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.

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 $\delta$. $I$ 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 $s_{\text{min}}$ and $s_{\text{max}}$ 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 $\tau_{ij}$. We allow each job to run in one data center only.

Power consumption of CPU is proportional to $V^2f$, 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, $\sum_{i \in [I]} \sum_{j \in [J]} p_i x_{ij} - \sum_{j \in [J]} \sum_{t \in [T]} C_j e_j(t)$.

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 \( \sum_{i \in [I]} \sum_{j \in [J]} b_i x_{ij} - \sum_{j \in [J]} \sum_{t \in [T]} C_j e_j(t) \). Hence the social welfare is

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

(payments are cancelled in the summation). To inspire online algorithm design, we first formulate the offline social welfare maximization problem:

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

Subject to:

\[
\begin{align*}
\sum_{j \in [J]} x_{ij} &\leq 1, \forall i \in [I] \hspace{1cm} (1a) \\
y_{ij}(t) &\leq d_i x_{ij}, \forall i \in [I], \forall j \in [J], \forall t \geq (t_i + a_{ij}) \hspace{1cm} (1b) \\
s_{ij}(t) &\leq s_{\max} y_{ij}(t), \forall i \in [I], \forall j \in [J], \forall t \geq (t_i + a_{ij}) \hspace{1cm} (1c) \\
w_i x_{ij} &\leq \sum_{t \in [T]} s_{ij}(t), \forall i \in [I], \forall j \in [J] \hspace{1cm} (1d) \\
r_{ij} y_{ij}(t) &\leq R_j, \forall j \in [J], \forall t \in [T] \hspace{1cm} (1e) \\
r_{ij} P(s_{ij}(t)) &\leq e_j(t), \forall j \in [J], \forall t \in [T] \hspace{1cm} (1f)
\end{align*}
\]

Constraint (1a) specifies that each job is accepted to run in at most one data center. Constraint (1b) guarantees that each job is executed between its arrival time and 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 \( l = \{ l(t) \}_{t \in [T] : y_{ij}(t) = 1} s_{ij}(t) \}_{t \in [T]} \). We use \( l(t) \) to represent the processor speed at time slot \( t \) in schedule \( l \), and use \( \zeta_{ij} \) 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:

Maximize \( \sum_{i \in [I]} \sum_{j \in [J]} b_i x_{ij} - \sum_{j \in [J]} \sum_{t \in [T]} C_j e_j(t) \) \hspace{1cm} (2)

Subject to:

\[
\begin{align*}
\sum_{j \in [J]} x_{ij} &\leq 1, \forall i \in [I] \hspace{1cm} (2a) \\
\sum_{t \in [T]} l(t) x_{ij} &\leq R_j, \forall j \in [J], \forall t \in [T] \hspace{1cm} (2b) \\
\sum_{t \in [T]} r_{ij} P(l(t)) x_{ij} &\leq e_j(t), \forall j \in [J], \forall t \in [T] \hspace{1cm} (2c)
\end{align*}
\]

Constraints (2a), (2b) and (2c) are equivalent to constraints (1a), (1e) and (1f), respectively. Constraints (1b), (1c) and (1d) are transformed into schedule set \( \zeta_{ij} \). 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):

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

Subject to:

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

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

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

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 $x_{ijl}$ is set to 1, and we update variables $y_{ijl}(t)$ and $s_{ijl}(t)$ 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):

$$u_i = \max_{j \in [J], t \in C_{ij}} \{0, b_i - \sum_{t' \in [T]: t' \in T} r_i k_j(t') - \sum_{t' \in [T]: t' \in T} r_i P(l(t)) v_j(t')\} \quad (4)$$

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, $\sum_{t \in [T]: t \in T} r_i k_j(t)$ 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 $l$.

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 examas 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 $[w_i/s_{max}]$ and $\min([w_i/s_{min}], |S_{available}|)$. 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 $\frac{1}{k}$ 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 $\frac{1}{\frac{1}{k}(L - E_j)}$ 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. $S_{avai} = \emptyset$;  
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 $S_{avai}$;  
6. end if  
7. end for  
8. Sort time slots in $S_{avai}$ by $k_j(z_j(t))$ in non-decreasing order;  
9. for $N = \lfloor w_i/s_{max}\rfloor$ do  
10. Select the first $N$ time slot $\{t_1', t_2', \ldots, t_N'\}$ from $S_{avai}$;  
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. $y_{ij^*}(t) = 1$, $s_{ij^*}(t) = l^*(t)$, $z_j(t) = z_j(t) + r_i$, $k_j(t) = k_j(z_j(t))$, $\forall t \in l'$;  
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 $t$.  

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

Where $U \leq \max \left\{ \frac{b_i}{\min r_i} \right\}$, $L \geq \min \left\{ \frac{b_i}{\min r_i} \right\}$, $E_j = C_j P(s_{max})$ and $\frac{1}{F} \leq \sum_{j \in [J]} \sum_{t \in [l]} z_j(t)$ TR$_j$. Here, $\frac{b_i}{\min r_i}$ is the maximum unit price per CPU core for job $i$ and $\frac{b_i}{\min r_i}$ is the minimum;  

$$\sum_{j \in [J]} \sum_{t \in [l]} z_j(t)$$

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 $2\beta$-competitive in social welfare, where $\beta = \ln \frac{U - E_{max}}{L - E_{max}}$ and $E_{max} = \max_{j \in [J]} E_j$.

If we assume that overall job demand for CPUs is high as compared to resource provisioning, then $\frac{1}{Z}$ 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 $\alpha$ 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 examine 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 $\alpha$. We observe that there is an obvious upward trend in social welfare as the number of jobs increases. Meanwhile, the social welfare decreases as $\alpha$ shifts from 2 to 3. This can be explained that the electricity cost rises as $\alpha$ 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.
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, 2017AA1A25), and grants from Hong Kong RGC under the contracts HKU 17204715, 17225516, C7036-15G (CRF).

**REFERENCES**


