Online Cloud Resource Allocation and Pricing with Server Speed Scaling

Ziyue Luo
SKLSE, School of Computer Science
Wuhan University
luozy@whu.edu.cn

Zongpeng Li
SKLSE, School of Computer Science
Wuhan University
zongpeng@whu.edu.cn

Chuan Wu
Dept. of Computer Science
The University of Hong Kong
cwu@cs.hku.hk

***Abstract*—The provisioning of cloud computing services typically incurs huge electricity costs. Utilization maximization of the cloud resources and efficient resource pricing have been key factors determining a cloud provider's revenue. On the other hand, dynamic CPU speed scaling has been widely supported by modern operating systems and hypervisors as an efficient technique for CPU energy saving, potentially useful for cutting down provider's electricity bill. In this paper, we propose an online mechanism for resource allocation and pricing on a cloud platform, which enables dynamic CPU speed scaling for achieving the best job execution efficiency. Using a novel compact infinite optimization technique and the primal-dual online algorithm design framework, our online mechanism achieves computational efficiency, truthfulness, and near-optimal social welfare during the long run of the cloud system. Trace-driven simulation studies further demonstrate good performance of our mechanism in realistic settings.**

I. Introduction

Cloud computing has been a popular service that grants users immediate access to abundant computing resources. Infrastructure-as-a-Service (IaaS), a major cloud service model provisioned by providers such as Amazon and Microsoft, advocates packing resources such as CPU, RAM and disk storage into virtual machines (VMs) or containers, for leasing to users. For example, Amazon EC2 provides customers with a wide range of VMs in various types [1]. Users can deploy their systems or process their jobs on these VMs/containers that meet their job requirements.

For operating a cloud platform, electricity cost accounts for a major part in the operational cost. On the other hand, power consumption by cloud computing has rendered a significant portion in the electricity market, from 1.3% of the global electricity supply in 2012 to the estimated 8% in 2020 [2]. CPU is one of the main energy-consuming components in servers, *e.g.*, 30% of the overall power is consumed by the CPU as in an Atom processor based server, and 60% in a Xeon-equipped server [3]. To alleviate CPU energy consumption, major chip manufacturers, including Intel and AMD, enable *dynamic speed scaling* in their CPUs: a processor's speed can be dynamically adjusted according to incoming workload. Supplementing hardware designs, most operating systems and hypervisors support dynamic speed scaling. Xen hypervisor includes a power management feature, CPU P-States, implemented by the *cpufreq* driver [4]. It periodically measures the system status and adjusts the CPU frequency according to the status for power consumption reduction. Such a dynamic speed scaling technique is suitable for cloud jobs such as human genome and other big data analysis with batch-processing nature: those jobs are delay tolerant to some extent; hence, we may alleviate electricity consumption of servers processing these job by slightly lowering CPU speed if needed.

This paper investigates efficient online algorithms for job scheduling on CPU resources in a geo-distributed IaaS cloud with dynamic speed scaling enabled on its servers. Especially, we design an online auction mechanism for CPU resource scheduling and pricing, aiming to maximize both the provider and users' utilities, by striking a good balance among user job valuation, electricity cost and deadline feasibility. Upon a user's arrival, it submits its CPU resource demand with the bidding price. The cloud provider immediately decides whether to accept the bid, schedules the job (if accepted) to run on processors in future time slots and decides the CPU speeds for running the job in respective time slots, as well as calculates the user's payment.

The online mechanism achieves several properties. *First*, the online mechanism runs in polynomial time to process each user's bid. *Second*, it achieves *truthfulness*, that a user must declare true job valuation to maximize its own utility. *Third*, it produces near-optimal job schedules that maximize the social welfare. We show that our online mechanism achieves a good *competitive ratio* (2.77 in typical scenarios), computed by dividing the optimal offline social welfare and the social welfare achieved by the online mechanism.

Our technical contributions to enable the efficient online mechanism are as follows. We design the auction framework and formulate the social welfare maximization problem into an integer non-linear program. Even without some constraints and variables, the problem is NP-hard. To tackle its NP hardness, we further formulate the original program into a *compact infinite* integer linear program (ILP). Even though this ILP contains an infinite number of variables, we novelly apply the primal-dual technique and design an efficient online auction mechanism.

The rest of the paper is organized as follow. We discuss related work in Sec. II, and introduce the system model in Sec. III. Sec. IV presents the online auction mechanism, Sec. V gives simulation results, and Sec. VI concludes the paper.

II. Related Work

A number of studies have investigated auction mechanism design for dynamic cloud resource provisioning and pricing in recent years. Zhang *et al.* [5] propose a one-round randomized auction approach that guarantees a low approximate ratio and truthfulness. Wang *et al.* [6] produce a two-dimensionally truthful online auction, in which users always hold allocated VMs for a continuous period of time. Zhang *et al.* [7] design an online auction with heterogeneous user demands. Both of the above work consider only a single type of VM for simplicity. Shi *et al.* [8] propose an online auction where the cloud provider provisions multiple types of VMs, and reassignment of already allocated resources is allowed. All these work do not take into account the operational cost of servers. A recent work by Zhang *et al.* [9] proposes an online auction mechanism, considering server costs in convex function forms. Different from all the literature, our work is the first one which combines VM allocation with dynamic server speed scaling, and allows elastic job execution for power cost saving.

Dynamic speed scaling has attracted substantial interest as an efficient technique to reduce power usage in electronic devices. Yao *et al.* [10] produce the first theoretical analysis of dynamic speed scaling on a single processor and design both an offline optimal scheduling algorithm and an online approximation algorithm. Bansal *et al.* [11] further prove the competitive ratio of the algorithm proposed by Yao *et al.* and design a new online algorithm that focuses on temperature management using speed scaling. Albers *et al.* [12] extend the study of dynamic speed scaling to multi-processor environments, considering either unit-size jobs or arbitrarily-sized jobs with deadlines. Differently, our work studies dynamic speed scaling in cloud VMs in an online cloud resource scheduling problem, targeting overall utility maximization of cloud provider and users.

Compact exponential optimization techniques have been proposed by Zhou *et al.* [13] for online cloud resource allocation. Sun *et al.* [14] further improve the technique and apply it to power scheduling for datacenter demand response. We propose a new *compact infinite optimization* technique based on their techniques, which is able to transform a non-linear integer program into an equivalent ILP at the price of introducing an infinite number of variables; nonetheless, we are still able to apply the primal-dual framework to derive an efficient online mechanism.

III. System Model

A. The Cloud System

Consider a cloud with J geo-distributed data centers. Let R_j denote the total processor capacity in data center $j \in [J]$,[1] which can be calculated as the total number of CPU cores provided on servers in the datacenter. Each CPU core can independently operate at a specified clock frequency. Without loss of generality, we assume every CPU core can be assigned to at most one job at any given time slot. Let C_j be the price of electricity that data center j consumes.

[1]We use $[X]$ to denote the integer set $\{1, 2, \ldots, X\}$.

The system works in a time-slotted fashion. The entire time span T is divided into T time slots and the length of each time slot is δ. I jobs arrive at the cloud system throughout T. Upon arrival, the owner of job $i \in [I]$ submits to the cloud provider the following: (1) b_i, the willingness-to-pay for completing its job in the cloud; (2) the arrival time t_i and the deadline d_i by when the job i should be completed; (3) w_i, the total number of required CPU cycles for job i; and (4) r_i, the number of CPU cores required by job i. That is, the bid of job request i can be expressed as:

$$Bid_i = \{b_i, t_i, d_i, w_i, r_i\}$$

Upon arrival of each job, the cloud provider decides whether to accept the job request immediately. If the job is accepted, it produces a resource allocation scheme together with job's payment. The job schedule is formulated by the following quantities. (1) Binary variable x_{ij} indicates whether job i is accepted and assigned to run in data center j ($x_{ij} = 1$), or not ($x_{ij} = 0$). (2) Binary variable $y_{ij}(t)$ denotes the executive schedule of bid i: $y_{ij}(t)$ equals 1 if job i is accepted to data center j and scheduled to run in time slot t, and 0, otherwise. (3) Variable $s_{ij}(t)$ denotes the speed of CPU cores allocated to job i in data center j at t. We assume that the speed is the same for all cores allocated to one job within one time slot, which may vary from one time slot to another. We use s_{min} and s_{max} to denote the minimum and maximum speed of a CPU core, respectively. Note that the execution time slots of one job are not necessarily continuous, *i.e.*, a job can be temporarily suspended at one time slot and resume in another. We also consider a transfer delay of dispatching job i to data center j, as indicated by a_{ij}. We allow each job to run in one data center only.

Power consumption of CPU is proportional to $V^2 f$, in which V denotes the supply voltage and the f is the clock frequency (speed) [15]. In practice, the supply voltage V is strongly related to the clock frequency, because operating at a high frequency must be accompanied with a high voltage. Therefore we can model the power consumed by a core, denoted by P, as a function of its speed, s. Consistent with much prior literature [11] [12] [16], we adopt the power function $s^\alpha, \alpha \geq 2$, and compute the power consumed by a single core during a time slot as $P(s) = \delta s^\alpha$.

Let v_i be user's truth value of job i, and p_i be the payment if the cloud provider accepts job i. Utility of job i is $u_i = v_i - p_i$ if bid i is accepted, and $u_i = 0$, otherwise. Let $e_j(t)$ be the total electricity consumption of CPU in time slot t at data center j. The cloud provider's utility can be computed as overall user payment minus the power cost, $\sum\limits_{i\in[I]} \sum\limits_{j\in[J]} p_i x_{ij} - \sum\limits_{j\in[J]} \sum\limits_{t\in[T]} C_j e_j(t)$.

B. Social Welfare Maximization Problem

We seek to maximize the sum of provider and users' utilities, *i.e.*, the social welfare. Assuming truthful bids, the overall utility of all users is $\sum\limits_{i\in[I]}\sum\limits_{j\in[J]} b_i x_{ij} - \sum\limits_{i\in[I]}\sum\limits_{j\in[J]} p_i x_{ij}$. Hence the social welfare is $\sum\limits_{i\in[I]}\sum\limits_{j\in[J]} b_i x_{ij} - \sum\limits_{j\in[J]}\sum\limits_{t\in[T]} C_j e_j(t)$ (payments are cancelled in the summation). To inspire online algorithm design, we first formulate the offline social welfare maximization problem:

$$\text{Maximize} \sum_{i\in[I]}\sum_{j\in[J]} b_i x_{ij} - \sum_{j\in[J]}\sum_{t\in[T]} C_j e_j(t) \tag{1}$$

Subject to:

$$\sum_{j\in[J]} x_{ij} \le 1, \forall i \in [I] \tag{1a}$$

$$y_{ij}(t)t \le d_i x_{ij}, \forall i \in [I], \forall j \in [J], \forall t \ge (t_i + a_{ij}) \tag{1b}$$

$$s_{ij}(t) \le s_{max} y_{ij}(t), \forall i \in [I], \forall j \in [J], \forall t \ge (t_i + a_{ij}) \tag{1c}$$

$$w_i x_{ij} \le \sum_{t\in[T]:t\ge t_i+a_{ij}} s_{ij}(t), \forall i \in [I], \forall j \in [J] \tag{1d}$$

$$\sum_{i\in[I]:t\ge t_i+a_{ij}} r_i y_{ij}(t) \le R_j, \forall j \in [J], \forall t \in [T] \tag{1e}$$

$$\sum_{i\in[I]:t\ge t_i+a_{ij}} r_i P(s_{ij}(t)) \le e_j(t), \forall j \in [J], \forall t \in [T] \tag{1f}$$

$$x_{ij}, y_{ij}(t) \in \{0,1\}, s_{ij}(t) \in [s_{min}, s_{max}] \bigcup \{0\}, e_j(t) \ge 0, \forall i \in [I], \forall j \in [J], \forall t \in [T] \tag{1g}$$

Constraint (1a) specifies that each job is accepted to at most one data center. Constraint (1b) guarantees that each job is executed between its arrival time and its deadline, while constraint (1d) ensures that the schedule is able to complete the corresponding job. Constraint (1c) makes sure that $y_{ij}(t)$ equals 1 only when $s_{ij}(t)$ is greater than 0. Moreover, constraint (1b), Constraint (1c) and constraint (1d) together ensure that a job is only executed in its assigned data center. Constraint (1e) expresses the processor capacity limitation in each data center. The total electricity consumption in time slot t at data center j is computed in constraint (1f).

If we set $P(\cdot) = 1$ and $s_{max} = s_{min} = 1$, problem (1) becomes an integer linear program (ILP). This ILP without constraints (1a) and (1b) is the same as the classic NP-hard knapsack problem. To design an efficient online mechanism without assuming knowledge of future incoming jobs, we adopt novel optimization techniques, to be detailed in the next section.

IV. Online Auction Mechanism Design

We first reformulate the non-linear problem in (1) into a *compact-infinite* integer program, based on which a primal-dual technique can be applied for online mechanism design. Let l denote a feasible resource allocation schedule for job i, which satisfies constraints (1b) and (1d). Schedule l is a vector such that $l = (\{t\}_{\forall t\in[T]:y_{ij}(t)=1}, \{s_{ij}(t)\}_{\forall t\in[T]})$. We use $l(t)$ to represent the processor speed at time slot t in schedule l, and use ζ_{ij} to denote the set of all feasible schedules of job i in data center j. Consequently, we formulate the *compact-infinite* integer program as follows:

TABLE I
NOTATION

Var	Definition
J	# of data centers
I	# of jobs
T	# of time slots
δ	length of a single time slot
R_j	capacity of CPU in data center j
C_j	electricity price in data center j
b_i	bidding price of job i
t_i	arrival time of job i
a_{ij}	estimated transfer delay for job i to data center j
w_i	required CPU cycles for job i
d_i	deadline of job i
r_i	number of CPU cores job i requires
p_i	payment of job i
x_{ij}	job i is accepted to run in data center j or not
$y_{ij}(t)$	job i is executed at time slot t in data center j or not
$s_{ij}(t)$	CPU speed when executing job i in data center j at t
$e_j(t)$	overall electricity consumption at time slot t in data center j
$P(s)$	power consumption when CPU speed is s

$$\text{Maximize} \sum_{i\in[I]}\sum_{j\in[J]}\sum_{l\in\zeta_{ij}} b_i x_{ijl} - \sum_{j\in[J]}\sum_{t\in[T]} C_j e_j(t) \tag{2}$$

Subject to:

$$\sum_{j\in[J]}\sum_{l\in\zeta_{ij}} x_{ijl} \le 1, \forall i \in [I] \tag{2a}$$

$$\sum_{i\in[I]}\sum_{l\in\zeta_{ij}:t\in l} r_i x_{ijl} \le R_j, \forall j \in [j], \forall t \in [T] \tag{2b}$$

$$\sum_{i\in[I]}\sum_{l\in\zeta_{ij}:t\in l} r_i P(l(t)) x_{ijl} \le e_j(t), \forall j \in [j], \forall t \in [T] \tag{2c}$$

$$x_{ijl} \in [0,1], e_j(t) \ge 0, \forall i \in [I], \forall j \in [J], \forall l \in \zeta_{ij} \tag{2d}$$

Constraints (2a), (2b) and (2c) are equivalent to constraints (1a), (1e) and (1f), respectively. Constraints (1b), (1c) and (1d) are transformed into schedule set ζ_{ij}. Therefore, we can observe that (2) produces the same solution as the original non-linear problem (1).

Problem (2) is a linear program, but it involves an infinite number of variables. Since we have a potentially infinite number of possible values for the speed $s_{ij}(t)$, the number of possible schedules l is infinite. Therefore, we have an infinite number of variables x_{ijl}. Nevertheless, our online algorithm designed based on problem (2) using a primal-dual technique runs efficiently in polynomial time, by evaluating only a polynomial number of possible schedules to find the optimal schedule.

Relaxing the integrality constraint on variable x_{ijl} to $x_{ijl} > 0$ and introducing dual variables u_i, $k_j(t)$ and $v_j(t)$ corresponding to constraints (2a), (2b) and (2c), respectively, we derive the dual of (2):

$$\text{Minimize} \sum_{i\in[I]} u_i + \sum_{j\in[J]} \sum_{t\in T} R_j k_j(t) \tag{3}$$

Subject to:

$$u_i \geq b_i - \sum_{t\in[T]:t\in l} r_i k_j(t) - \sum_{t\in[T]:t\in l} r_i P(l(t)) v_j(t), \quad \forall i \in [I], \forall j \in [J], \forall l \in \zeta_{ij} \tag{3a}$$

$$v_j(t) \leq C_j, \forall j \in [J], \forall t \in [T] \tag{3b}$$

$$u_i, k_j(t), v_j(t) \geq 0, \forall i \in [I], \forall j \in [J], \forall t \in [T] \tag{3c}$$

We next present the online auction mechanism designed based on the primal and dual problems in (2) and (3) (Sec. IV-A), and then analyze properties achieved by the mechanism (Sec. IV-B).

A. The Online Auction Mechanism

In the online setting, the cloud provider only knows information of a job at the time when it arrives. Upon a job's arrival, the cloud provider immediately decides whether to accept it and if so, to which data center the job is to be dispatched, as well as a feasible schedule l for running the job in the selected data center, by solving problem (2). If the job is assigned to a data center j, corresponding x_{ij} is set to 1, and we update variables $y_{ij}(t)$ and $s_{ij}(t)$ according to l. By the rule of complementary slackness [17], variable x_{ijl} equals 0 unless dual constraint (3a) is tight. Hence, we let dual variable u_i be the maximum of 0 and right hand side (RHS) of (3a):

$$u_i = \max_{j\in[J], l\in\zeta_{ij}} \{0, b_i - \sum_{t\in[T]:t\in l} r_i k_j(t) - \sum_{t\in[T]:t\in l} r_i P(l(t)) v_j(t)\} \tag{4}$$

Once the value of u_i is decided, variables j and l are those that maximize RHS of (3a). If $u_i > 0$, job i is accepted to run in data center j and executed according to schedule l.

In order to maximize the RHS of (3a), we shall now turn our attention to the other dual variable, $v_j(t)$. We interpret it as the unit price of electricity in data center j. By complementary slackness, if $e_j(t)$ is greater than 0, which means that there is power usage in time slot t at data center j, dual constraint (3b) must be tight, making $v_j(t)$ equal the electricity price C_j. If $e_j(t)$ equals 0, indicating that there is no power consumed then in j, then $v_j(t)$ can be any non-negative value and we set it to C_j. In summary, we let dual variable $v_j(t)$ always equal the electricity price C_j and $\sum_{t\in[T]} r_i P(l(t)) v_j(t)$ is the total electricity cost under schedule l.

Next, we interpret dual variable $k_j(t)$ as the marginal price per CPU core at t in data center j. In this way, $\sum_{t\in[T]:t\in l} r_i k_j(t)$ becomes the overall resource cost of running job i. Combined with previous analysis, we can regard RHS of (3a) as the bidding price minus the total cost if running job i in data center j under schedule l.

Algorithm 1 Online Auction Mechanism: A_{online}

```
Input: profile of jobs $\{b_i, t_i, d_i, w_i, r_i\}$, $\{a_{ij}\}, \forall i \in [I], j \in [J]$
1: Define function $k_j(z_j(t))$ according to (5);
2: Initialize $x_{ij} = 0, y_{ij}(t) = 0, s_{ij}(t) = 0, e_j(t) = 0, z_j(t) = 0, k_j(t) = k_j(0), \forall j \in [J], \forall i \in [I], \forall t \in [T]$;
3: for every arrival of new job $i$ do
4:     $(u_i, p_i, x_{ij}, \{y_{ij}(t)\}, \{s_{ij}(t)\}, \{z_j(t)\}, \{k_j(t)\}) = A_{schedule}(\{b_i, t_i, d_i, w_i, r_i\}, \{a_{ij}\}, \{z_j(t)\}, \{k_j(t)\})$;
5:     if $u_i > 0$ then
6:         Allocate the job according to $x_{ij}$ and schedule the job according to $y_{ij}(t)$ and $s_{ij}(t)$;
7:         Charge the user at payment $p_i$;
8:     else
9:         Reject job $i$;
10:    end if
11: end for
```

We now describe our online mechanism which computes the best potential schedule for each job i upon its arrival and decides whether to accept the job based that schedule. Our algorithm framework consists of two parts: $A_{schedule}$, the sub-algorithm that produces the optimal schedule if we are to accept job i, and A_{online}, which decides whether to accept the job based on the output of $A_{schedule}$.

Even though there seems to be an infinite number of schedule candidates to be evaluated, we design $A_{schedule}$ in Alg. 2 such that it only exams a polynomial number of feasible schedules and outputs the optimal one. The idea behind the algorithm is as follows. We consider all the data centers that a job may be assigned to. For each data center, the number of time slots job i takes to complete is between $\lceil w_i/s_{max} \rceil$ and $\min\{\lfloor w_i/s_{min} \rfloor, |S_{avail}|\}$. We further calculate the optimal speed given a fixed number of time slots for completing the job in lines 10-13. Data center j^* with schedule l^* that minimizes the overall cost of job i is the optimal schedule we want. Then we calculate RHS of (3a) as b_i minus the overall cost in line 17. If $RHS > 0$, we accept the job and assign it according to j^* and l^*; otherwise, the job request is rejected.

For updating dual variable $k_j(t)$, the marginal price per CPU core, we introduce variable $z_j(t)$ indicating the number of CPU cores that have already been allocated to prior jobs. As $z_j(t)$ increases, available processor resources become more precious and $k_j(t)$ increases. Therefore, we decide $k_j(t)$ as a function of $z_j(t)$ in (5). Here, U represents the maximum unit price of CPU and L represents the minimum unit price. E_j denotes maximum electricity cost per CPU core in data center j. Since the unit price of CPU cores should take both the marginal price due to user resource occupation and electricity cost into account, we can assume that $L > E_j$. Constant $\frac{1}{\mathcal{F}}$ represents the lower bound of resource usage over all jobs in the whole system and can be estimated using historical data. Therefore, $k_j(z(t))$ is initialized to $\frac{1}{2\mathcal{F}}(L - E_j)$ and eventually

Algorithm 2 Scheduling Algorithm for job i: $A_{schedule}$

```
Input: profile of job $i$ $\{b_i, t_i, d_i, w_i, r_i\}$, $\{a_{ij}\}$, $\{z_j(t)\}$,
    $\{k_j(t)\}$, $\forall j \in [J]$
Output: $u_i, p_i, x_{ij^*}, \{y_{ij^*}(t)\}, \{s_{ij^*}(t)\}, \{z_j(t)\}, \{k_j(t)\}$,
    $\forall j \in [J], t \in [d_i, t_i]$
 1: for all $j \in [J]$ do
 2:     $S_{avail} = \emptyset$;
 3:     for all $t \in [t_i + a_{ij}, d_i]$ do
 4:         if $z_j(t) + r_i \leq R_j$ then
 5:             Add $t$ to $S_{avail}$;
 6:         end if
 7:     end for
 8:     Sort time slots in $S_{avail}$ by $k_j(z_j(t))$ in non-decreasing
        order;
 9:     for $N = \lceil w_i/s_{max} \rceil$ to $\min\{|S_{avail}|, \lfloor w_i/s_{min} \rfloor\}$ do
10:         Select the first $N$ time slot $\{t'_1, t'_2, \ldots, t'_N\}$from
            $S_{avail}$;
11:         $s_{ij}(t) = \frac{w_i}{N}$;
12:         Save the corresponding executing time slots and
            running speed into $l$;
13:         $p_i = \sum_{n=1}^{N} r_i k_j(t'_n) + \sum_{n=1}^{N} r_i P(s_{ij}(t'_n))C_j$;
14:     end for
15: end for
16: Set $j^*$, $l^*$ as the arguments that minimize the value of $p_i$
    and set $p_i$ as the minimum value;
17: $u_i = b_i - p_i$;
18: if $u_i > 0$ then
19:     $x_{ij^*} = 1$;
20:     $y_{ij^*}(t) = 1, s_{ij^*}(t) = l^*(t), z_j(t) = z_j(t) + r_i, k_j(t) =$
        $k_j(z(t)), \forall t \in l^*$;
21: end if
22: return $u_i, p_i, x_{ij^*}, y_{ij^*}(t), s_{ij^*}, z_j(t), k_j(t)$;
```

reaches $U - E_j$ as $z_j(t)$ approaches R_j. In $A_{schedule}$, we ensure that if $z_j(t)$ was greater than R_j, we would not schedule the job in time slot t.

$$k_j(z_j(t)) = \frac{1}{2\mathcal{F}}(L - E_j)\left(\frac{U - E_j}{\frac{1}{2\mathcal{F}}(L - E_j)}\right)^{\frac{z_j(t)}{R_j}} \quad (5)$$

Where $U \leq \max\limits_{i\in[I]} \frac{b_i}{\frac{w_i}{s_{max}} r_i}$, $L \geq \min\limits_{i\in[I]} \frac{b_i}{\frac{w_i}{s_{min}} r_i}$, $E_j = C_j P(s_{max})$ and $\frac{1}{\mathcal{F}} \leq \frac{\sum_{j\in[J]} \sum_{t\in[T]} z_j(t)}{\sum_{j\in[J]} TR_j}$. Here, $\frac{b_i}{\frac{w_i}{s_{max}} r_i}$ is the maximum unit price per CPU core for job i and $\frac{b_i}{\frac{w_i}{s_{min}} r_i}$ is the minimum; $\frac{\sum_{j\in[J]} \sum_{t\in[T]} z_j(t)}{\sum_{j\in[J]} TR_j}$ calculates the resource usage in all data centers across the entire time span.

Our online mechanism A_{online} is given in Alg. 1. In the beginning, A_{online} defines function $k_j(z_j(t))$ and initializes all the variables in line 1-2. Then, it waits for incoming jobs. Upon the arrival of a job, A_{online} resorts to sub-algorithm $A_{schedule}$ for optimal scheduling and decides whether to accept the job in line 5-10. $A_{schedule}$ always produces the optimal schedule and updates remaining resource amount $z_j(t)$ and marginal price $k_j(t)$ if the job will be accepted.

B. Theoretical Analysis

We now analyze the properties achieved by our online mechanism. All missing proof can be found in our online technical report [18].

Theorem 1. *The CPU speed for running job i, $s_{ij}(t)$, as computed in every iteration in lines 9-15 in $A_{schedule}$, minimizes the overall cost (the sum of the total CPU cost and the total electricity cost), if job i is assigned to data center j.*

Theorem 2. *A_{online} in Alg. 1 computes a feasible solution to problems (1), (2) and (3) in polynomial time.*

Theorem 3. ***Alg.*** *A_{online} in Alg. 1 is truthful.*

Theorem 4. ***Alg.*** *A_{online} in Alg. 1 is 2β-competitive in social welfare, where $\beta = \ln \frac{U - E_{max}}{\frac{1}{2\mathcal{F}}(L - E_{max})}$ and $E_{max} = \max\limits_{j\in[J]} E_j$.*

If we assume that overall job demand for CPUs is high as compared to resource provisioning, then $\frac{1}{\mathcal{F}}$ is close to 1. When $\frac{U - E_{max}}{L - E_{max}}$ is 2, the competitive ratio is close to 2.77.

V. Performance evaluation

We evaluate our online auction mechanism A_{online} through trace-driven simulation studies. We use the Google Cluster Data [19], which contains abundant job information including the start time, execution duration and resource requirements (including CPU usage) of the jobs over a 7-hour period. We translate each job into a bid according to the data from the trace. We assume each time slot is 5 minutes long and the entire time span is 24 hours with 288 time slots. Processor's frequency is set to be within [2.20GHz, 3.00GHz] based on that of Intel Xeon processors [20]. Jobs' demand for CPU cores is normalized between $[0, 1]$. We derive the workload for each job from the trace. Job deadlines are generated randomly between its arrival time and the end of the system time span. Constant α in the power function is between 2 and 3 [15]. We set the number of data centers as 10.

We first examine the performance of A_{online} in terms of social welfare and competitive ratio. Further, we exam the difference between enabling dynamic speed scaling and not.

Fig. 1 illustrates the social welfare achieved by A_{online} with different numbers of jobs and values of α. We observe that there is an obvious upward trend in social welfare as the number of jobs increases. Meanwhile, the social welfare decreases as α shifts from 2 to 3. This can be explained that the electricity cost rises as α increases, yielding a reduction in social welfare. In A_{online}, unit price of resources is defined based on the value of $\frac{U - E_{max}}{L - E_{max}}$. Since the value of E_{max} is based on the maximum CPU frequency, we can evaluate the impact on social welfare under different values of $\frac{U - E_{max}}{L - E_{max}}$, as shown in Fig. 2. We observe a positive impact of the value of $\frac{U - E_{max}}{L - E_{max}}$ on the social welfare. That is because higher $\frac{U - E_{max}}{L - E_{max}}$ ensures larger bidding prices (for accepted jobs). Thus, these high value bids raise the overall social welfare.

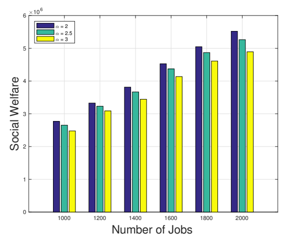

Fig. 1. Social Welfare: different numbers of jobs and α

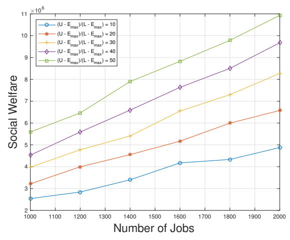

Fig. 2. Social Welfare: different numbers of jobs and $\frac{U-E_{max}}{L-E_{max}}$

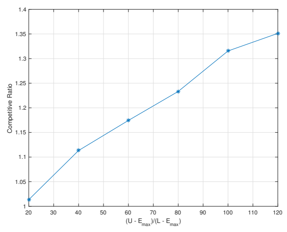

Fig. 3. Competitive ratio: different $\frac{U-E_{max}}{L-E_{max}}$ with 16 jobs and $\alpha = 2$

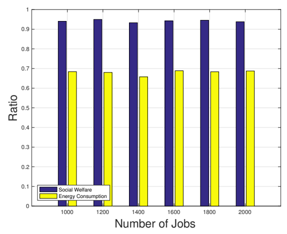

Fig. 4. Ratio in social welfare and electricity consumption: enabling dynamic speed scaling vs. not

Fig. 3 shows the ratio of the offline optimal social welfare obtained by directly solving problem (1) exactly through CPLEX function based on exploiting a branch and bound search tree and the overall social welfare obtained by A_{online}. The figure demonstrates that rise in $\frac{U-E_{max}}{L-E_{max}}$ leads to growth in the ratio. This is consistent with our theoretical analysis that the competitive ratio is related to the value of $\frac{U-E_{max}}{L-E_{max}}$. We observe that the ratio is much better than the theoretical bound. For example, the overall resource usage lower bound, $\frac{1}{\mathcal{F}}$, is at most 1, thus the theoretical bound of competitive ratio when $\frac{U-E_{max}}{L-E_{max}}$ equals 20 is 7.3778 while the observed ratio is less than 1.1. This is because the theoretical bound is calculated in worst cases which is uncommon in realistic scenarios.

Finally we investigate the difference made when dynamic speed scaling is enabled and not. Fig. 4 illustrates the ratio of the social welfare achieved (overall electricity consumed) when dynamic speed scaling is enabled and that when dynamic speed scaling is not enabled. In the case that dynamic speed scaling is not enabled, we set the CPU speed to 3.00GHz. We can see that applying dynamic speed scaling decreases social welfare by less than 10%, while significantly reducing the power consumption by more than 30%. Therefore, it shows that our algorithm efficiently decreases the power consumption at the price of relatively small fall in social welfare.

VI. Conclusion

This paper proposes an online mechanism for CPU resource scheduling and pricing in cloud systems with dynamic CPU speed scaling enabled. Leveraging a compact infinite optimization technique, we transform the non-linear integer problem for job scheduling into a *compact infinite* ILP. We resort to the primal-dual technique to derive the efficient online auction algorithm based on the ILP. Our complete mechanism comprises of a primal-dual online auction framework that decides the acceptance of incoming jobs and a sub-algorithm that produces the optimal CPU schedule for each job. Our online mechanism is efficient in computation and achieves truthfulness, with a near-optimal social welfare as compared to the offline optimum.

Acknowledgment

This work was supported in part by NSFC (61628209, 61571335) and HKU URC Matching Fund, Hubei Science Foundation (2016CFA030, 2017AAA125), and grants from Hong Kong RGC under the contracts HKU 17204715, 17225516, C7036-15G (CRF).

References

[1] *Amazon EC2 Instance Types*, https://aws.amazon.com/ec2/instance-types/.
[2] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav, "It's not easy being green," in *Proc. of the ACM SIGCOMM*, 2012.
[3] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.
[4] *Xen*, https://wiki.xenproject.org/wiki/Xen_power_management/.
[5] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: A randomized auction approach," in *Proc. of IEEE INFOCOM*, 2014.
[6] W. Wang, B. Liang, and B. Li, "Revenue maximization with dynamic auctions in iaas cloud markets," in *Proc. of IEEE IWQoS*, 2013.
[7] H. Zhang, B. Li, H. Jiang, F. Liu, A. V. Vasilakos, and J. Liu, "A framework for truthful online auctions in cloud computing with heterogeneous user demands," in *Proc. of IEEE INFOCOM*, 2013.
[8] W. Shi, C. Wu, and Z. Li, "Rsmoa: A revenue and social welfare maximizing online auction for dynamic cloud resource provisioning," in *Proc. of IEEE IWQoS*, 2014.
[9] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. M. Lau, "Online auctions in iaas clouds: Welfare and profit maximization with server costs," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1034–1047, Apr. 2017.
[10] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proc. of IEEE 36th Annual Foundations of Computer Science*, 1995.
[11] N. Bansaland, T. Kimbrel, and K. Pruhs, "Speed scaling to manage energy and temperature," *J. ACM*, vol. 54, no. 1, p. 3, Mar. 2007.
[12] S. Albers, F. Müller, and S. Schmelzer, "Speed scaling on parallel processors," in *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, 2007.
[13] R. Zhou, Z. Li, C. Wu, and Z. Huang, "An efficient cloud market mechanism for computing jobs with soft deadlines," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 793–805, Apr. 2017.
[14] Q. Sun, C. Wu, Z. Li, and S. Ren, "Colocation demand response: Joint online mechanisms for individual utility and social welfare maximization," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3978–3992, Dec. 2016.
[15] A. Wierman, L. L. H. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems," in *Proc. of IEEE INFOCOM*, 2009.
[16] L. Chen and N. Li, "On the interaction between load balancing and speed scaling," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2567–2578, Dec. 2015.
[17] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
[18] "Online cloud resource allocation and pricing with server speed scaling," Tech. Rep., https://www.dropbox.com/s/3840b1s8mr1mgi6/technical_report.pdf?dl=0.
[19] *Google Cluster Data*, https://github.com/google/cluster-data/.
[20] *Intel Xeon Processor E5 v4 Family*, http://ark.intel.com/products/series/91287/Intel-Xeon-Processor-E5-v4-Family/.