

Efficient Bidding for Virtual Machine Instances in Clouds

Sharrukh Zaman

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

Daniel Grosu

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

Abstract—Combinatorial auctions are efficient mechanisms for allocating Virtual Machine (VM) instances to cloud computing users. Despite the fact that in general these mechanisms lead to higher revenues than the currently employed fixed-price mechanisms, the cloud computing providers do not employ them to allocate their resources. One of the main reasons is the complexity faced by the users when determining the bid (i.e., the bundle of VM instances and the bid value). We address this issue by developing an efficient bidding strategy for the users requesting VM instances. We design new metrics for evaluating bundles of VM instances based on the characteristics of the computing tasks. These metrics allow us to determine the valuation of the requested bundles and to design algorithms for selecting the best bundles to bid for. We perform simulation experiments to evaluate the proposed strategy in a simulated cloud environment.

Keywords-cloud computing; combinatorial auctions; bidding strategies;

I. INTRODUCTION

Combinatorial auctions are efficient mechanisms for allocating the VM instances in clouds [1]. In a combinatorial auction-based VM instance allocation mechanism, a user bids for a bundle of VM instances of different types required for executing her application by specifying the bundle and a value. The value represents how much the user is willing to pay for the bundle if allocated. The auction mechanism determines the set of winning users, allocates their requested bundles, and computes the amount they have to pay. The goal of the cloud provider is to efficiently allocate the available VM instances to the users and generate the maximum possible revenue. On the other hand, a user desires to maximize her own *utility*, that is, the value she derives from obtaining the bundle minus the amount she has to pay for using the bundle. If the user does not obtain any bundle, her utility is zero. The combinatorial auction mechanisms presented in [1] are *incentive-compatible*, that is, they guarantee that a user maximizes her utility by bidding her ‘true’ valuation of the requested bundle. Therefore, to achieve maximum utility, a user needs to determine (i) the bundle of VM instances that guarantees the best performance for her application, and (ii) the correct (or true) valuation of the bundle.

However, generating such an efficient bid is a nontrivial problem, especially if the user needs to execute parallel

applications on the cloud platform. The degree of parallelism of the application and the system’s parameters, such as communication delay, limit the speedup of a parallel application. Thus, it is imperative to incorporate these parameters when determining the best VM bundle necessary for executing an application. A realistic valuation of a given bundle of VM instances should also consider the performance gain of the application achieved by the bundle. Finally, the bid should not exceed the budget of the user.

Our Contribution. We address the above problem by first designing a user’s valuation function that considers both the application and the system’s parameters. The proposed valuation function determines the value of a bundle of VM instances on which a user executes her application. Then, we propose an algorithm that uses this valuation function to generate efficient bids within the users’ budgets. We analyze the complexity of our proposed bidding algorithm and perform extensive simulation experiments to investigate its properties.

Related Work. Recently, many researchers have investigated the design of efficient bidding strategies for combinatorial auctions. Sui and Leung [2] proposed an adaptive bidding strategy for combinatorial auctions. They considered a multi-round first-price combinatorial auction for allocating computing resources. In their proposed bidding strategy the bidder updates her bid dynamically based on the outcome of the previous bidding round, in order to maximize her utility. A similar idea is used in [3] to design a strategy for bidding in the Amazon EC2 Spot Instances auction [4]. The authors use statistical analysis to show how a task can be completed at reduced cost and without premature termination on the Amazon EC2 cloud. Our work differs from the above two in that we are considering a combinatorial auction mechanism which guarantees the maximum utility for a user when she bids truthfully. Therefore, our proposed bidding strategy is based on determining the user’s ‘true’ valuation for a bundle and it does not require knowledge of past auction outcomes. This is a big advantage since the users do not require information from past auctions when making bidding decisions. An *et al.* [5] proposed a bidding strategy algorithm that considers a given ‘synergy’ value between different

items, when calculating the valuation of a bundle of items in general combinatorial auctions. In our work, the synergy between the VM instances in a bundle is based on a speedup function that takes into accounts the overheads of execution on multiple VM instances. Our speedup function is an extension of the speedup function for malleable parallel jobs proposed in [6]. Carroll and Grosu [7] considered the problem of scheduling malleable parallel jobs and designed incentive-compatible scheduling mechanisms to solve it. A comprehensive survey classifying parallel applications and their scheduling strategies can be found in [8].

Other related research includes designing an agent for bidding in combinatorial auctions in grids [9] and proposing an efficient policy for obtaining cloud resources for large applications [10]. Developing scheduling algorithms that satisfy the budget constraint of the users were investigated in [11] and [12].

Organization. This paper is organized as follows. In Section II, we describe the combinatorial auction-based VM allocation mechanism that we consider. In Section III, we describe our proposed bidding strategy algorithm and analyze its complexity. In Section IV, we evaluate the proposed bidding strategy by extensive simulation experiments. In Section V we summarize our results and present possible directions for future research.

II. COMBINATORIAL AUCTIONS IN CLOUDS

In a previous paper [1], we presented two combinatorial auction mechanisms for VM allocation in clouds. Here we chose one of these mechanisms, called CA-GREEDY, to illustrate the design of our bidding strategy. The bidding strategy is not designed specifically for this mechanism and it is valid for any mechanism that solves the same VM allocation problem solved by CA-GREEDY. CA-GREEDY is based on a greedy allocation function and guarantees *incentive-compatibility* (i.e., users obtain maximum utility only by bidding their true valuations for the requested bundles).

The CA-GREEDY mechanism solves the following problem of allocating VM instances in clouds. A cloud provider, who is also the auctioneer, needs to allocate k copies of m different types of VM instances among n competing users¹. The different types of VMs are represented as $VM_1, \dots, VM_m$ and their relative ‘computing powers’ are denoted by $\mathbf{w} = (w_1, \dots, w_m)$, where w_i is the number of processors in VM_i . We assume that the VM instances have other computing resources (e.g., memory, storage, and bandwidth) in proportion to the number of processors. Here we assume that $w_1 = 1$ and $w_i \leq w_{i+1}$. An example is $\mathbf{w} = (1, 2, 4, 8)$ that means the cloud provider offers four

types of VM instances having 1, 2, 4, and 8 processors, respectively.

A user u_j , $j = 1, \dots, n$ participates in the auction for VM instances by submitting her bid $B_j = (r_1^j, \dots, r_m^j, v_j)$. Here, r_i^j is the number of VM_i instances that user u_j requests and v_j is the maximum amount that u_j is willing to pay if the bundle is allocated to her. The goal of the CA-GREEDY mechanism is to determine an allocation vector $\mathbf{x} = (x_1, \dots, x_n)$, where $x_j = 0$ indicates that user u_j does not obtain her requested bundle and $x_j = 1$ means that u_j obtains her requested bundle. The mechanism also determines a payment vector $\mathbf{y} = (y_1, \dots, y_n)$, where y_j is the amount user u_j has to pay for the allocated bundle.

The optimization problem for which the CA-GREEDY provides an approximate solution is

$$\max \sum_{j=1}^n x_j v_j \quad (1)$$

subject to,

$$\sum_{j=1}^n x_j r_i^j \leq k, \quad i = 1, \dots, m \quad (2)$$

$$y_j \leq v_j, \text{ if } x_j = 1 \quad (3)$$

$$y_j = 0, \text{ if } x_j = 0 \quad (4)$$

$$x_j \in \{0, 1\} \quad j = 1, \dots, n \quad (5)$$

Equation (1) indicates that the allocation needs to maximize the *social welfare*, which is the sum of valuations of all ‘winning’ bidders. The solution has to satisfy the following constraints: (i) the total allocation should not exceed the number of available VM instances (Equation (2)); (ii) the winners’ payment should not exceed their valuations (Equation (3)); and (iii) the users who do not obtain their bundles should pay 0 (Equation (4)).

The CA-GREEDY mechanism, presented in Algorithm 1, works as follows. It first sorts the bids in descending order of ‘density of valuation’ $v_j / \sqrt{p_j}$, where the number of processors is given by $p_j = \sum_{i=1}^m r_i^j w_i$ (lines 3 to 6). Allocation is determined greedily by selecting bids in this order as long as Equation (2) is satisfied (lines 9 to 11). To compute the payment of a winning user u_j , the mechanism finds the losing user $u_{j'}$ that has the maximum valuation density and that would win if u_j would not participate. The payment y_j of u_j is then calculated as $(v_{j'} / \sqrt{p_{j'}}) * \sqrt{p_j}$ (lines 14 to 19). This mechanism gives an approximate solution to the combinatorial auction problem and is incentive-compatible.

We assume that the combinatorial auction for VM instances is conducted periodically (e.g., once per hour). Each auction allocates the maximum possible number of resources to the users for only one period. Users who did not complete their application execution in one period need to bid again for the next period. It is the users’ responsibility to determine how many units of time they need certain resources and submit their bids until their work is completed.

¹To simplify the presentation we use the same number k of instances for each VM type. The mechanisms presented in [1] and our bidding strategy do not require the same number of copies of each type of VM.

Algorithm 1 CA-GREEDY Mechanism

Input: $B_j = (r_1^j, \dots, r_m^j, v_j)$, k ; $j = 1, \dots, n$
Returns: x_j, y_j ; $j = 1, \dots, n$

- 1: {Initialization}
- 2: $W \leftarrow \emptyset$; **for** $j = 1, \dots, n$ **do** $x_j \leftarrow 0$ { W : set of winners}
- 3: **for** $j = 1, \dots, n$ **do**
- 4: $p_j \leftarrow \sum_{i=1}^m r_i^j w_i$
- 5: $q_j \leftarrow v_j / \sqrt{p_j}$ { q_j : ‘density’ of valuation}
- 6: **end for**
- 7: {Allocation}
- 8: re-order users in decreasing order of density of valuation, that is $q_1 \geq q_2 \geq \dots \geq q_n$
- 9: **for** $j = 1, \dots, n$ **do**
- 10: **if** for all $i = 1, \dots, m$, $r_i^j + \sum_{u_{j'} \in W} r_i^{j'} \leq k$
- 11: **then** $W \leftarrow W \cup u_j$; $x_j \leftarrow 1$
- 12: **end for**
- 13: {Payment}
- 14: **for each** $u_j \in W$ **do**
- 15: $W'_j \leftarrow \{u_l : u_j \notin W \Rightarrow u_l \in W\}$
- 16: $l \leftarrow$ minimum index in W'_j
- 17: **if** $W'_j \neq \emptyset$ **then** $y_j \leftarrow q_l \sqrt{p_j}$
- 18: **else** $y_j \leftarrow 0$
- 19: **end for**
- 20: **for all** $u_j \notin W$, $y_j \leftarrow 0$
- 21: **return** $(\mathbf{x}, \mathbf{y})$

III. PROPOSED BIDDING STRATEGY

A *bidding strategy* should create efficient *bids* for users participating in combinatorial auctions for VM instances in clouds. The strategy must take into account the type and workload of the application the user intends to run, and her budget. It should generate a bundle of VM instances and a *valuation* of that bundle in order to provide the user with the best performance within her budget.

A. Execution Time and Speedup

We consider a set of users who intend to run parallel applications on clouds. In particular, we consider *malleable applications*, which are parallel applications that can be executed on any given number of processors [6]. However, by employing more processors we cannot gain speedup indefinitely. This is because by adding more processors the overhead added to the execution time increases. This overhead is a function of both system-specific and application-specific parameters, as we will discuss in the following.

Let us consider that a user u_j needs to run a malleable parallel application a_j on a cloud platform. The application is characterized by its workload ω_j , expressed as a processor-time product. Havill and Mao [6] modeled the execution time T_j of an application a_j as:

$$T_j = \omega_j / p_j + (p_j - 1)\theta \quad (6)$$

where p_j is the number of processors used, and θ is a system-specific parameter called the ‘setup time’. The setup time is the time spent to create, dispatch, or destroy multiple processes. The first term represents the time spent

executing the workload only, which is equally divided to p_j processors. The second term corresponds to the system-specific overhead (e.g., time to create or destroy a process).

We argue that the overhead of execution on multiple processors is not only dependent on the setup time but also on the time spent on distributing the input data during initialization and on the time spent on communication and synchronization during program execution. We assume that the amount of information to be exchanged during initialization and execution can be expressed as a fraction of the total workload. Hence, our proposed model for the parallel execution time of a malleable application is given by

$$T_j = \omega_j / p_j + (p_j - 1)\theta + (p_j - 1)\delta\tau_j\omega_j + (R_j - 1)\delta c_j\omega_j \quad (7)$$

Our new model adds two new terms to the existing two terms of Equation (6). The third term in the new equation for T_j characterizes the initialization overhead. Here, δ is the time to communicate one unit of data between different VM instances in the system, and τ_j is the amount of input data to be transferred, expressed as a fraction of the total application workload. Therefore, the amount of input data that is transferred is $\tau_j\omega_j$ and the overhead for data transfer during initialization is $(p_j - 1)\delta\tau_j\omega_j$. The fourth term represents the communication and synchronization overhead incurred during processing. Here, c_j is the communication overhead expressed as a fraction of the workload. R_j is the number of VM instances the total number of processors are divided into. Here we assume that communication within a VM instance takes negligible amount of time compared to the communication between VM instances. Therefore, we multiply the amount of communication by $R_j - 1$. This implies that if all p_j processors reside on one VM instance (i.e., $R_j = 1$), the overhead for communication and synchronization is negligible. Otherwise, it increases with the number of VM instances employed in running the application.

Using Equation (7) we derive the speedup of the application as:

$$S_j = \frac{\omega_j}{\omega_j / p_j + (p_j - 1)\theta + (p_j - 1)\delta\tau_j\omega_j + (R_j - 1)\delta c_j\omega_j} \quad (8)$$

This speedup equation serves as the basis for the valuation function and the bidding strategy algorithm that we propose.

B. Valuation Function and Bidding Algorithm

We now determine a valuation function which gives the ‘value’ a user receives upon completion of her application on a bundle of VM instances. The function should depend on the parameters of the application and the bundle allocated to it. We define the valuation for using a bundle for one unit of time as follows:

$$v_j = (\lambda_j S_j)(S_j / S_{opt}) \quad (9)$$

Here, S_j is given by Equation (8) and λ_j denotes how much user u_j values each unit of speedup. The valuation can be interpreted as follows. When the optimal speedup can be achieved (i.e., $S_j = S_{opt}$) within the budget of the application the valuation function is directly proportional to the speedup. When the optimal speedup cannot be achieved the valuation is proportional to the ‘scaled speedup’, where the scaling factor is given by $S_j/S_{opt} < 1$. That is if the optimal speedup is not achieved the user values less the unit of speedup than in the case in which the optimal speedup is achieved.

The valuation v_j can also be interpreted as the user’s maximum willingness to pay for the given bundle. This value is the bid amount that the user submits to the auctioneer (i.e., cloud provider) along with her requested bundle. If the bundle is allocated, user u_j needs to pay a price y_j determined by the auction. The user’s utility is defined as the surplus value retained by the user after paying the price for executing her application.

$$U_j = \begin{cases} v_j - y_j & \text{if } x_j = 1 \\ 0 & \text{if } x_j = 0 \end{cases} \quad (10)$$

Here, x_j is the indicator variable of whether user u_j receives her requested bundle of VM instances. Obviously, the user’s goal is to maximize her utility.

Now we present our proposed Efficient Bidding Strategy (EBS). As we mentioned in Section II, an auction is run each unit of time (e.g., every hour). Therefore, a bidding strategy needs to determine the bid $B_j = (r_1^j, \dots, r_m^j, v_j)$ and the time required to complete the application, T_j . Here r_i^j is the number of VM instances of type VM_i included in the requested bundle of u_j . The goal of the strategy is to find a bundle and valuation so that the user can obtain the best possible speedup within her budget V_j , and thus, maximize her utility. Since v_j is the value the user is willing to pay for the allocation of the bundle for one unit of time, the algorithm has to ensure that the maximum total cost for the user does not exceed her budget, i.e.,

$$v_j T_j \leq V_j \quad (11)$$

Our bidding strategy algorithm presented in Algorithm 2 has two phases. In the first phase, it determines the optimum number of processors (i.e., the number of processors that gives the maximum speedup). Phase 1 consists of lines 1 to 12. First, the optimal number of processors is calculated (line 2). To compute the optimal number of processors we consider that all processors are part of the same VM, that is, we set $R_j = 1$ in Equation (7). We then differentiate T_j with respect to p_j and solve $\frac{dT_j}{dp_j} = 0$ for p_j as follows.

$$-p_j^2 + \theta + \delta\tau_j\omega_j = 0 \quad (12)$$

Solving the above equation gives us the optimal value of p_j as:

$$p_{opt} = \lfloor \sqrt{\omega_j/(\theta + \delta\tau_j\omega_j)} \rfloor \quad (13)$$

Algorithm 2 Efficient Bidding Strategy (EBS)

Input: $\omega_j, \tau_j, c_j, V_j, \lambda_j$

Returns: $(B_j = (r_1^j, \dots, r_m^j, v_j), T_j)$

```

1: {Phase I: Finding optimal bundle and speedup}
2:  $p_j \leftarrow \lfloor \sqrt{\omega_j/(\theta + \delta\tau_j\omega_j)} \rfloor$ 
3:  $z \leftarrow p_j$ 
4:  $R_j \leftarrow 0$ 
5: for  $i := m$  downto 1 do {initialize  $r_i^j$ }
6:    $r_i^j \leftarrow \lfloor z/w_i \rfloor$ 
7:    $z \leftarrow z \bmod w_i$ 
8:    $R_j \leftarrow R_j + r_i^j$ 
9: end for
10:  $T_j \leftarrow \omega_j/p_j + (p_j - 1)\theta + (p_j - 1)\delta\tau_j\omega_j + (R_j - 1)\delta c_j\omega_j$ 
11:  $S_j \leftarrow \omega_j/T_j$ 
12:  $S_{opt} \leftarrow S_j$  {save value of optimal speedup}
13: {Phase II: Finding optimal valuation and adjust if necessary}
14:  $v_j \leftarrow (\lambda_j S_j)$ 
15: while  $v_j T_j > V_j$  and  $p_j > 0$  do
16:    $h \leftarrow 0$ 
17:   for  $i := m$  downto 1 do {increase number of VMs}
18:     if  $h = 0$  and  $r_i^j > 0$  then
19:        $r_i^j \leftarrow r_i^j - 1$ 
20:        $R_j \leftarrow R_j - 1$ 
21:        $h \leftarrow i$ 
22:        $z \leftarrow w_i$ 
23:     else if  $h > 0$  then
24:        $r_i^j \leftarrow r_i^j + \lfloor z/w_i \rfloor$ 
25:        $R_j \leftarrow R_j + \lfloor z/w_i \rfloor$ 
26:        $z \leftarrow z \bmod w_i$ 
27:     end if
28:   end for
29:   if  $h = 0$  then {maximum possible  $R_j$  reached}
30:      $p_j \leftarrow p_j - 1$ 
31:      $z \leftarrow p_j$ 
32:      $R_j \leftarrow 0$ 
33:     for  $i := m$  downto 1 do {initialize  $r_i^j$  for new  $p_j$ }
34:        $r_i^j \leftarrow \lfloor z/w_i \rfloor$ 
35:        $z \leftarrow z \bmod w_i$ 
36:        $R_j \leftarrow R_j + r_i^j$ 
37:     end for
38:   end if
39:    $T_j \leftarrow \omega_j/p_j + (p_j - 1)\theta + (p_j - 1)\delta\tau_j\omega_j + (R_j - 1)\delta c_j\omega_j$ 
40:    $S_j \leftarrow \omega_j/T_j$ 
41:    $v_j \leftarrow (\lambda_j S_j)(S_j/S_{opt})$  {scale valuation w.r.t speedup}
42: end while
43: if  $p_j = 0$  then
44:   return "Budget too low"
45: end if
46: return  $(B_j = (r_1^j, \dots, r_m^j, v_j), T_j)$ 

```

We then generate the bundle with the least number of VM instances possible for p_j processors (lines 3 to 9). Lines 10 and 11 compute the optimal execution time and the speedup with respect to the optimal values of p_j and R_j , respectively. We save the value of the optimal speedup for later use (line 12).

In the second phase, the algorithm adjusts the number of processors and/or number of VM instances if the valuation corresponding to the maximum speedup exceeds the user’s budget. That is, R_j is increased if the valuation exceeds

the user budget. If the valuation exceeds the budget even when $R_j = p_j$ (i.e., the bundle is composed of only one-processor VM instances), then we decrease the value of p_j and iterate until a valuation within the budget is found or $p_j = 0$, which denotes that it is not possible to execute this application under the given budget V_j .

Lines 13 to 46 constitute the second phase of the algorithm. First, we calculate the optimal valuation of the bundle for allocation of one unit time, based on the optimal speedup (line 14). If this valuation cannot satisfy Equation 11, the while loop in lines 15-42 iterates until a feasible bundle is found or it is determined that no such bundle exists. The for loop in lines 17 to 28 selects one of the largest VM instances in the bundle and replaces it with smaller VM instances, thus increasing R_j and generating a lower value bundle. The indicator variable h declared in line 16 captures the status whether we can increase R_j for the current value of p_j or not. If R_j has reached its maximum, we decrease p_j , create a new bundle (lines 29-38), and compute T_j and S_j as above (lines 39-40). For this reduced speedup, we compute v_j as before, but then scale it with the ratio of S_j and the optimal speedup (line 41). The implication of this can be understood from the following equation.

$$\begin{aligned} v_j T_j &= (\lambda_j S_j)(S_j/S_{opt})T_j = (\lambda_j \omega_j/T_j)(S_j/S_{opt})T_j \\ &= (\lambda_j \omega_j)(S_j/S_{opt}) \end{aligned} \quad (14)$$

From the above equation, we see that scaling the valuation makes sure that the maximum total cost of executing the application depends on the ratio of the achieved and the optimal speedup. Thus, the total cost of the application decreases and the loop at line 15 terminates when a solution is found within the budget constraint. The algorithm returns the bid and the time requirement if a feasible solution is found, otherwise it returns "Budget too low".

Since the EBS algorithm ensures the best speedup within the budget, this maximizes the user's utility. This is because the user bids her true valuation to a combinatorial auction that is incentive-compatible [1].

Running Time. In Algorithm 2, the while loop starting at line 15 dominates the running time. In the worst case, this loop will iterate until $p_j = 0$. In that case, the for loop at line 17 iterates p_j times for each value of p_j , where $1 \leq p_j \leq p_{opt}$. Each iteration of this loop replaces one larger VM instance with two or more smaller VM instances until the bundle contains p_j one-processor VM instances. Therefore, for one value of p_j , the loop at line 17 iterates p_j times in the worst case. The worst-case running time of the algorithm is $\sum_{i=1}^{p_{opt}} i = O(p_{opt}^2)$.

IV. EXPERIMENTAL RESULTS

We perform simulation experiments to evaluate the proposed bidding strategy. Our goal is to answer the following questions: (i) How does the bidding strategy affect users' utilities? (ii) How does the bidding strategy affect the cloud

providers' revenue? (iii) How does the number of strategic bidders participating in an auction influence other bidders' utilities?

A. Naive Bidding

We compare our proposed strategy with a 'naive strategy'. The naive bidding strategy does not consider the application's speedup function when determining the bid, but it is not completely arbitrary. A naive bidder has different values for different types of VM instances and assumes the existence of some overhead which she incorporates in her bids. The Naive-Bidding (NB) algorithm is given in Algorithm 3.

The naive user u_j assigns a value ρ_i^j to each VM_i as follows. She bases the assignment on a known fixed-price vector $f_1, \dots, f_m$ of a cloud provider where $f_i/f_{i'} = w_i/w_{i'}$, for $i = 1, \dots, m$. The value for VM_1 is determined by multiplying f_1 by a valuation factor σ_1^j (line 1). The values of the other VMs, ρ_i^j , $i = 2, \dots, m$, are determined by multiplying σ_1^j by the second valuation factor, σ_2^j , and by the relative values of other VM_i instances, $i = 2, \dots, m$ (lines 2-4).

The bid is generated in lines 6 to 12. First, a VM type is selected at random (line 6). Then the user creates a bundle and determines the time required to execute her application so that her application can be completed based on her assumption about the overhead, given by overhead factors β_1^j and β_2^j . The basis of generating the bundle is that with no overhead, r_i^j instances of VM_i should execute an application a_j with workload ω_j in t_j units of time if $w_i r_i^j t_j = \omega_j$. Given w_i , one set of values of r_i^j and t_j that satisfies this condition is $r_i^j = t_j = \sqrt{z}$, where $z = \omega_j/w_i$ (line 8). The user tries to avoid requesting an excessive number of VM instances (i.e., making an attempt to reduce the overhead)

Algorithm 3 Naive-Bidding (NB)

Input: $\omega_j, V_j, \sigma_1^j, \sigma_2^j, \beta_1^j, \beta_2^j$
Returns: $(B_j = (r_1^j, \dots, r_m^j, v_j), t_j)$

- 1: $\rho_1^j \leftarrow \sigma_1^j f_1$
- 2: **for** $i := 2$ **to** m **do** {set user's 'own' value for VM_i }
- 3: $\rho_i^j \leftarrow \sigma_2^j \rho_1^j (w_i/w_1)$
- 4: **end for**
- 5: **repeat** {select a random VM type VM_i }
- 6: $i \leftarrow \text{random}(1, m)$
- 7: {select r_i^j, t_j such that $r_i^j t_j > \omega_j/w_i$ }
- 8: $z \leftarrow \omega_j/w_i$
- 9: $r_i^j \leftarrow \lceil \beta_1^j \sqrt{z} \rceil$
- 10: $t_j \leftarrow \lceil \beta_2^j z / r_i^j \rceil$
- 11: $v_j \leftarrow \rho_i^j r_i^j$
- 12: {reject VM_i if budget exceeded}
- 13: **until** $v_j t_j \leq V_j$ **or** all VM types are rejected
- 14: **if** $v_j t_j > V_j$ **then**
- 15: **return** "Budget too low"
- 16: **end if**
- 17: **return** $(B_j = (0, \dots, r_i^j, \dots, 0, v_j), t_j)$

by setting r_i^j as $\sqrt{z}$ times overhead factor β_1^j (line 9). t_j is computed as the value required to satisfy $r_i^j t_j = z$ times the other overhead factor β_2^j , in an attempt to allocate some extra time for completion of the job (line 10). The valuation is simply the product of r_i^j and ρ_i^j (line 11). Lines 6 to 12 are enclosed in a repeat-until loop between lines 5 to 14 that changes the VM selection when the valuation-time product exceeds the budget. The algorithm returns “Budget too low” if no selection of VM type can satisfy the budget constraint.

B. Simulation Parameters

We perform two sets of simulation experiments. In the first set of experiments, each user takes part in two separate auctions. In the first auction, all users participate by generating their bids using the EBS algorithm while in the second auction, they all use the NB algorithm. A user bids in both auctions simultaneously until their application is completed or its deadline is exceeded. The parameters characterizing the application of a user are kept the same for both auctions. This set of experiments allows us to investigate the overall effect of the bidding strategies on the system.

In the second set of experiments, we run an auction with mixed user population. In each experiment, we set a distribution of naive and strategic users. Users are generated according to this distribution and all users take part in a single auction. These experiments allow us to investigate how effective our bidding strategy is on helping the users achieve better utilities. Also, they allow us to examine the effect of strategic bidders on the naive bidders and vice versa.

Each simulation experiment generates $N = 50,000$ users on a span of $D = 5$ simulated days. We run 24 auctions each day, one per hour. Users are generated each hour, at an average rate of $N/(24D)$ plus a randomly chosen deviation between $[-20\%, 20\%]$. The workload and the budget of a user are randomly selected from $[\omega^{min}, \omega^{max}]$ and $[V^{min}, V^{max}]$, respectively. Similarly, the parameters τ_j , c_j , λ_j , σ_1^j , σ_2^j , β_1^j , and β_2^j are randomly chosen from their respective minimum and maximum ranges presented in Table I. The bidding algorithms generate the bundles of VMs, their valuations, and the time required for application completion. The deadline of a task is determined by multiplying a deadline factor d_j with the time required, T_j , thus the deadline is $d_j T_j$. Here, d_j is chosen randomly from $[d^{min}, d^{max}]$.

We consider $m = 4$ types of VM instances in the system, each having k copies available, where $k = 200, 500$. We set the weight and fixed-price vectors of VM instance types to $\mathbf{w} = (1, 2, 4, 8)$ and $\mathbf{f} = (\$0.12, \$0.24, \$0.48, \$0.96)$. These are taken from the Windows Azure’s fixed-price VM allocation mechanism [13]. Finally, for the second set of experiments, the parameter π denotes the distribution of the two types of users. The distributions we use are (5%, 95%), (10%, 90%), (25%, 75%), and (50%, 50%),

Table I: Simulation Parameters

Parameter	Description	Value(s)
D	Simulated days	5
θ	Constant setup time for an application	0.01, 0.05
δ	Communication time between VMs	0.01, 0.02
m	Types of VMs	4
k	Available VMs of each type	200, 500
$\mathbf{w}$	Relative weight of VMs	(1, 2, 4, 8)
$\mathbf{f}$	Fixed price of VMs	(.12, .24, .48, .96)
N	Maximum number of users	50,000
n	Users participating in an auction	Varies
$\omega^{min}, \omega^{max}$	Min. & Max. workload	10, 50 20, 100
V^{min}, V^{max}	Min. & Max. user’s budget	5, 25 10, 50
d^{min}, d^{max}	Min. & Max. deadline factor	4, 8
τ^{min}, τ^{max}	Min. & Max. initialization overhead	1%, 5% 1%, 10%
c^{min}, c^{max}	Min. & Max. communication overhead	1%, 5% 1%, 10% 5%, 25%
$\lambda^{min}, \lambda^{max}$	Min. & Max. valuation for speedup	0.1, 0.25 0.1, 0.5
$\sigma_1^{min}, \sigma_1^{max}$	Min. & Max. valuation factor 1	0.75, 1.50 1.0, 2.0
$\sigma_2^{min}, \sigma_2^{max}$	Min. & Max. valuation factor 2	0.75, 1.50 1.0, 2.0
$\beta_1^{min}, \beta_1^{max}$	Min. & Max. overhead factor 1	90%, 95%
$\beta_2^{min}, \beta_2^{max}$	Min. & Max. overhead factor 2	105%, 125%
π	Distribution of users	(5%, 95%), (10%, 90%), (25%, 75%), (50%, 50%)

where (5%, 95%) means that there are 5% users who bid strategically using EBS and the rest are naive bidders using NB.

We also vary the system parameters θ and δ in different experiments. We list all the simulation parameters and their values in Table I. By varying the parameters we perform 1,536 experiments with combinatorial auctions with separate user population and 6,144 experiments with combinatorial auctions with mixed user population.

C. Analysis of Results

First we compare and analyze the outcomes of the experiments where each user generates a naive bid and a strategic bid for the same application and participates in two different auctions. In Figures 1a, 1b, and 1c, we show the percentage of users served, the average utility, and the total revenue generated. In these figures, the users whose applications are complete at the end of the simulation are called the ‘served users’. ‘Partially served users’ are those whose applications are only partially executed. The users whose applications did not even start fall into the category of ‘not served’ users. Finally, the users who did not bid because they exceeded their budget are included in the ‘did not bid’ group.

The utilities of the users are computed using Equation (10), where the valuation is determined using Equation (9), because this is the correct valuation of a requested

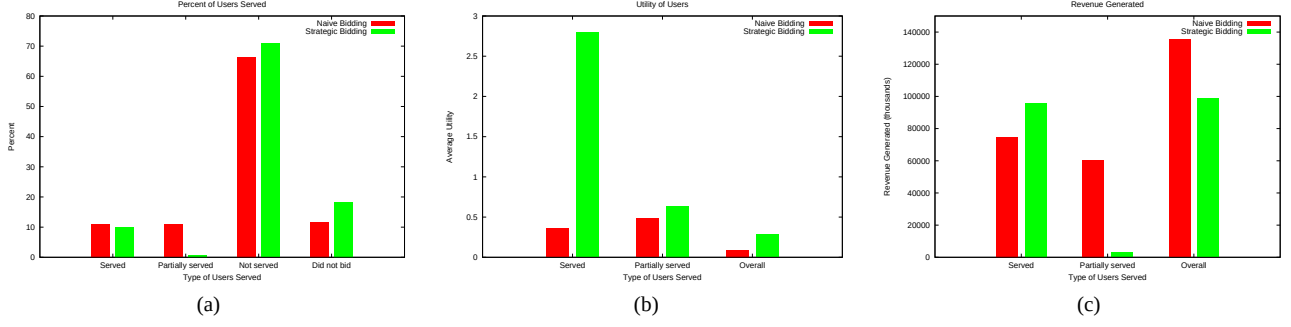


Figure 1: Separate auctions: overall results; (a) Users served; (b) Average utility; (c) Total revenue.

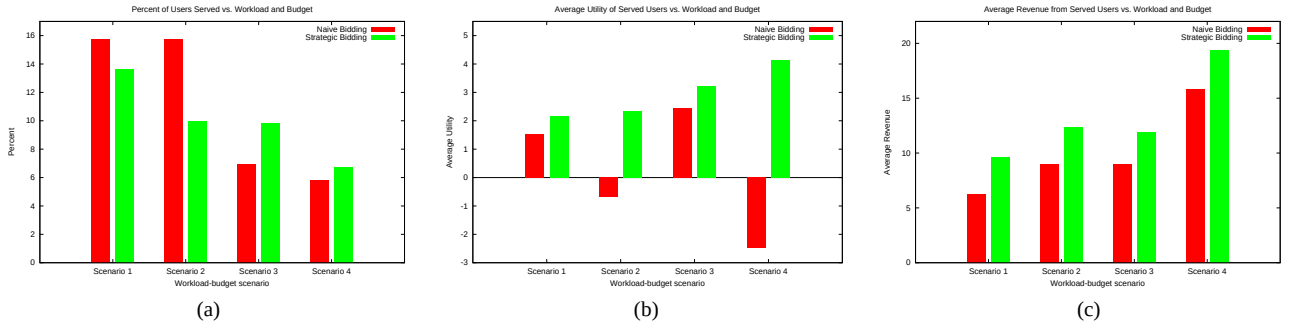


Figure 2: Separate auctions: outcomes vs. workload and budget; (a) Users served; (b) Average utility; (c) Total revenue. Each scenario represents a combination of $(\omega^{min}, \omega^{max}, V^{min}, V^{max})$ -values as follows: Scenario 1 $\equiv (10, 50, 5, 25)$, Scenario 2 $\equiv (10, 50, 10, 50)$, Scenario 3 $\equiv (20, 100, 5, 25)$, and Scenario 4 $\equiv (20, 100, 10, 50)$.

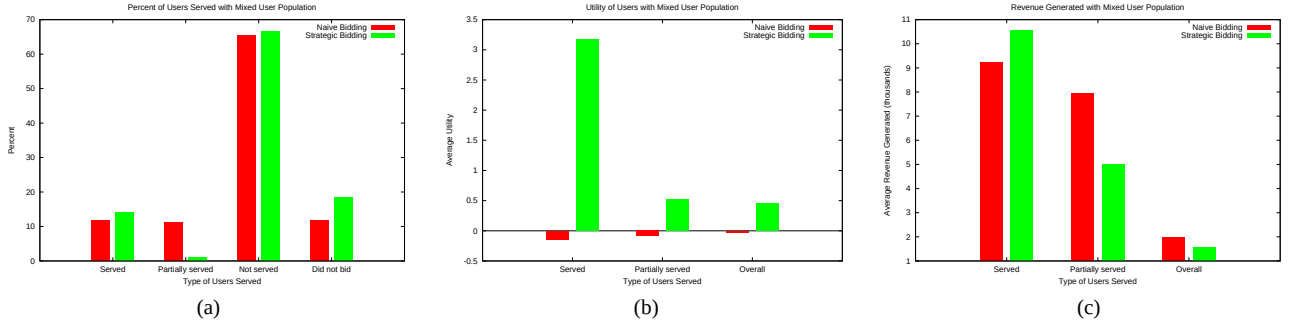


Figure 3: Auctions with mixed user population; (a) Users served; (b) Average utility; (c) Total revenue.

bundle irrespective of the method of generating the bundle. We see that although the number of served users are slightly larger when all users bid naively, the average utility of the served users is much larger when all bidders bid strategically. The reason behind having more users served while bidding naively is that each user selects one type of VM instances at random. Therefore, there is less competition among users as compared to the other auction (with strategic bidders), where users tend to bid for the largest VM instances if the budget permits. On the other hand, since naive users

are not guaranteed to bid their ‘true’ valuations, often a naive bidder ends up with a negative utility even when her application is completed. These factors reduce the average utility of naive users. We also see that (Figure 1c) although the total revenue is larger when bidders are naive, a large portion of it comes from the partially served users, which is not desirable to maintain user satisfaction. The amount of revenue generated when users bid strategically consists mainly from the revenue generated by the served users.

We now present some plots to show the auction outcomes

as the ranges of workload and budget vary. Figure 2 shows the percentage of served users, the average utility of served users, and the average revenue collected from the served users versus four ‘scenarios’ of workload and budget. The scenarios are different combination of the minimum and maximum workload and budget. For example, Scenario 1 represents the experiments where minimum and maximum workload are 10 and 50 and the budget ranges between 5 and 25. We see in Figure 2a that strategic bidding performs better than naive bidding in terms of the percentage of users served when the workload is higher. This illustrates that the probability of arbitrarily generating an efficient bid decreases as the workload increases. This is because the optimal number of processors and time have a non-linear relationship with the workload. Without this knowledge it is not possible to generate efficient bids for any workload. The percentage of users served decreases in Scenarios 3 and 4 because here users have larger workloads, while the amount of resources are the same in all four experimental scenarios. Also, Figures 2b and 2c show that when the workload and the budget increases, the strategic bidding can yield more utility for the users and more revenue for the cloud provider. Generating more utility for the users is a desirable property of a bidding strategy. Generating more revenue for the providers is a positive side-effect.

We now present the results from the second set of experiments, where both naive and strategic users participate in the same auction. The percentage of users served, the average utility and the revenue per user are shown in Figure 3. We see that in terms of served users, their utility and average revenue collected from them, the strategic bidders have better performance over the naive bidders. It is notable that in the presence of strategic bidders, the inefficiency of naive bidding is clearly seen as their average utility is negative. This is because a naive bidder is not guaranteed to bid for an optimal bundle with a true valuation. When all the participants are naive bidders, it is possible that some bidders’ valuations are better than others and they receive positive utility. On the other hand, when strategic bidders are also participating, even the ‘best’ naive bidders are not always capable of getting the job done along with receiving a positive utility. We can conclude that using the EBS Algorithm the users are able to generate efficient bids, and thus, ensure good completion times and higher utility for their applications. This also helps the auctioneer generate a fair amount of revenue while maintaining user satisfaction.

V. CONCLUSION

We investigated the problem of generating efficient bids in combinatorial auction-based VM allocation mechanisms in clouds. We designed an efficient bidding algorithm and performed extensive simulation experiments to investigate its properties. We believe that this research will encourage users and cloud providers to adopt combinatorial auction-

based mechanisms for allocating their VM instances. Future work include the deployment of combinatorial auction-based allocation mechanisms and the evaluation of our proposed efficient bidding strategy on an experimental cloud platform.

REFERENCES

- [1] 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.
- [2] X. Sui and H.-F. Leung, “An adaptive bidding strategy for combinatorial auction-based resource allocation in dynamic markets,” in *Proc. 11th Pacific Rim Intl. Conf. on Artificial Intelligence*, 2010, pp. 510–522.
- [3] S. Yi, D. Kondo, and A. Andrzejak, “Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud,” in *Proc. 3rd IEEE Intl. Conf. on Cloud Computing*, 2010, pp. 236–243.
- [4] Amazon, “Amazon EC2 spot instances.” [Online]. Available: <http://aws.amazon.com/ec2/spot-instances/>
- [5] N. An, W. Elmaghraby, and P. Keskinocak, “Bidding strategies and their impact on revenues in combinatorial auctions,” *J. Revenue and Pricing Manag.*, vol. 3, no. 4, pp. 337–357, 2005.
- [6] J. T. Havill and W. Mao, “Competitive online scheduling of perfectly malleable jobs with setup times,” *European J. of Oper. Res.*, vol. 187, no. 3, pp. 1126–1142, 2008.
- [7] T. E. Carroll and D. Grosu, “Incentive-compatible online scheduling of malleable parallel jobs with individual deadlines,” in *Proc. 39th Intl. Conf. on Parallel Processing*, 2010, pp. 418–425.
- [8] D. G. Feitelson, “Job scheduling in multiprogrammed parallel systems,” IBM, Research Report RC 19790 (87657), 1994.
- [9] M. Schwind, T. Stockheim, and O. Gujo, “Agent’s bidding strategies in a combinatorial auction controlled grid environment,” in *Proc. of AAMAS Agent Theories, Architectures, and Languages*, 2006, pp. 149–163.
- [10] F. Chang, J. Ren, and R. Viswanathan, “Optimal resource allocation in clouds,” in *Proc. 3rd IEEE Intl. Conf. on Cloud Computing*, 2010, pp. 418–425.
- [11] A.-M. Oprea and T. Kielmann, “Bag-of-tasks scheduling under budget constraints,” in *Proc. 2nd IEEE Intl. Conf. on Cloud Computing Technology and Science*, 2010, pp. 351–359.
- [12] W. Shi and B. Hong, “Resource allocation with a budget constraint for computing independent tasks in the cloud,” in *Proc. 2nd IEEE Intl. Conf. on Cloud Computing Technology and Science*, 2010, pp. 327–334.
- [13] Microsoft, “Windows Azure FAQ.” [Online]. Available: <http://www.microsoft.com/windowsazure/faq/>