We perform extensive simulation experiments with real workload data to evaluate the CA-PROVISION mechanism. We compare the performance of CA-PROVISION with the performance of a combinatorial auction-based mechanism, called CA-GREEDY, that uses static VM provisioning. In our previous work [6] we investigated the performance of CA-GREEDY against the performance of the fixed-price VM allocation mechanism in use by current cloud providers. The mechanism showed significant improvements over the fixed-price allocation mechanism, thus, making it a good candidate for our current experiments. The total number of experiments is 264, which are generated using eleven workload logs from the Parallel Workloads Archive [19] and 24 different combination of other parameters for each workload.

A. Experimental Setup

The experiments consist of generating job submissions from a given workload and then running both CA-GREEDY and CA-PROVISION concurrently to allocate the jobs and provision the VMs. For setting up the experiments we have to address several issues such as workload selection, bid generation, and setting up the auction. We discuss all these issues in the following subsections.

TABLE I
Statistics of workload logs

Logfile	Duration (hours)	Jobs/ hour	Avg. runtime	Avg proc. per job	No. of proc.
ANL-Intrepid-2009	5759	11.97	2.09	5063	163,840
DAS2-fs0-2003	8744	25.81	1.09	10.27	144
DAS2-fs1-2003	8633	4.67	1.23	8.38	64
DAS2-fs2-2003	8760	7.58	1.29	9.45	64
DAS2-fs3-2003	8712	7.66	1.17	4.96	64
DAS2-fs4-2003	7963	4.24	1.67	3.66	64
LLNL-Atlas-2006	4308	9.92	2.52	400.7	9,216
LLNL-Thunder-2007	3605	33.58	1.52	42.54	4,008
LLNL-uBGL-2006	5339	21.09	1.25	575.8	2,048
LPC-EGEE-2004	5728	41.01	1.80	1	140
SDSC-DS-2004	9387	10.24	2.88	62.41	1,664

1) Workload selection: To the best of our knowledge, standard cloud computing workloads were not publicly available at the time of writing this paper. Thus, to overcome this limitation we rely on well studied and standardized workloads from The Parallel Workloads Archive [19]. This archive contains a rich collection of workloads from various grid and supercomputing sites. Out of the twenty-six real workloads available, we selected eleven logs from the ones that were reported most recently. These logs are: 1) ANL-Intrepid-2009, from a Blue Gene/P system at Argonne National Lab; 2) DAS2-fs0-2003 - DAS2-fs4-2003, from a research grid of five clusters at the Advanced School of Computing and Imaging in the Netherlands; 3) LLNL-Atlas-2006 and LLNL-Thunder-2007 from two Linux clusters (Atlas and Thunder) located at Lawrence Livermore National Lab; 4) LLNL-uBGL-2006, from a Blue Gene/L system at Lawrence Livermore National Lab; 5) LPC-EGEE-2004, from a Linux cluster at The Laboratory for Corpuscular Physics, Univ. Blaise-Pascal, France; and 6) SDSC-DS-2004, from a 184-node IBM eServer pSeries 655/690 called DataStar located at the San Diego Supercomputer Center. The number of jobs submitted ranges from many thousands to more than a couple of hundred thousands, while the number of processors ranges from 64 to 163,840. These large variations in the number of processors and the number of submitted jobs make these logs very suitable for experimentation, providing us with a wide range of simulation scenarios. We list some statistics of the workload files in Table I.

2) Job and bid generation: For each record in a log file we generate a job that a user needs to execute and create a bid for it. There are two important parameters associated with a job that we need to generate, the requested bundle of VMs and the associated bid. First, to generate the bundle of VM instances for a job, we determine its communication to computation ratio as follows

$$\text{Communication ratio} = 1 - \frac{\text{Average CPU time}}{\text{Total run time}}.$$

TABLE II
SIMULATION PARAMETERS

Name	Description	Value(s)
N	Total users	From log file
M	Total CPUs	From log file
T	Simulation hours	From log file
(c_I, c_R)	Idle and running cost of unit VM	(.05, .1), (.1, .25), (.15, .5)
μ	Factor for CPUs for 'first choice' VM type	50%, 75%
$\mathbf{h}$	Static distribution of processors among VM types	(.25, .25, .25, .25), (.07, .13, .27, .53)
$\mathbf{f}$	Valuation factors for types of users	(.5, 1, 1.5, 2, 2.5), (1, 1.5, 2, 3, 4)
C_1, C_2, C_3	Boundaries of communication ratios	(.05, .15, .25)

This measures the fraction of the execution time that is spent for communication among processes of a given job. Based on this value, we categorize the job in one of the m categories, where m is the number of VM types available. The job category specifies a 'first choice' of VM type for the job. This works as follows. We define a factor μ that characterizes how many of the total requested VMs will be requested as 'first choice' type VM instances. For example, a job of category i requesting q_j processors will create a bundle comprising a number of VM_i instances required to allocate μq_j processors. The rest of the processors will be requested by arbitrarily choosing other VM types. After creating the bundle, we generate the associated bid. To do that we first determine the speedup of the job as follows

$$\text{speedup} = \text{number of CPU} \times \frac{\text{average time per CPU}}{\text{total run time}}.$$

This speedup is multiplied by a 'valuation factor' to generate the bid. This valuation factor is linked to the type of user. We divide the users into five categories using their user ID, modulo five. The last parameter we set for a job is its deadline, for which there is no information provided in the workload logs. We assume that the deadline is between 4 times to 8 times the time required to complete the job. Hence, we set the deadline of a job to the required time multiplied by a random number between 4 and 8.

We run CA-GREEDY and CA-PROVISION mechanisms concurrently and independently on the jobs available for execution. A user (or job) participates in the auction until her job completes or it becomes certain that the job cannot finish by the deadline. A user is 'served' if her job completes execution and 'not served' otherwise. Without loss of generality, we assume that each user is submitting only one job and we will use 'user' and 'job' interchangeably in the rest of the paper.

3) Auction setup: We consider a cloud provider that offers four different types of virtual machines VM_1, VM_2, VM_3, and VM_4. These VM types are characterized by the weight vector $\mathbf{w} = (1, 2, 4, 8)$. From each workload file, we extract N, the total number of users and M, the total number of processors available. The number of users participating in a particular auction is determined dynamically as the auction progresses. That is, n is the number of users that has been generated, not yet been served, and the deadline for their job has not been exceeded yet.

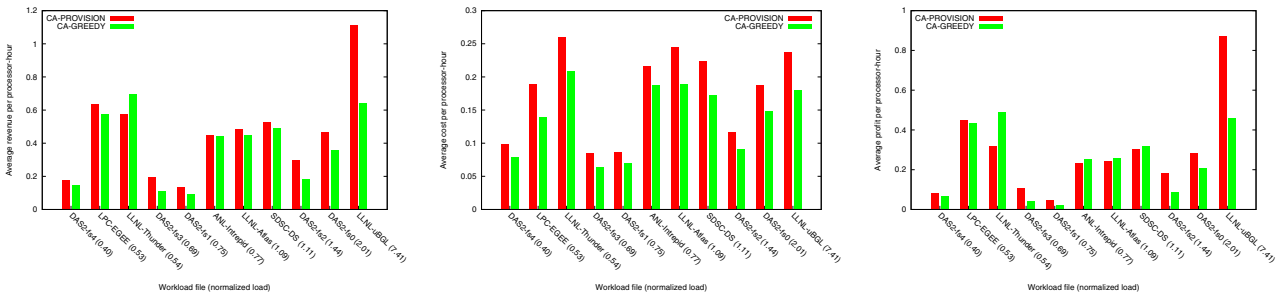
We setup few parameters to generate bundles specific to the jobs submitted by a user. The vector (C_1, C_2, C_3) determines the communication ratios used to categorize the jobs. We use $(C_1, C_2, C_3) = (0.05, 0.15, 0.25)$, as follows. A job having communication ratio below 0.05 is a job of type 1 and the majority of its needed VM instances μq_j will be requested as VM_1, where q_j is the number of processors requested by user u_j. We consider the following values for μ, 0.5 and 0.75. The rest of the bundle is arbitrarily determined using the other types of VM instances. We use the user ID field of the log file to determine the valuation range of the user. There are five classes of users submitting jobs. The class t of a user is determined by $(\text{user ID})\%5$. The logs have real user IDs, therefore this classification virtually creates a practical distribution of users. Each class t of users is associated with a 'valuation factor' f_t. Having determined that a user is of class t, we determine the valuation of her bundle using the speedup (as shown in the previous subsection) and the 'valuation factor' f_t from the vector $\mathbf{f}$. The vector $\mathbf{f}$ has five elements (equal to the number of classes of users), each representing the mean value of how much a user of that class 'values' each 'unit of speedup'. In particular, a user u_j having a speedup of S_j for her job is willing to pay $f_t S_j$ on average for each hour of her requested bundle of VMs, given that u_j falls in class t. We generate a random value between 0 and $2f_t$, and then multiply it with S_j to generate valuations with a mean of $f_t S_j$. We use two sets of vectors for $\mathbf{f}$, as shown in Table II.

CA-PROVISION determines by itself the configuration of the VMs that needs to be provisioned by the cloud provider, whereas CA-GREEDY assumes static VM provisioning, and thus, needs the VM configuration provisioned in advance. To generate the static provision of VMs required by CA-GREEDY we use a vector $\mathbf{h}$ as follows. We consider two instances of $\mathbf{h}$ in the simulation. The first one, $\mathbf{h}$ = (0.07, 0.13, 0.27, 0.53) ensures that, given the weight vector $\mathbf{w}$, the number of VM instances of each type are equal or almost equal. The other instance of this vector, $\mathbf{h} = (0.25, 0.25, 0.25, 0.25)$ ensures that the total number of processors are equally distributed to different types of VMs. We list all simulation parameters in Table II. With all combinations of values, we perform 24 experiments with each log file, for a total of 264 experiments.

B. Analysis of Results

We investigate the performance of the two mechanisms for different workloads. Since the workloads are heterogeneous in several dimensions, we first define a metric in order to characterize the workloads, and thus, be able to establish an order among them. Then, we normalize the performance values of the mechanisms and compare them with respect to the workload characteristics.

We define a metric for comparing the workload logs as follows. Looking at the workload characteristics listed in Table I, we determine that the best metric to compare the



(a) (b) (c)

Fig. 1. Comparison between normalized performance of CA-PROVISION and CA-GREEDY: (a) Average revenue per processor-hour; (b) Average cost per processor-hour; (c) Average profit per processor-hour; vs normalized load. Horizontal axis shows the log file name with normalized load in parenthesis.

workloads is the *normalized load* defined as:

$$\frac{\text{Jobs per hour} \times \text{Avg runtime} \times \text{Avg processors per job}}{\text{Total processors}}.$$

The number of jobs per hour multiplied with the average processors per job determines how many processors are requested by the jobs arriving each hour. Multiplying this with the average runtime gives an estimate of the average number of processors requested by all jobs in an hour. The normalized load gives us an ordering of the set of workloads.

From each set of simulation experiments, we compute the total revenue generated, the total cost incurred, and the total profit earned by each auction. But the workloads were generated for different durations of time for systems with different number of processors. We therefore scale the profit, revenue, and cost with respect to the total simulation hours and the number of processors. We define the *profit per processor-hour* as:

$$\frac{\text{Profit}}{\text{Total processors} \times \text{Total hours}}.$$

We define *revenue per processor-hour* and *cost per processor-hour* in a similar fashion.

We plot the average revenue, the average cost, and the average profit per processor-hour versus the workload logs in Figures 1a to 1c. In these figures, the workloads are sorted in ascending order of their normalized load. Note that the CA-PROVISION mechanism yields higher revenue in most of the cases. And for normalized load 1.44 and beyond the revenue obtained by CA-PROVISION steadily increases exceeding that obtained by CA-GREEDY by up to 40%. This leads us to conclude that CA-PROVISION is capable of generating higher revenues where there is high demand for resources.

In Figure 1b we observe that CA-PROVISION incurs a higher total cost for all workloads. Since CA-PROVISION decides about the number of VMs dynamically, it can allocate a higher number of processors than CA-GREEDY in an auction with identical bidders. Since a running processor incurs more cost ($c_R > c_I$), allocating more processors definitely leads to a higher total cost.

Now the question is whether the interplay between increased revenue and increased cost can generate a higher profit. Utilizing more resources means serving more customers hence selecting more bidders as winners. This has two mutually opposite effects on the revenue. Obviously, increasing the number of winners has a positive effect on revenue. On the other hand, selecting more winners pushes down their critical values, and thus, individual payments decrease. If the net effect is positive, we get a higher revenue and when it surpasses the increase in cost, we obtain a higher profit, and thus, achieve economies of scale. From Figure 1c we see that for normalized loads greater than 1.44, CA-PROVISION consistently generates higher profit than CA-GREEDY and the difference in profit grows rapidly. We also observe that for the workloads having load factors below 1.44 CA-PROVISION and CA-GREEDY has higher profit in equal number of cases. This suggests that for low loads the relative outcome of the mechanisms depends on other parameters.

In Figures 2 and 3 we compare the resource utilization and the percentage of served users obtained by the two mechanisms. CA-PROVISION achieves higher values for both utilization and percentage of served users. We want to draw the attention of the reader to the fact that in most of the cases the difference in utilization is around 30%. This is where we can improve a lot if we switch from static allocation to a dynamic provisioning and allocation. Since combinatorial auctions are already established tools for efficient allocation, combining them with dynamic provisioning can lead to a highly efficient resource allocation mechanism for clouds.

The number of users served is higher for CA-PROVISION because the VM instances are not statically provisioned. Therefore, a user requesting two VM_1 instances will not be left unallocated if there are no VM_1 instances available but a VM_2 instance is available as in the case of CA-GREEDY. Rather, CA-PROVISION 'sees' the available resource as a computing resource equivalent to two VM_1 instances and will allocate this, for instance, to a user bidding for two VM_1 instances or a user bidding for one VM_2 instances, depending on whose reported valuation is higher. This approach increases the number of users served by CA-PROVISION mechanism.

We can summarize the experimental results as follows. The CA-GREEDY mechanism is capable of generating higher revenue when there is matching demand with the supply. Also,

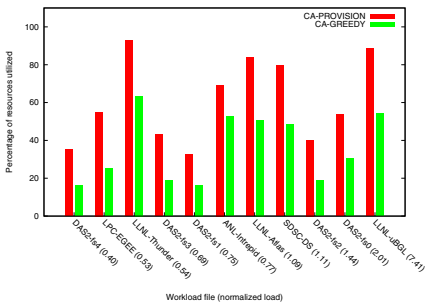


Fig. 2. Resource utilization by CA-PROVISION and CA-GREEDY vs. normalized load

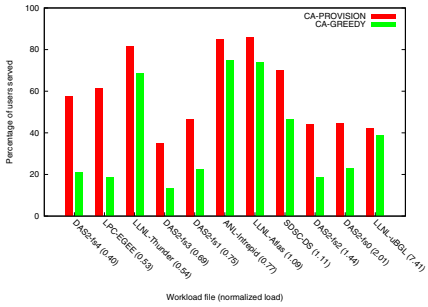


Fig. 3. Percent users served by CA-PROVISION and CA-GREEDY vs. normalized load

in an auction where items are not 'configurable' as in the case of cloud auctions, CA-GREEDY is a very efficient auction. But when we have reconfigurable items as in clouds, it is very hard to predict the demand very well in advance. In that case, CA-PROVISION is a better option and as today's technology supports, it can be deployed as a stand-alone configuration and allocation tool without much human intervention. There is also another use of this mechanism. One can combine CA-GREEDY and CA-PROVISION in a way that periodically CA-PROVISION will be executed to capture the current market demand, determine the static allocation that best matches the demand, and instantiate CA-GREEDY. If the utilization falls below a certain threshold, CA-PROVISION can be called to determine a good configuration again. This can also eliminate the need of detailed statistical analysis of demand to find an efficient static configuration for CA-GREEDY.

V. Conclusion

We addressed the problem of dynamically provisioning VM instances in clouds in order to generate higher profit, while determining the VM allocation with a combinatorial auction mechanism. We designed a mechanism called CA-PROVISION to solve this problem. We performed extensive simulation experiments with real workloads to evaluate our mechanism. The results showed that CA-PROVISION can effectively capture the market demand, provision the computing resources to match the demand, and generate higher revenue for the cloud provider, especially in high demand cases. In some of the low demand cases, CA-GREEDY performs better than CA-PROVISION in terms of profit but not in terms of utilization and percentage of served users. We conclude that an efficient VM instance provisioning and allocation system can be designed combining these two combinatorial auction-based mechanisms. We look forward to setting up a private cloud and implementing such a system in near future.

Acknowledgment

This work is partially supported by NSF grants DGE-0654014 and CNS-1116787.

References

[1] Microsoft, "Windows Azure platform, `http://www.microsoft.com/windowsazure/`."

[2] Amazon, "Amazon Elastic Compute Cloud (Amazon EC2), `http://aws.amazon.com/ec2/`."

[3] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proc. of the 9th IEEE/ACM Intl. Symp. on Cluster Comp. and the Grid*, May 2009, pp. 124–131.

[4] R. Wang, "Auctions versus posted-price selling," *The American Economic Review*, vol. 83, no. 4, pp. 838–851, 1993.

[5] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[6] 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.

[7] P. Shivam, A. Demberel, P. Gunda, D. Irwin, L. Grit, A. Yumerefendi, S. Babu, and J. Chase, "Automated and on-demand provisioning of virtual machines for database applications," in *Proc. ACM SIGMOD International Conference on Management of Data*, 2007, pp. 1079–1081.

[8] T. Dornemann, E. Juhnke, and B. Freisleben, "On-demand resource provisioning for BPEL workflows using amazon's elastic compute cloud," in *Proc. 9th IEEE/ACM Intl. Symp. on Cluster Comp. and the Grid*, May 2009.

[9] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in *Proc. 10th IEEE/ACM International Conference on Grid Computing*, 2009, pp. 50–57.

[10] H. N. Van, F. D. Tran, and J.-M. Menaud, "Autonomic virtual resource management for service hosting platforms," in *Proc. ICSE Workshop on Software Engineering Challenges in Cloud Computing*, 2009.

[11] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan, "Analyzing market-based resource allocation strategies for the computational grid," *Intl. J. of High Performance Comp Appl.*, vol. 15, no. 3, pp. 258–281, 2001.

[12] J. Gomoluch and M. Schroeder, "Market-based resource allocation for grid computing: A model and simulation," in *Proc. 1st International Workshop on Middleware for Grid Computing*, 2003, pp. 211–218.

[13] A. Das and D. Grosu, "Combinatorial auction-based protocols for resource allocation in grids," in *Proc. 19th International Parallel and Distributed Processing Symposium, 6th Workshop on Parallel and Distributed Scientific and Engineering Computing*, 2005.

[14] 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.

[15] M. Risch, J. Altmann1, 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.

[16] S. de Vries and R. V. Vohra, "Combinatorial auctions: A survey," *INFORMS Journal on Computing*, vol. 15, no. 3, pp. 284–309, 2003.

[17] P. Cramton, Y. Shoham, and R. Steinberg, *Combinatorial Auctions*. The MIT Press, 2005.

[18] D. Lehmann, L. I. O'Callaghan, and Y. Shoham, "Truth revelation in approximately efficient combinatorial auctions," *Journal of the ACM*, vol. 49, no. 5, pp. 577–602, 2002.

[19] D. G. Feitelson, "Parallel Workloads Archives: Logs," http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.