Occupation-Oblivious Pricing of Cloud Jobs via Online Learning

Xiaoxi Zhang[†], Chuan Wu[†], Zhiyi Huang[†], Zongpeng Li[§]
[†]Department of Computer Science, The University of Hong Kong, Email: {xxzhang2,cwu,zhiyi}@cs.hku.hk
[§]Department of Computer Science, University of Calgary, Canada, Email: zongpeng@ucalgary.ca

***Abstract*—State-of-the-art cloud platforms adopt pay-as-you-go pricing, where users pay for the resources on demand according to occupation time. Simple and intuitive as it is, such a pricing scheme is a mismatch for new workloads today such as large-scale machine learning, whose completion time is hard to estimate beforehand. To supplement existing cloud pricing schemes, we propose an occupation-oblivious online pricing mechanism for cloud jobs without pre-specified time duration and for users who prefer a pre-determined cost for job execution. Our strategy posts unit resource prices upon user arrival and decides a fixed charge for completing the user's job, without the need to know how long the job is to occupy the requested resources. At the core of our design is a novel multi-armed bandit based online learning algorithm for estimating unknown input by exploration and exploitation of past resource sales, and deciding resource prices to maximize profit of the cloud provider in an online setting. Our online learning algorithm achieves a low regret sub-linear with the time horizon, in terms of overall provider profit, compared with an omniscient benchmark. We also conduct trace-driven simulations to verify efficacy of the algorithm in real-world settings.**

I. INTRODUCTION

Cloud computing has revolutionized industries by democratizing data centers and enabling easy resource access for everyone. Lying at the core of a cloud business model is its resource pricing scheme. State-of-the-art cloud platforms adopt various pay-as-you-go pricing options. Google Cloud [1] and Microsoft Azure [2] employ per-minute billing to charge various types of virtual machine (VM) instances. Amazon Web Services (AWS) [3] adopt hourly charges for on-demand, Spot or the new burstable instances [4], and joint upfront and hourly payment for reserved instances during their terms. Different pricing plans are recommended for users with different workloads, budgets, and service requirements. For example, users with steady, persistent resource usage may prefer reserved instances, and users with transient workload would choose on-demand instances, or opt for spot or burstable instances if preemption is tolerable. However, a key question remains:

Are these pricing models suitable for jobs with spiky workload, non-preemption requirement, and uncertain completion time, such as training jobs in machine learning?

Nowadays large-scale machine learning (*e.g.*, deep learning [5][6][7]) has been widely adopted in the industry for processing big datasets and learning prediction/inference models. Many cloud platforms have started to provision tailored products to support machine learning (ML) jobs, exemplified by the new virtual GPU servers or GPU instances. Amazon EC2 announced the new P2 instances in September 2016 [8], and Google enabled GPUs on Google cloud in February 2017 [9]. These GPU instances are purchased in a similar way as other types of VMs at a per-hour or minute rate [3][2], or by adding the price per GPU per unit time [6].

Computation-intensive jobs that need to be run on GPU servers (*e.g.*, ML training jobs) often take hours or days to complete, and the job owner is typically uncertain of the time it may take. It can be quite costly for running an ML training job on existing cloud platforms under such uncertainties. For example, it takes about 4.8 days to train a convolutional neural network on a server with 4 NVIDIA TITAN GPUs and 6GB of RAM [10], and would cost more than $103 by using a p2.xlarge instance from Amazon EC2 [3]. For such jobs, preemption is less than favorable, since it further prolongs their completion. Consequently, spot or burstable instances, often adopted for cost saving, are inadequate. For such jobs, it is desirable to let users know a one-off price for job completion, in order to assure users that the costs are within their budgets.

The history of modern economics suggests that customers often prefer price certainty upon committing to a transaction, especially when on a budget. Examples can be found in driving schools where a flat-fee tuition is available for training until passing the road test, and in the telecom market where a flat monthly fee covers voice and data communication regardless of usage patterns. In doing so, the service provider moves the onus of price risk management away from their customers to themselves, who are both more *capable* (with more informed decision making based on historic job execution of similar types) and *inclined* to do the job (for customer satisfaction and retention in the growingly more fierce cloud market).

To supplement the current cloud charging models, we propose a novel occupation-oblivious pricing option: the cloud provider charges a lump-sum price for completing a user's ML job, depending on the amount of resources the job requires, but not the usage time of those resources. Especially, our pricing mechanism is suitable for jobs (i) that are intolerable to preemption during execution and (ii) for which the job owner does not have a good knowledge of the runtime needed in advance, but (iii) has a budget and hence prefers to know a pre-determined cost for completing the job.

Our pricing scheme works in an online fashion: when a user arrives and submits a job request (specifying job details and resource demand, but not the usage duration or budget), the cloud provider informs the user of unit resource prices on

the spot, and confirms a total charge for completing its job, regardless of how long the job will run till completion. The user decides if the charge is acceptable according to its budget, and runs the job in the cloud if accepted.

The unit price of each resource may vary upon arrivals of different users, carefully computed using an online algorithm based on resource availability, to achieve cloud provider profit maximization over the long run. Maximizing provider's profit is more desirable in practice, but in general harder than social welfare maximization in online algorithm design. Without assuming knowledge of user budgets and job runtimes, the cloud provider's profit maximization problem cannot be formulated as a standard online optimization problem, *e.g.*, an integer linear program (ILP) as in most existing work on social welfare maximization [11][12].

The critical challenge is how to strategically decide the occupation-oblivious resource prices, such that more jobs with high budgets (tolerant to higher prices), and reasonable resource occupation durations (adequate to the budgets) will be accepted. Our answer to this call is to employ a novel multi-armed bandit (MAB) based online learning strategy, to meticulously estimate an expected budget for jobs using the same type of resources and the inverse of job duration in expectation, assuming an unknown underlying distribution of the run times and budgets. In classical MAB problems, decisions are not related to those in the future through global constraints, while in our problem decisions at different times are tightly coupled due to resource constraints. In addition, the feedback of an online pricing decision (*i.e.*, job's actual runtime) is not known upon decision making, but until the job ends; such delayed feedback heavily undermines efficiency of an online learning method, which always assumes receiving feedback immediately when a decision is made. To address these challenges, we further combine the MAB based online learning with an online algorithmic framework, which together produce resource prices over time. Our detailed technical contributions are as follows.

▷ We carefully derive a price-profit relationship to approximate the expected total profit achievable with each candidate set of resource prices, and design reward functions in the MAB method based on this relationship. Different from existing MAB algorithms where the reward in each round is only relevant to the current decision, our reward functions respect global resource capacity constraints and estimate the expected overall profit, and fit our MAB online learning nicely within an online algorithm framework.

▷ We borrow the high-level idea of exploration and exploitation trade-off from existing MAB algorithms, but apply it in a new way. We divide our online algorithm into two stages: exploration at the beginning, when we set nil prices and accept all jobs, to cumulate inputs for run time and budget learning; exploitation after the exploration phase, when we decide best resource prices maximizing estimated overall profit upon job arrivals, using job run time and budget learned from all the history. Duration of the exploration phase is carefully set by trading off between profit lost due to zero prices and estimation error in the exploitation phase due to lack of job samples.

▷ We compare our pricing mechanism with an offline strategy which adopts the best fixed resource prices computed using full knowledge of the system, and rigorously prove a regret bound for our algorithm, which is sublinear with system timespan, resource capacities and total number of users.

II. Related Work

Dynamic pricing for cloud resources has been studied in recent years, following the initiative of Amazon EC2 Spot Instances [13]. Many studies propose offline or online auction mechanisms for allocating and pricing cloud resources, targeting social welfare or profit maximization [11][14]. In these auctions, users submit resource demand, usage time slots, and valuation of their jobs. Some studies focus on posted-price mechanisms [11][15], where users arrive over time and a take-it-or-leave-it price is offered to each user for the asked resources, and a user accepts the price if its true valuation is higher than the price. Zhang *et al.* [11] design price functions to update posted prices according to resource utilization levels. Zhang *et al.* [15] further prove optimal forms of the price update functions. These mechanisms require users to submit resource usage durations. Our mechanism is essentially a posted-price scheme, while we do not require knowledge of job durations for deciding resource prices, but still guarantee completion of a submitted job.

Multi-armed bandit (MAB) optimization is an effective online learning and optimization framework with partial feedback [16]. In an MAB problem, an agent faces a set of arms (actions) and selects one arm in each round. After pulling an arm, the agent receives a reward (or incurs a loss) on the pulled arm (but not on all arms, a.k.a partial feedback), which is a realization of some unknown, underlying reward (or loss) distribution associated with that arm. The goal is to obtain the maximal cumulative reward (or incur the minimal cumulative loss) over multiple rounds. MAB algorithm design pursues a good trade-off between *exploration* and *exploitation*, *i.e.*, to try less attempted arms to collect more feedback (exploration), or stick to the arm that brings the highest reward (or lowest loss) based on already learned information (exploitation). The performance of a bandit algorithm is commonly evaluated by regret, which is the gap between the overall reward obtained (or loss incurred) by an offline omniscient strategy and the expected reward (or loss) of the bandit algorithm.

MAB optimization has been applied to ad-display optimization [17] and scheduling in multichannel wireless networks [18]. In conventional settings of MAB problems, decision variables in different rounds are related to each other only through continuous learning of the unknown distributions. In our problem, cloud resources are shared by jobs arriving at different times and hence our online decisions made upon arrivals of different jobs are restricted by global resource capacity constraints. Moreover, feedback (resource occupation time of each job) is not received until job completion.

Non-clairvoyant job scheduling. Job scheduling without knowing the job size (*i.e.*, runtime) has been studied in the

theory community [19][20]. The aim is to design competitive algorithms for minimizing the sum of completion times of all jobs, or minimizing the maximum completion time among all jobs, which is different from our goal. Moreover, their online algorithms assume extra resources available in online scheduling while such extra resources are not used in computing the offline optimum, in order to prove a good competitive ratio, which is not practical in our setting.

III. Problem Model

We consider a cloud platform offering computational resources in the form of various preconfigured virtual instances. A virtual instance can represent the following: (i) a virtual machine (VM) or a virtual server equivalently, such as the typical VMs provided on Amazon EC2 [3], Google Cloud [1], or Microsoft Azure [2]; or (ii) a virtual cluster, consisting of multiple VMs. Representative examples of virtual clusters are the scale tiers in Google cloud machine learning (ML) engine [6]. A scale tier consists of a number of ML training units. A training unit can be a worker or a parameter server in the ML framework [21] logically; physically it is a combination of computing resources such as GPU, CPU, and memory. For example, a `BASIC` scale tier consists of one ML training unit (*i.e.*, one worker) and a `PREMIUM_1` scale tier consists of 75 ML training units (*i.e.*, many workers and parameter servers) [6]. Users can choose among the scale tiers to run their ML training jobs on the Google Cloud.

Suppose there are K types of virtual instances in total. The maximum number of available type-k instances is C_k, $\forall k \in 1, \cdots, K$. The system works in a time slotted fashion, with a total timespan of T time slots. n users arrive at different times during $1, \ldots, T$. Each user i requests one or multiple virtual instances of type k_i to run his job, *e.g.,* to train a deep learning model with a few GPU virtual servers of the same configuration, or a chosen scale tier. Let d_{ik} denote the number of type-k virtual instances that user i requests, such that $d_{ik} \in (0, C_k]$ if $k = k_i$, and $d_{ik} = 0$, otherwise. User i has a budget v_i for completing his job. Throughout the paper, we refer to jobs requesting type-k virtual instances as *type-k jobs*, for ease of presentation.

Upon the arrival of user i, he submits to the cloud the virtual instance type k_i he wants and the demand d_{ik_i}. The budget is private information of user i and will not be revealed to the cloud provider. After receiving the user request, the cloud provider informs user i of the current price p_{ik_i} for each type-k_i virtual instance. User i accepts the deal and runs his job in the cloud if and only if his overall payment, $p_{ik_i} \times d_{ik_i}$, is no larger than budget v_i; otherwise, the user can leave without purchasing the resources. Let y_k^i denote the total number of type-k virtual instances that have been allocated and are still occupied at the time of arrival of job i. We use an indication function to denote whether user i would accept the deal: $\mathbb{1}\{d_{ik_i}+y_{k_i}^i \le C_{k_i}, p_{ik_i}d_{ik_i} \le v_i\}$ equals 1 if $d_{ik_i}+y_{k_i}^i \le C_{k_i}$ and $p_{ik_i}d_{ik_i} \le v_i$, and 0, otherwise. Once a user has made a payment and started to run his job, the cloud provider ensures that the allocated resources will not be preempted and the job will run to completion.[1] Let τ_i denote the runtime of job i, which is not known by the cloud provider until the job is done.

The cloud provider targets overall profit maximization over the system timespan through online resource pricing. The offline optimization problem can be formulated as follows. Key notation is summarized in Table I, for ease of reference.

$$\max_{\mathbf{p}} \sum_{i=1}^{n}\sum_{k=1}^{K} p_{ik}d_{ik}\mathbb{1}\{d_{ik}+y_k^i \le C_k, \sum_{k=1}^{K} p_{ik}d_{ik} \le v_i, \forall k\} \quad (1)$$

subject to:

$$y_k^i = \sum_{\substack{j\in\{1,\cdots,i-1\}:\\ t_j+\tau_j \ge t_i}} d_{jk}\mathbb{1}\{d_{jk}+y_k^j \le C_k, \sum_{k=1}^{K} p_{jk}d_{jk} \le v_j, \forall k\},$$

$$\forall k \in [1,K], i \in [1,n] \ (1a)$$

$$p_{ik} \ge 0, \ \forall k \in [1,K], i \in [1,n] \ (1b)$$

The objective function maximizes total profit obtained from users that run their jobs in the cloud. (1a) computes the number of type-k virtual instances occupied upon arrival of user i.

Using problem (1) as a reference, we design an online pricing scheme for the cloud provider to decide price p_{ik_i} upon arrival of each user i, without knowing v_i nor τ_i. We make the following stochastic model assumptions for algorithm design and analysis, which are nonetheless not followed in our empirical studies of the algorithm performance in Sec. VI. For each type of virtual instance k, the (budget, demand) pairs of type-k user jobs, *i.e.*, (v_i, d_{ik}), are i.i.d. (independently and identically distributed), drawn from an underlying unknown distribution F_k. This essentially tells that budgets and demands of jobs requiring the same type of virtual instances follow a joint unknown static distribution, which can generalize budgets and demands of these jobs each of which follows an unknown static distribution. For each user i, the arrival time t_i falls uniformly randomly within the system timespan $[1, T]$.[2] Let $U(T)$ denote the discrete uniform distribution with random variable generated from interval $[1, T]$. We have $t_i \sim U[T]$. Users are indexed according to their order of arrivals in any fixed realization of the arrival process. The runtimes of all type-k jobs are i.i.d., drawn from an unknown static distribution $\mathcal{T}_k$, with an expectation of $\bar{\tau}_k$.

IV. Pricing Mechanism for Profit Maximization

A. Design Rationale

To set prices for profit maximization, the key is for the cloud provider to estimate how likely a user may accept the offered price and the runtime of user job, such that it can set the best price that a user would accept, which is also adequate for the duration of user resource occupation. We adopt an online learning approach: the cloud provider estimates distributions of user budgets, demands and job runtimes over time, making

[1]There are special cases that a job does not automatically stop, *e.g.*, due to program bugs. It is reasonable to assume that normal users will check out status of their jobs from time to time and kill a job if a mistake is found.

[2]This assumption holds true in common stochastic arrival processes. For example with a Poisson process, the total number of arrivals within a period of time is drawn from the Poisson Distribution and each arrival time is uniformly sampled from the corresponding timespan.

TABLE I: Notation

n	total # of users/jobs	n_k	# of type-k jobs
τ_i	duration of job i	$\mathcal{T}$	distribution of τ_i
$\bar{\tau}$	$= E_{\tau_i \sim \mathcal{T}}[\tau_i]$	T	# of time slots
K	# of virtual instance types	v_i	budget of job i
$\mathcal{F}_k$	distribution of (v_i, d_{ik})	t_i	arrival time of i

v_k^{max}	upper-bound of per-type-k-instance budget
k_i	job i's required type of virtual instances
d_{ik}	demand of type-k virtual instance of job i
C_k	total # of type-k virtual instances
p_{ik}	unit price of type-k virtual instances to job i
$D_k(p_k)$	expected # of type-k instances sold at p_k
y_k^i	# of type-k instances occupied upon job i
$Rev(\vec{p})$	expected total profit under a fixed price vector $\vec{p}$
$\hat{R}_{ik}(p_k)$	reward of choosing price p_k for job i
η_i	equals $\frac{1}{\tau_i}$
$\bar{\eta}_k$	the expectation of η_i for jobs with $k_i = k$
η_{ik}^U	UCB of $\bar{\eta}_k$ before job i where $k_i = k$
$D_{ik}^U(p_k)$	UCB of $D_k(p_k)$ before job i where $k_i = k$

use of price acceptance/rejection by users and runtimes of completed jobs in the past, and decides prices to new jobs based on learned distributions.

We discretize all possible prices of each type of instances and let the cloud provider choose from a set of prices $\mathcal{P}_k$ in our algorithm (note that the optimum prices we compare our algorithm with in the analysis may not fall into the discretized price set). To set a price which contributes to profit maximization over the long run, we analyze an upper-bound of the overall profit, and choose a price that maximizes the upper-bound. Suppose the same price $p_k \in \mathcal{P}_k$ is offered to all type-k jobs that arrive in $1, \ldots, T$, which can be considered as the expectation of realized prices to type-k jobs. Let $D_k(p_k)$ denote the expected number of type-k virtual instances sold at price p_k to any user i who requests type-k instances. That is, $D_k(p_k) = E_{(v_i, d_{ik}) \sim F_k}[\hat{d}_{ik}]$, where random variable $\hat{d}_{ik}$ equals d_{ik} if $v_i \geq p_k d_{ik}$, and zero, otherwise. Let n_k denote the total number of type-k users in $[1, T]$. Intuitively, if the total number of type-k virtual instances is sufficient to serve demands of all these n_k jobs, expected total profit under fixed price p_k should be $p_k n_k D_k(p_k)$. However, when considering the resource capacity limitation, at most C_k type-k virtual instances can be allocated at any time; each instance may be occupied by a job for multiple time slots, and can be re-allocated to a new job if that job has been completed at the arrival time of the new job. Recall that user arrival times are uniformly distributed in $[1, T]$. Since T is much larger than τ_i, the probability that for any given user, the probability that $t \in [T]$ is his arrival time can be approximated as $\frac{1}{\tau_i}$. Hence, at each time t, if type-k resource is exhausted, the maximum expected number of instances that are released and reused by new arrivals is $C_k E_{i:k_i=k}[\frac{1}{\tau_i}]$. Let $\eta_i = \frac{1}{\tau_i}$, and $\bar{\eta}_k = E_{i:k_i=k}[\eta_i]$. Thus, the maximum number of type-k jobs that the system can take by exhausting all type-k virtual instances is $TC_k\bar{\eta}_k$ in expectation. Therefore, the maximum overall profit collected at price p_k is $p_k TC_k\bar{\eta}_k$ in expectation.

Let $Rev(\vec{p})$ denote the expected overall profit under price vector $\vec{p} = \{p_1, \ldots, p_K\}$, where p_k is the fixed unit price for all type-k virtual instances. $Rev^k(p_k)$ is the expected profit obtained from type-k jobs at p_k. We have

$$Rev(\vec{p}) = \sum_{k=1}^{K} Rev^k(p_k) \leq \sum_{k=1}^{K} \min\{p_k TC_k\bar{\eta}_k, p_k n_k D_k(p_k)\} \tag{2}$$

The above inequality provides an upper-bound on the expected overall profit achieved by using a fixed price for each type of virtual instances throughout $[1, T]$, which we refer to as a *price-profit relationship*. Over time, we seek to estimate a best (expected) unit price p_k for each type of virtual instances, $\forall k \in [1, K]$, which maximizes the upper-bound in the right-hand side (RHS) of (2), in order to boost the overall profit $Rev(\vec{p})$. However, since the distributions of user budget, demand and job runtime are unknown in the online setting, we seek prices that maximize an estimate of the upper-bound instead.

We employ a multi-armed bandit (MAB) based online learning method for estimating those unknown distributions, and decide prices that maximize the estimated upper-bound. We prepare a set of price candidates (arms) for each type k of virtual instances, and select a price p_k from the set with the highest reward computed, upon arrival of each type-k job. The reward for choosing price p_k for a type-k job i, denoted by $\hat{R}_{ik}(p_k)$, is an estimate of the upper-bound of expected profit in RHS of (2):

$$\hat{R}_{ik}(p_k) = p_k \cdot \min\{TC_k\eta_{ik}^U, n_k D_{ik}^U(p_k)\} \tag{3}$$

Here, η_{ik}^U is the Upper Confidence Bound (UCB) of the expected runtime $\bar{\tau}_k$ estimated before the arrival of job i. Similarly, $D_{ik}^U(p_k)$ is the UCB of the expected number of type-k instances sold to a type-k user at p_k, $D_k(p_k)$, estimated before arrival of job i. We have

$$\eta_{ik}^U = \hat{\eta}_{ik} + r_i(\hat{\eta}_{ik}) \tag{4}$$

$$D_{ik}^U(p_k) = \hat{D}_{ik}(p_k) + r_i(\hat{D}_{ik}(p_k)) \tag{5}$$

where $\hat{\eta}_{ik}$ ($\hat{D}_{ik}(p_k)$) is the average of all the realizations of $\bar{\eta}_k$ ($D_k(p_k)$) that have been seen before the arrival of job i, and $r_i(\cdot)$ denotes the confidence radius of the respective random variable (which upper-bounds the deviation of an empirical average from its expectation with high probability). Therefore, η_{ik}^U is the upper-bound of confidence interval $[\hat{\eta}_{ik} - r_i(\hat{\eta}_{ik}), \hat{\eta}_{ik} + r_i(\hat{\eta}_{ik})]$ for estimating $\bar{\eta}_k$, and $D_{ik}^U(p_k)$ is the upper-bound of the confidence interval $[\hat{D}_{ik}(p_k) - r_i(\hat{D}_{ik}(p_k)), \hat{D}_{ik}(p_k) + r_i(\hat{D}_{ik}(p_k))]$ to estimate $D_k(p_k)$.

Let random variable X_i denote whether user i accepts price p_{ik_i} ($X_i = 1$) or not ($X_i = 0$). We have $E_{v_i \sim F_{k_i}}[d_{ik_i} X_i \mid \{d_{jk_j} X_j\}_{j<i, k_j=k_i}] = D_{k_i}(p_{k_i})$. We can compute empirical averages and confidence radiuses in a standard way [16]:

$$\hat{\eta}_{ik} = \frac{\sum_{j<i:k_j=k} \eta_j X_j \mathbb{1}(t_j + \tau_j < t_i)}{\sum_{j<i:k_j=k} X_j \mathbb{1}(t_j + \tau_j < t_i)}, \tag{6}$$

$$\hat{D}_{ik}(p_k) = \frac{\textit{total \# of type-}k\textit{ instances sold at } p_k}{\textit{\# of times that } p_k \textit{ is used}} \tag{7}$$

$$r_i(\hat{\eta}_{ik}) = \frac{\alpha}{1+\sum_{j<i:k_j=k} X_j \mathbb{1}(t_j+\tau_j<t_i)} + \sqrt{\frac{\alpha\hat{\eta}_{ik}}{1+\sum_{j<i:k_j=k} X_j \mathbb{1}(t_j+\tau_j<t_i)}} \tag{8}$$

$$r_i(\hat{D}_{ik}(p_k)) = \frac{\alpha}{1+N_i(p_k)} + \sqrt{\frac{\alpha\hat{D}_{ik}(p_k)}{1+N_i(p_k)}} \tag{9}$$

Here, we set $\alpha = O(\log n_k)$ to guarantee the high probability in Lem. 1 and relatively small confidence radiuses. $N_i(p_k)$ is the number of times p_k is used to price type-k jobs before arrival of job i. Thus, using η_{ik}^U and $D_{ik}^U(p_k)$ as the estimate of $\bar{\eta}_k$ and $D_k(p_k)$ to calculate $\hat{R}_{ik}(p_k)$, we guarantee that with high probability, $Rev^k(p_k)$ is upper-bounded by $\hat{R}_{ik}(p_k)$.

Lemma 1. *With probability at least* $1-n_k^{-2}$*, we have*

$$\hat{R}_{ik}(p_k) \geq Rev^k(p_k) \tag{10}$$

Proof of Lem. 1 is given in our technical report [22].

B. Pricing Mechanism

Our online pricing mechanism, *TOP*, is summarized in Alg. 1. In the algorithm, v_k^{max} denotes the upper-bound of per-instance budget of users requesting type-k instances. Naturally, price p_{ik} should fall in $[0, v_k^{max}]$. We discretize $[0, v_k^{max}]$ into the price set $\mathcal{P}_k$, where the candidate prices form a geometric sequence with common ratio $1+\delta_k$. Here δ_k determines the discretization granularity, which should be carefully decided: if δ_k is too small, the number of prices (arms) to choose among in the MAB algorithm would be very large, affecting algorithm performance; if δ_k is too large, the difference between the real optimal price and the best among our candidates will be large. The value of δ_k that we use in Alg. 1 has been carefully derived, supporting our analysis towards a sublinear regret bound of *TOP* in Sec. V.

In our system, the feedback of job runtime is received after a job is completed, not at the time of pricing the job. We adopt an *exploration phase* to handle delayed feedback: at the beginning, the cloud takes in $\theta_k n_k$ arrived jobs of type k, $\forall k \in [1, K]$, by setting nil prices, and collects runtimes of these jobs (lines 2-4). The collected runtimes are used to compute estimate of $\bar{\eta}_k$ (lines 16-18). The value of θ_k decides the duration of the exploration phase. A longer exploration phase leads to more profit loss due to zero charges, but brings more samples of job runtimes; and a shorter exploration phase risks larger estimation errors of (4) and (8) with fewer runtime samples. The value of θ_k in Alg. 1 has been carefully derived, to achieve a good tradeoff between profit lost and estimation error. It supports our proof that the estimation error of (4) and (8) (due to delayed feedback of job runtimes) in the later exploitation phase can be upper-bounded by the estimation error by the end of the exploration phase (Sec. V); then the reward obtained in the exploitation stage by choosing the best price as in (3) can well approximate the RHS of (2). After the exploration stage, the algorithm proceeds to the *exploitation phase* (lines 6-14). Upon the arrival of a user, if the virtual instances he requires are available, a price achieving the highest reward will be chosen and sent to the user. Whenever a user accepts the offered price (lines 11-14) or a job completes (lines 16-18), we obtain more samples to reshape the estimation of $D_k(p_k)$ and $\bar{\eta}_k$, respectively.

Algorithm 1: Occupation-Oblivious Pricing Mechanism – TOP

Input: K, T, $\mathbf{n}$, $\mathbf{C}$, $\mathbf{v^{max}}$, θ, δ
Output: $p_{ik}, \forall i \in \{1,\cdots,I\}, k = k_i$
Initialize: $D_{0k}(p_k) = \hat{\eta}_{0k} = y_k^0 = 0$, $\forall k \in \{1,\cdots,K\}$
Initialize: $\delta_k = (TC_k)^{-\frac{1}{3}} \log^{\frac{2}{3}} n_k$,
$\theta_k = \frac{(TC_k)^{\frac{2}{3}} \log^{\frac{2}{3}} n_k}{n_k}$,
$\mathcal{P}_k = \{\delta_k(1+\delta_k)^j \cap [0, v_k^{max}], j \in \mathbb{Z}\}$

Upon: arrival of job i
1 Set $k = k_i$;
2 **if** $\sum_{j=1}^{i} \mathbb{1}(k_j = k) \leq \theta_k n_k$ **then**
3 $p_{ik} = 0$ /* Allocate resource to job i*/;
4 Update allocated resource amount $y_k^{i+1} = y_k^i + d_{ik}$;
5 **else**
6 **if** $d_{ik} + y_k^i \leq C_k$ /* *Available virtual instances are sufficient to serve job* i */ **then**
7 Choose $p_{ik} \in \text{argmax}_{p_k \in \mathcal{P}_k} \hat{R}_{ik}(p_k)$, where $\hat{R}_{ik}(p_k)$ is computed using (3)-(5) /* Choose price with highest reward*/;
8 **else**
9 Post the unavailability of type-k virtual instances;
10 **end**
11 **if** *user* i *accepts the price* **then**
12 Update allocated resource amount $y_k^{i+1} = y_k^i + d_{ik}$;
13 Update empirical average number of instances sold $\hat{D}_{ik}(p_k)$ and confidence radius $r_i(\hat{D}_{ik}(p_k))$ according to (7) and (9);
14 **end**
15 **end**

Upon: completion of job i
16 $k = k_i$;
17 $y_k^{i+1} = y_k^{i+1} - d_{ik}$ /* virtual instances released*/;
18 Update $\hat{\eta}_{ik}$ and $r_i(\hat{\eta}_{ik})$ according to (6) and (8);

Discussions. Though we compute the reward functions based on the upper-bound of expected overall profit in (2) assuming fixed price p_k used for all type-k jobs, in the online algorithm, unit prices for type-k jobs arriving at different times may well be different. Each price p_{ik} is chosen as one maximizing the reward computed using the up-to-date estimates of η_{ik}^U and $D_{ik}^U(p_k)$. In our MAB online learning framework, we respect global resource capacity constraints to update rewards over time, based on the RHS of (2). By always choosing prices

maximizing expected overall profit using all resources, we avoid early exhaustion of resources by jobs of low charges.

In practice, there could be malicious users who purposely let their jobs run for a much longer time than the normal behavior of the job, *e.g.*, by creating a non-stopping loop. Such runtime samples would enlarge estimated $\bar{\eta}_k$, misleading the cloud provider to choose sub-optimal resource prices, which repel normal users and harm provider profit. As commonly handled in statistical modeling, we can adopt outlier detection procedures [23] to exclude suspicious samples from the estimation of $\bar{\eta}_k$. On the other hand, we note that rationale, non-malicious users have no intent to deliberately prolong their job runtimes, *e.g.*, a customer would like its ML model to be trained as soon as possible for usage in their inference services.

We note that n_k and v_k^{max} are needed as input to the algorithm, whose exact values are not known in the online setting. We can adopt an estimation of n_k when running the algorithm, *e.g.*, by calculating the empirical arrival rate of type-k users based on history and multiplying the arrival rate by T. We also use an estimated upper-bound of per-instance user budget as v_k^{max}, and will evaluate the impact of inaccurate estimation on algorithm performance in Sec. VI.

V. Theoretical Analysis

We now analyze our online learning based pricing mechanism in terms of time complexity and overall profit. All missing proofs can be found in our technical report [22].

Theorem 1 (Polynomial Runtime)**.** *Upon arrival of a user i, TOP produces price p_{ik_i} in $O(T^{\frac{1}{3}}C_{max}^{\frac{1}{3}}\log^{\frac{1}{3}} n_{max})$ time, where $C_{max} = \max_{k\in[1,K]} C_k$, $n_{max} = \max_{k\in[1,K]} n_k$.*

We next evaluate the *regret* of *TOP*, a common metric to evaluate performance of an online learning algorithm. The regret is defined to be the difference between the overall profit that our online algorithm obtains and that achieved by a best static pricing strategy. The best static pricing strategy knows all information of distributions F_k and $\mathcal{T}_k$, computes the best pricing vector $\vec{p} = \{p_1, \ldots, p_K\}$ that maximizes the expected overall profit based on the distributions, and uses $p_k \in \vec{p}$ for pricing all type-k virtual instances in the entire timespan, $\forall k \in [1, K]$. Such a best static solution has been widely used as benchmarks in regret analysis of online learning algorithms [16]. Our regret is defined as follows:

$$\begin{aligned} Regret &= \max_{\vec{p}\succeq 0}(Rev(\vec{p})) - E[Rev(\mathcal{A})] \\ &= Rev(\vec{p}^*) - E_{t_i\sim U(T),(v_i,d_{ik})\sim F_k,\tau_k\sim\mathcal{T}_k}[Rev(\mathcal{A})] \end{aligned} \quad (11)$$

Here $Rev(\mathcal{A})$ is the overall profit obtained by *TOP* under any realized input job sequence. Recall $Rev(\vec{p})$ denotes the expected overall profit at price vector $\vec{p}$, $Rev(\vec{p}^*)$ represents the expected overall profit achieved by the best static price vector $\vec{p}^*$, with $Rev(\vec{p}^*) = \max_{\vec{p}\succeq 0}(Rev(\vec{p}))$. We have $Rev(\vec{p}^*) = \sum_{k\in[1,K]} Rev^k(p_k^*)$, $Rev(\mathcal{A}) = \sum_{k\in[1,K]} Rev^k(\mathcal{A})$, and $Rev^k(\mathcal{A}) = \sum_{i:k_i=k} p_{ik} d_{ik} X_i$ (recall that X_i is a $0-1$ variable indicating whether user i accepts price p_{ik}).

We next upper-bound the regret in (11). The proof comprises three main parts: (1) upper-bound the discrepancy between the expected overall profit obtained by the best static price vector, where prices for different types of virtual instances are chosen from the respective candidate price sets in Alg. 1 (referred to as *the best static candidate prices*), and the expected overall profit obtained by running a revised version of *TOP*: prices are still computed as in line 7 in Alg. 1 and posted to users even if resource capacity is exceeded (*i.e.*, the condition in line 6 is not used); (2) upper-bound the discrepancy between the expected overall profit obtained by the best static candidate prices and that by *TOP* (considering resource constraint); and then (3) upper-bound the profit discrepancy between the actual best static pricing strategy (prices chosen from all possible prices instead of candidate sets in Alg. 1) and *TOP* (considering resource constraint).

Let $Rev(\mathcal{A}')$ denote the expected overall profit obtained at prices calculated by running the revised version of *TOP*. We have $Rev(\mathcal{A}') = \sum_{k\in[1,K]} Rev^k(\mathcal{A}')$ and $Rev^k(\mathcal{A}') = \sum_{i:k_i=k} p_{ik} D_k(p_k)$. In addition, we define $Rev(\vec{p}_c^*) = \sum_{k\in[1,k]} Rev^k(p_{ck}^*)$ to represent the expected overall profit obtained by the best static candidate price vector.

(1) Upper-bound of $\mathbf{Rev^k(p_{ck}^*) - Rev^k(\mathcal{A}')}$

The following lemma establishes a connection between $Rev^k(p_{ck}^*)$ and prices p_{ik}'s chosen in *TOP*, which enables us to further compare the upper-bound with the profit of *TOP*.

Lemma 2. *With high probability $1 - n_k^{-2}$, we have the following inequality for all i with $k_i = k$:*

$$\begin{aligned} Rev^k(p_{ck}^*) \le p_{ik} \min\{&TC_k(\bar{\eta}_k + 2r_i(\hat{\eta}_{ik})), \\ &n_k(D_k(p_k) + 2r_i(\hat{D}_{ik}(p_k)))\} \end{aligned} \quad (12)$$

where p_{ik} is the price offered to i, computed by TOP.

Lemma 3 is a straightforward corollary of Lemma 2, which is the key of algorithm TOP and will be used later. It guarantees that if the actual allocated number of type-k instances is at least $\max_{i=1}^{n_k} TC_k(\bar{\eta}_k + 2r_i(\hat{\eta}_{ik}))$, the profit obtained by selling type-k instances with TOP is at least $Rev^k(p_{ck}^*)$.

Lemma 3. *With high probability $1 - n_k^{-2}$, we have*

$$p_{ik} \ge \frac{Rev^k(p_{ck}^*)}{TC_k(\bar{\eta}_k + 2r_i(\hat{\eta}_{ik}))}, \quad \forall i : k_i = k \quad (13)$$

where p_{ik} is the price for i, computed by the revised TOP.

Proof. According to Lemma 2, we have

$$Rev^k(p_{ck}^*) \le p_{ik} TC_k(\bar{\eta}_k + 2r_i(\hat{\eta}_{ik})), \quad (14)$$

□

Let $\Delta(p_{ik})$ denote the difference between the expected profit per type-k job derived by the best static candidate prices and the profit obtained by offering p_{ik} job i.

$$\Delta(p_{ik}) = \max\{0, Rev^k(p_{ck}^*)/n_k - p_{ik} D_k(p_k)\} \quad (15)$$

Let $\Delta(p_k)$ be defined by (15) at any candidate price $p_{ik} = p_k$ and $\Delta(p_k) = 0$ if p_k is never chosen. Let $N(p_k)$ denote the

number of times that p_k is chosen in T time slots (the largest number of $N_i(p_k)$ over all i as used in (9). We have

$$Rev^k(p^*_{ck}) - Rev^k(\mathcal{A}^{'}) \leq \sum\nolimits_{p_k \in \mathcal{P}_k} \Delta(p_k) N(p_k) \quad (16)$$

We next upper-bound $\Delta(p_k)N(p_k)$ in Lemma 6, which is based on Lemmas 4 and 5. Intuitively, if we exactly know $D_k(p_k)$ for all $p_{ik} \in \mathcal{P}_k, \forall i : k_i = k$, we have an accurate estimate for the second term in the reward, *i.e.,* $p_k n_k D^U_{ik}(p_k)$ in (3). Then, we can use $p_{ik} n_k D_k(p_k)$ to upper-bound $Rev^k(p^*_{ck})$ (see the second term in (12)). Therefore, such an upper-bound exactly equals $p_{ik} D_k(p_k)$. Briefly, it means that if we know $D_k(p_k)$, $\Delta(p_{ik})$ will be zero (see the definition of $\Delta(p_{ik})$ in (15)). Thus, the existence of non-zero $\Delta(p_{ik})$ results from inaccurate estimation of $D_k(p_k)$. That is why $\Delta(p_{ik})$ is upper-bounded by $r_i(\hat{D}_{ik}(p_k))$, as below:

Lemma 4. *For each $i : k_i = k$, we have*

$$\Delta(p_{ik}) \leq p_{ik} O\big(r_i(\hat{D}_{ik}(p_k))\big) \quad (17)$$

Lemma 5 upper-bounds the confidence radiuses.

Lemma 5. *With high probability at least $1 - n_k^{-2}$, for each $i : k_i = k$, we have*

$$r_i(\hat{\eta}_{ik}) \leq \max\big(\frac{O(\log n_k)}{1 + \sum_{j<i:k_j=k} X_j \mathbb{1}(t_j + \tau_j < t_i)}, \sqrt{\frac{O(\log n_k)\bar{\eta}_k}{1 + \sum_{j<i:k_j=k} X_j \mathbb{1}(t_j + \tau_j < t_i)}}\big) \quad (18)$$

$$r_i(\hat{D}_{ik}(p_k)) \leq \max\big(\frac{O(\log n_k)}{N_i(p_k)+1}, \sqrt{\frac{O(\log n_k) D_k(p_k)}{N_i(p_k)+1}}\big) \quad (19)$$

Lemma 6. *Let $N(p_k)$ denote the number of times that price p_k is chosen in the entire timespan $[1, T]$ and η^U_k denote the UCB of $\bar{\eta}_k$ at the last time when p_k is chosen. We have*

$$\Delta(p_k)N(p_k) \leq O(p_k \log n_k)\big(1 + \frac{TC_k \eta^U_k}{n_k \Delta(p_k)}\big) \quad (20)$$

Note that the profit loss due to using prices chosen by TOP, compared to $Rev^k(p^*_{ck})$ comprises two parts. One is $\sum_{p_k \in \mathcal{P}_k} \Delta(p_k)N(p_k)$ which can be calculated by (20). The other is due to the exploration phase, where prices are set to zero. Given $p_k \in [\delta_k, v^{max}_k]$, the profit loss in the exploration phase can be upper-bounded by $v^{max}_k \theta_k n_k$. Therefore, we have the following based on (16) and Lemma 6:

$$\begin{aligned} & Rev^k(p^*_{ck}) - Rev^k(\mathcal{A}^{'}) \\ \leq & \sum_{\substack{p_k \in \mathcal{P}_k: \\ \Delta(p_k) \geq \epsilon_k}} \Delta(p_k)N(p_k) + \sum_{\substack{p_k \in \mathcal{P}_k: \\ \Delta(p_k) < \epsilon_k}} \Delta(p_k)N(p) + v^{max}_k \theta_k n_k \\ \leq & \; v^{max}_k \big(|\mathcal{P}_k| O(\log n_k)\big(1 + \frac{TC_k \eta^U_k}{\epsilon_k n_k}\big) + \epsilon_k n_k + \theta_k n_k\big), \end{aligned} \quad (21)$$

(2) Upper-bound of $\mathbf{Rev^k}(\mathbf{p^*_{ck}}) - \mathbf{E}[\mathbf{Rev^k}(\mathcal{A})]$

We have $E[d_{ik}X_i \mid \{d_{jk}X_j\}_{j<i:k_j=k_i=k}] = D_k(p_k)$, at price p_{ik} chosen in *TOP* for job i (with resource constraints). The profit of *TOP* can be calculated as:

$$Rev^k(\mathcal{A}) = \sum\nolimits_{i:k_i=k} p_{ik} d_{ik} X_i,$$

$$\text{conditioned on } \sum_{\substack{i:k_i=k \\ t_i \leq t < t_i + \tau_i}} d_{ik} X_i \leq C_k, \forall t = 1, \cdots, T \quad (22)$$

Lemma 7. *Let d^{max}_k be the maximum number of type-k instances required per user and $r_{max}(\bar{\tau}_k)$ be the maximum confidence radius on $\bar{\tau}_k$ after the exploration phase. The expected profit collected by* TOP *for selling type-k virtual instances can be lower-bounded as*

$$\begin{aligned} E[Rev^k(\mathcal{A})] \geq \min\big(& Rev^k(p^*_{ck})\big(1 - O(\frac{2r_{max}(\bar{\eta}_k)}{\bar{\eta}_k + 2r_{max}(\bar{\eta}_k)} + \frac{d^{max}_k}{C_k})\big) \\ & - v^{max}_k \theta_k n_k, \; Rev^k(\mathcal{A}^{'}) - O(v^{max}_k \sqrt{n_k \log n_k})\big) \end{aligned} \quad (23)$$

Basically, we prove Lem. 7 by lower-bounding the profit gained by TOP in two cases and use the smaller one to lower-bound $E[Rev^k(\mathcal{A})]$. In the first case, the expected allocated number of type-k instances by TOP is at least $T(C_k - d^{max}_k)\bar{\eta}_k$, which approaches the upper-bound of the total allocation number by the best static candidate price. Moreover, the prices chosen by TOP are high enough to satisfy (13), thus the total profit can be lower-bounded. The profit loss in the exploration phase is at most $v^{max}_k \theta_k n_k$.s In the other case, we resort to using $Rev^k(\mathcal{A}^{'})$ to lower-bound $E[Rev^k(\mathcal{A})]$, the second term in (23). Note $Rev^k(p^*_{ck}) \leq v^{max}_k TC_k \bar{\eta}_k$. Based on Lemma 7 and (21), we have

$$\begin{aligned} & Rev^k(p^*_{ck}) - E[Rev^k(\mathcal{A})] \\ & \leq v^{max}_k \big(|\mathcal{P}_k| O(\log n_k)\big(1 + TC_k \eta^U_k / (n_k \epsilon_k)\big) + \epsilon_k n_k + \theta_k n_k\big) \\ & + O\big(v^{max}_k [\sqrt{n_k \log n_k} + TC_k \bar{\eta}_k (\frac{2r_{max}(\bar{\eta}_k)}{\bar{\eta}_k + 2r_{max}(\bar{\eta}_k)} + \frac{d^{max}_k}{C_k})]\big) \end{aligned} \quad (24)$$

(3) Upper-bound of $\mathbf{Rev}(\vec{p}^*) - \mathbf{E}[\mathbf{Rev}(\mathcal{A})]$

Recall that our candidate prices in *TOP* satisfy $p_k \in [\delta_k, v^{max}_k], \forall p_k \in \mathcal{P}_k$. If $p^*_k \leq \delta_k$, then we have $Rev^k(p^*_k) - Rev(p^*_{ck}) \leq \delta_k TC_k \bar{\eta}_k$. Otherwise, suppose $\tilde{p}_k$ is the highest candidate price for type-k virtual instances that is no higher than p^*_k, *i.e.*, $\tilde{p}_k = \max\{p_k \in \mathcal{P}_k : \tilde{p}_k \leq p^*_k\}$. Since p^*_{ck} is the best candidate price which leads to the largest profit collected from selling type-k instances among all the price candidates in the best static pricing strategy, we have

$$Rev(\vec{p}^*_c) = \sum_{k=1}^{K} Rev^k(p^*_{ck}) \geq \sum_{k=1}^{K} Rev^k(\tilde{p}_k) = Rev(\vec{\tilde{p}}\,)$$

Moreover, we have $\tilde{p}_k \geq \frac{p^*_k}{1+\delta_k}$, $\forall k \in 1, \cdots, K$, according to definitions of $\tilde{p}_k$ and $\mathcal{P}_k$ in Alg. 1. Since $D_k(p_k)$ is non-increasing with respect to p_k, we further have

$$\begin{aligned} & Rev(\vec{p}^*_c) \geq \sum\nolimits_{k=1}^{K} Rev^k(\tilde{p}_k) \geq \sum\nolimits_{k=1}^{K} Rev^k(p^*_k)/(1+\delta_k) \\ & \geq \sum_{k=1}^{K} Rev^k(p^*_k)(1-\delta_k) \geq Rev(\vec{p}^*) - \sum_{k=1}^{K} v^{max}_k \delta_k TC_k \bar{\eta}_k \end{aligned} \quad (25)$$

Summing (24) over all $k \in [1, K]$ and combining with (25), we upper-bound $Rev(\vec{p}^*) - E[Rev(\mathcal{A})]$.

Theorem 2 (Regret)**.** *Let $\bar{\tau}^{max}_k$ be the largest possible value of the expected runtime of type-k jobs. Recall that d^{max}_k is the maximum number of type-k instances required per user. If $\bar{\tau}^{max}_k \leq \max\{T, \frac{T^{\frac{5}{3}} C_k^{\frac{2}{3}} \log^{\frac{2}{3}} n_k}{2n_k}\}$ and $d^{max}_k \leq (C_k \log n_k)^{\frac{2}{3}} T^{-\frac{1}{3}}$, with $\delta_k = (TC_k)^{-\frac{1}{3}} \log^{\frac{2}{3}} n_k$ and $\theta_k = \frac{(TC_k)^{\frac{2}{3}} \log^{\frac{2}{3}} n_k}{n_k}$, the regret of* TOP *is upper-bounded by $O\big(\sum_{k=1}^{K} v^{max}_k \big((TC_k)^{\frac{2}{3}} \log^{\frac{2}{3}} n_k + \sqrt{n_k \log n_k}\big)\big)$.*

In Theorem 2, a condition on $\bar{\tau}_k^{max}$ to guarantee that the expected number of completed jobs in the exploration phase is lower bounded, whose runtimes we need for estimating $\bar{\eta}_k$ before start of the exploitation stage. The condition on d_k^{max} to upper bound the maximum resource demand per user is for upper-bounding the maximum discrepancy between the allocated number of instances by TOP and C_k, such that the profit loss due to the extreme case that remaining resources are not sufficient to serve one more user is bounded (see $\frac{d_k^{max}}{C_k}$ part in (23)). Let $M(\theta_k n_k)$ be the expected number of jobs requiring type-k instances that complete in the exploration phase. Basically, a larger τ_k^{max} leads to a smaller $M(\theta_k n_k)$. Moreover, we can upper-bound $r_{max}(\bar{\eta}_k)$ by $r_i(\bar{\eta}_k)$ with $i = M(\theta_k n_k)$. Because the total number of realized η_i we collect is at least that aggregated in the exploitation phase. With more realizations revealed, the estimate of $\bar{\eta}_k$ is improved and $r_{max}(\bar{\eta}_k)$ becomes smaller. The regret bound is sublinear with the time horizon (T), resource capacity (C_k), and the total number of users that enter the system (n_k). It ensures that when time grows, the average difference between TOP and the best static pricing strategy goes to zero. Moreover, a regret sublinear w.r.t. resource capacity and total number of users is desirable; otherwise, the algorithm is trivial since a regret linear with C_k can be easily achieved even when the algorithm always chooses nil prices.

VI. Performance Evaluation

A. Simulation Setup

We evaluate our online pricing scheme using trace-driven simulations, exploiting Spark data analytics traces [24] (we identify a lack of ML job traces with GPU usage of each job). We extract from the traces jobs whose duration is larger than 30 minutes, given that typical machine learning jobs may take hours or days to finish. The job runtimes from the traces roughly follow a heavy-tailed distribution, which is representative in practice. There are in total $T = 10^5$ time slots and each time slot is 10 seconds long, simulating a period of roughly 10 days. We set job arrival times according to job start times in the traces (instead of following any i.i.d. assumption). There are 5 types of virtual instances, configured following `p2.xlarge`, `g3.8xlarge`, `g3.16xlarge`, `p2.8xlarge` and `p2.16xlarge` on Amazon EC2. The resources used by jobs in the traces, together with a randomly generated number of GPUs within $[1, 100]$, are mapped to these 5 types of instances to produce resource demands of jobs. The total number of jobs is about 10^5, and the number of jobs requesting each type of instances is around 17000, 19000, 27000, 20000, 17000, respectively. Note that the job durations and resource demands may not satisfy the assumptions in Theorem 2. The overall number of each type of instances is set to be the total number of instances demanded by users multiplied by a random factor in $[0.1, 1]$ by default. We produce the budget per instance (v_i/d_{ik}) for each user by multiplying a random factor from $[1, 300]$ by per-hour price of the requested type of GPU instances in Amazon EC2.

B. Impact of Parameters

We compute the upper-bound of profit of the best static pricing strategy by finding the best prices maximizing RHS of (2) with the Nelder-Mead Simplex algorithm [25], and the profit achieved by TOP. Each is computed for 50 times over different realizations of v_{ik}/d_{ik}, to derive the expectation used in computing an upper bound of the $regret$ of TOP in (11).

Fig. 1 shows the $regret$ obtained when the system runs for different timespans with different values of $n_k, \forall k$ (at different ratios of their default values). The regret is smaller when n_k's are smaller, consistent with our analysis in Theorem 2. Next, we evaluate the impact of δ_k by multiplying its value used in Alg. 1 by different factors. Fig. 2 shows that our choice of δ_k in Alg. 1 achieves the smallest regret. We next present the $regret$ obtained under different resource capacity levels by multiplying the default C_k's by 1, 0.1, 5, and 10. Fig. 3 shows that TOP performs better when the capacity is more scarce. which is consistent with our theoretical analysis. In Fig. 4, we further vary the ranges of resource demands of jobs, d_{ik}'s, by multiplying the range generated from the traces by 1, 5, 0.5, and 0.2. The influence of d_{ik} on the $regret$ is not obvious.

We next investigate the influence of inaccurate estimates of v_k^{max}'s on the $regret$ in Fig. 5. We observe that over-estimation may incur a larger regret while under-estimation is more desirable. An over-estimation may produce price candidates higher than the budgets of users, leading to lower overall profit, while lower candidate prices may not achieve the highest profit but are more likely acceptable by users.

Interestingly, we observe that the change of the regret with the growth of T is not monotonic (but definitely not a linear increase). Also the regret is negative in most cases, implying that our algorithm, a dynamic pricing strategy, in fact outperforms the best static pricing strategy. This reveals in practice, if the cloud provider fixes a set of prices (even set with knowledge of all upcoming jobs) and uses it on all incoming jobs, the profit obtained is worse than using our dynamic pricing strategy.

C. Comparison with Alternative Algorithms

We next compare TOP with three alternatives: (i) RPD, adapted from the online pricing mechanism in [11], where the prices are calculated by the price functions given in [11]; (ii) $TOP\text{-}simp$, where the exploration phase is omitted; and (ii) $TOP(\hat{n}_k)$, where the number of each type k of users used in Alg. 1 is always estimated by calculating the empirical arrival rate of type-k users based on history and multiplying the arrival rate by T. Fig. 4 shows that TOP always outperforms other pricing algorithms. The performance of $TOP(\hat{n}_k)$ is close to TOP, demonstrating the efficiency of the method we use to estimate n_k in an online setting. We also observe that it takes around 0.04s for our algorithm to produce the price upon arrival of each user, on a computer with a 1.3 GHz Intel Core i5 processor and 16 GB 1867 MHz LPDDR3 memory.

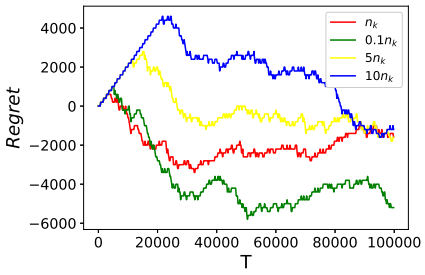

Fig. 1: *Regret* (varying n_k)

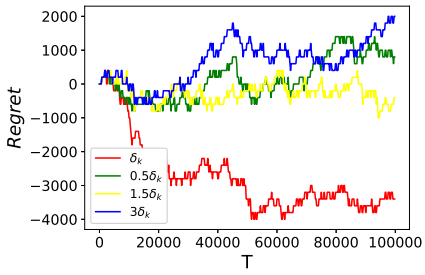

Fig. 2: *Regret* (varying δ_k)

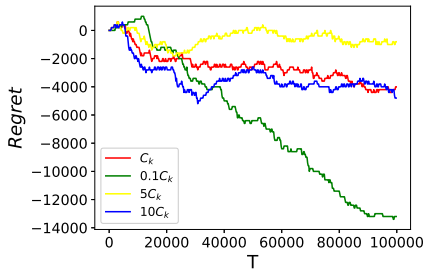

Fig. 3: *Regret* (varying C_k)

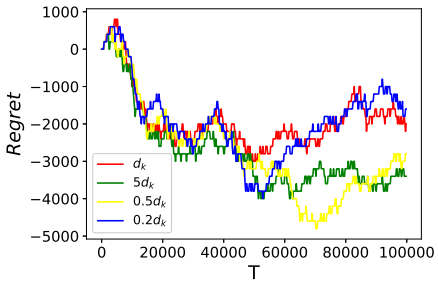

Fig. 4: *Regret* (varying d_{ik})

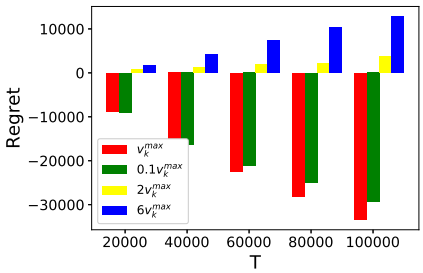

Fig. 5: *Regret* (inaccurate v_k^{max})

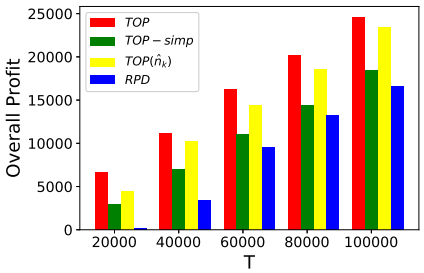

Fig. 6: Profit Comparison with Alternative Algorithms

VII. Conclusion

This paper proposes an occupation-oblivious pricing option for a cloud provider to charge a pre-determined lump-sum price to user jobs with uncertain completion times. Our pricing mechanism is based on a multi-armed bandit online learning framework, with novel designs of exploration and exploitation phases. Targeting provider profit maximization over the long run, we fit the online estimations into an online algorithm for dynamical pricing over time. Theoretical analysis shows a regret bound of our algorithm sublinear with the time horizon, resource capacities and user numbers. Trace-driven simulations also verify good performance of our algorithm under realistic settings.

VIII. Acknowledgements

This work was supported in part by grants from Hong Kong RGC under the contracts HKU 17204715, 17225516, 17202115E, C7036-15G (CRF), HKU URC Matching Funding, grants NSFC 61628209, 61571335, and grants Hubei Science Foundation 2016CFA030, 2017AAA125.

References

[1] "Google Cloud Platform," https://cloud.google.com/pricing/.
[2] "Microsoft Azure: Linux Virtual Machines Pricing," https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/.
[3] "Amazon EC2 Pricing," https://aws.amazon.com/ec2/pricing/on-demand/.
[4] "Amazon EC2 T2 Instances," https://aws.amazon.com/ec2/instance-types/t2/.
[5] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep Learning Applications and Challenges in Big Data Analytics," *Journal of Big Data*, vol. 2, no. 1, pp. 1 – 21, 2015.
[6] "Google Cloud Machine Learning Engine," https://cloud.google.com/ml-engine/.
[7] "Microsoft Cognitive Toolkit: Model Gallery," https://www.microsoft.com/en-us/cognitive-toolkit/features/model-gallery/.
[8] "https://aws.amazon.com/about-aws/whats-new/2016/09/introducing-amazon-ec2-p2-instances-the-largest-gpu-powered-virtual-machine-in-the-cloud/," 2016.
[9] "https://cloudplatform.googleblog.com/2017/02/GPUs-are-now-available-for-Google-Compute-Engine-and-Cloud-Machine-Learning.html," 2017.
[10] O. Yadan, K. Adams, Y. Taigman, and M. Ranzato, "Multigpu Training of ConvNets," *arXiv:1312.5853*, 2013.
[11] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. Lau, "Online Auctinos in IaaS: Welfare and Profit Maximization with Server Costs," in *Proc. of ACM SIGEMETRICS*, 2015.
[12] ——, "An Online Stochastic Buy-Sell Mechanism for VNF Chains in the NFV market," *IEEE JSAC - Special Issue on Game Theory for Networks*, vol. 35, no. 2, pp. 1 – 15, 2017.
[13] "Amazon EC2 Spot Instances Pricing," https://aws.amazon.com/ec2/spot/pricing/.
[14] W. Wang, B. Liang, and B. Li, "Revenue Maximization with Dynamic Auctions in IaaS Cloud Markets," in *Proc. of IEEE ICDCS*, 2013.
[15] Z. Zhang, Z. Li, and C. Wu, "Optimal Posted Prices for Online Cloud Resource Allocation," in *Proc. of ACM SIGMETRICS*, 2017.
[16] R. Kleinberg, A. Slivkins, and E. Upfal, "Multi-Armed Bandits in Metric Spaces," in *Proc. of ACM STOC*, 2008.
[17] R. Combes, S. Magureanu, A. Proutiere, and C. Laroche, "Learning to Rank," in *Proc. of ACM SIGMETRICS*, 2015.
[18] Y. Liu and M. Liu, "An Online Approach to Dynamic Channel Access and Transmission Scheduling," in *Proc. of MobiHoc*, 15.
[19] S. Anand, N. Garg, and A. Kumar, "Resource Gugmentation for Weighted Flow-Time Explained by Dual Fitting," in *Proc. of SODA*, 2012.
[20] B. Kalyanasundaram and K. Pruhs, "Speed is as Powerful as Clairvoyance," *Journal of the ACM*, vol. 47, pp. 214 – 221, 2000.
[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *Proc. of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, 2016.
[22] "Occupation-oblivious pricing of cloud jobs via online learning," Tech. Rep., https://1drv.ms/f/s!Al00m3RtXAw8ae2emwbA1HFTcG4.
[23] G. M. Oyeyemi, A. Bukoye, and I. Akeyede, "Comparison of Outlier Detection Procedures in Multiple Linear Regressions," *American Journal of Mathematics and Statistics*, vol. 5, no. 1, pp. 37 – 41, 2015.
[24] "Spark performance analysis – data analytics traces," https://kayousterhout.github.io/trace-analysis/.
[25] "Nelder-Mead Simplex algorithm," https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html.