# Dynamic Pricing and Placement for Distributed Machine Learning Jobs

1st Xueying Zhang
*School of Cyber Science and Engineering*
*Wuhan University*
snowyzhang@whu.edu.cn

2nd Ruiting Zhou
*Wuhan University*
*The Chinese University of Hong Kong*
ruitingzhou@whu.edu.cn

3rdJohn C.S. Lui
*Department of Computer Science and Engineering*
*The Chinese University of Hong Kong*
cslui@cse.cuhk.edu.hk

4th Zongpeng Li
*School of Computer Science*
*Wuhan University*
zongpeng@whu.edu.cn

***Abstract*—Nowadays distributed machine learning (ML) jobs usually adopt a parameter server (PS) framework to train models over large-scale datasets. Such ML job deploys hundreds of concurrent workers, and model parameter updates are exchanged frequently between workers and PSs. Current practice is that workers and PSs may be placed on different physical servers, bringing uncertainty in jobs' runtime. Also, existing cloud pricing policy often charges a fixed price according to the job's runtime. Although this pricing strategy is simple to implement, such pricing mechanism is not suitable for distributed ML jobs whose runtime is stochastic and can only be estimated according to its placement after job admission. To supplement existing cloud pricing schemes, we design a dynamic pricing and placement algorithm, DPS, for distributed ML jobs. DPS aims to maximize cloud provider's profit, which dynamically calculates unit resource price upon a job's arrival, and determines job's placement to minimize its runtime if offered price is accepted to users. Our design exploits the multi-armed bandit (MAB) technique to learn unknown information based on past sales. DPS balances the exploration and exploitation stage, and selects the best price based on the reward which is related to job runtime. Our learning-based algorithm increases the provider's profit, and achieves a sub-linear regret with both the time horizon and the total job number, compared to benchmark pricing schemes. Extensive evaluations also validates the efficacy of DPS.**

## I. INTRODUCTION

Nowadays, machine learning (ML) has become an indispensable framework which trains models over large-scale datasets. To train a large model, hundreds of concurrent workers (typically implemented on virtual machines (VMs) or containers) are deployed in parallel to update shared model parameters, in particular, using the popular *parameter server* (PS) architecture [1][2]. In the PS framework, one or multiple PSs store and maintain global model parameters. In each training iteration, the PSs pull computed gradients from workers and update their maintained parameters respectively; and then PSs push updated parameters back to the workers. Workers and PSs of a ML job can be distributed on different physical servers, when they cannot be completely placed on the same server, or to maximize the utilization of expensive cloud resources on servers [3].

Different from general cloud computing jobs, distributed ML jobs have their distinct features. First, due to the frequent exchange of parameter updates between workers and PSs, the parameter transmission time accounts for a large proportion of job runtime, and if workers and PSs are deployed on different servers, then it will consume significant amount of inter-server bandwidth [4]. Furthermore, it is typically difficult for the job owner to estimate how long a job may take, before the placement of the job is determined. Second, running ML jobs that are often deployed on GPU servers is time-consuming and costly. For example, training a GoogLeNet model over the ImageNet-1k dataset takes 23.4 hours on a Titan supercomputer server with 32 NVIDIA K20 GPUs [5], and would cost more than $172 by renting p2.8xlarge instances from Amazon EC2 [6]. For such jobs, preemption is not acceptable since it may further delay their job completion time. It is also common that job owners prefer to know the price before job admission, such that the cost is within their budget.

In today's cloud market, service providers often adopt the pay-as-you-go pricing policy, where users pay a fixed unit price for resource demand according to the job runtime. Amazon EC2 [6], Google Cloud [7] and Microsoft Azure [8] all adopt the per-hour charging model for on-demand or preemptible VM instances (*e.g.,* spot instances). Another preferred pricing option is an advanced purchase of VMs for one to three years in a specified region. For example, Amazon EC2 provides significant discount (up to 75%) with savings plans and reserved instances [6]. However, existing pricing mechanism is not suitable for distributed ML jobs, due to following reasons. First, different users have different budgets with heterogeneous demands. Fixed pricing fails to attract

1. Corresponding author: Ruiting Zhou.

2. This work was supported by the Fundamental Research Funds for the Central Universities (2042019kf0016) and the GRF 14201819. Part of this work by Ruiting Zhou was done while she was visiting CUHK.

978-1-7281-8275-9/20/$31.00 ©2020 IEEE
DOI 10.1109/BigCom51056.2020.00029

many customers and cannot capture the changing supply and demand in the market. As a result, either overpricing or underpricing would happen and this jeopardizes users' experience as well as the provider's profit. Although dynamic pricing is offered by Amazon EC2 spot instances, they are only recommended for jobs that can tolerate preemption. Second, existing providers require job owners to estimate job runtime, and pay in advance before the job admission. The job owners will be further charged if they underestimate the runtime. However, as mentioned before, the runtime of a distributed ML job is uncertain and depends on job placement.

Hence, a fundamental problem for ML service providers is: *Given limited resources, how to dynamically charge and place distributed ML jobs, such that the job runtime is minimized and the provider's profit is maximized, without knowing users' budgets?*

To supplement existing cloud pricing models, we propose a novel dynamic pricing and placement mechanism, DPS, for distributed ML jobs. To the best of our knowledge, this paper is the *first* formal study that combines dynamic pricing and placement design in a dynamic online setting for ML jobs. As shown in Fig. 1, our online algorithm involves two stage decisions: (i) A user arrives and informs the cloud service provider of its job configuration. It specifies the type and the number of workers and PSs needed, parameter size and the number of required training epoch, but the user doesn't need to submit any information about the job's runtime and budget. The cloud service provider posts unit resource prices upon its arrival, and calculates the cost to complete its jobs. The user evaluates the price according to its budget. (ii) If the user accepts the offered price, the cloud provider deploys this job on its servers to minimize job runtime. Note that shorter runtime has a positive impact on the provider's profit, as more resources can be released and then resold. We employ a multi-armed bandit (MAB) framework to learn from past sales, and select best unit price based on rewards, while the reward is computed according to job runtime. The detailed technical contributions are as follows:

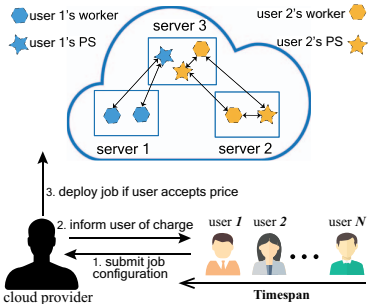

Fig. 1: An illustration of pricing and placement process.

**First**, We formulate the profit maximization problem as a mixed integer linear program (MILP). The program precisely models the feature of ML jobs (uncertain runtime), and captures all factors that would influence the decisions (resource capacity constraints and budget limitation). Even in the offline setting with known information, this problem is proven to be NP-hard. The challenges further escalate when both the budget and the job runtime is stochastic and unknown. To overcome these challenges, we divided our design into two steps: pricing strategy and placement algorithm.

**Second**, the critical challenge in pricing design is that the budget of each job is a private information and its runtime is stochastic and hard to estimate before the job admission. To tackle this issue, we design an online learning strategy based on the MAB framework. Specifically, we first get the upper-bound of profit related to unit resource prices as well as the runtime of jobs. Job runtime is calculated according to the experience and its placement, and its exact value is updated when a job is completed. The price interval is appropriately discretized and we get a set of prices (*arms*) for selection. Each price corresponds to a related reward contributing to the total profit. The unit price with the highest reward will be used for the current job. Then its reward is adjusted according to the feedback (*i.e.,* whether the user accepts the offered charge and job runtime). Therefore, the job that has a high budget and its resources occupation (involving resources demand and job runtime) matches its budget can be accepted, which means the higher profit can be obtained.

**Third**, in the placement design, to reduce the time for parameter transmission among different physical servers, we deploy as few servers as possible to serve a job. Hence, we place jobs on servers in a greedy manner so workers and PSs of a job are placed as close as possible, which can reduce the job runtime. Our online algorithm, DPS, takes both pricing and placement into account and they work in concert with each other.

**Last but not the least**, we conduct rigorous theoretical analysis to examine our algorithm's performance. DPS has a polynomial time complexity. Moreover, we derive a sub-linear upper-bound on the regret, which implies that our algorithm has an asymptotically optimal performance. The results show that DPS outperforms other benchmark algorithms. The overall profit achieved by DPS is 125%, 115%, 122% and 238% of *BFP*'s, *DPS-simple*'s, *TOP*'s [9] and *Random*'s, respectively. This percentage increases over time, and the performance of DPS in practice is better than the theoretical analysis.

The rest of the paper is organized as follows. Sec. II reviews related literature. The system model is introduced in Sec. III. The learning-based algorithm is presented in Sec. IV and evaluated in Sec. V. Sec. VI concludes the paper.

## II. Related Work

***Dynamic Pricing for Cloud Resources.*** Compared with traditional cloud resource pricing methods, dynamic pricing strategies which can enhance cloud provider profit have been explored in recent years. Wang *et al.* [10] and Shi *et al.* [11] study how to dynamically price VMs to pursue overall profit or social welfare maximization in online auctions. An auction-based online mechanism for virtual machines pricing in clouds is proposed in [12]. Those pricing strategies either focus on posted price mechanism [13] or request the user

to determine the runtime of its job. However, the runtime of a ML job is stochastic and unknown to users before its completion.

***Multi-armed Bandit Schemes.*** To address the unknown budget and runtime of jobs, we design an our pricing algorithm based on MAB, which is an effective online learning and optimization framework [14]. Bubeck *et al.* [15] has proven that MAB is efficacious to get a good trade-off between exploration and exploitation in sequential decisions. The basic MAB framework learns to choose an optimal arm without considering any system constraints. Mahdavi *et al.* [16] extend the study of MAB where the learner aims to maximize total reward, given that some additional constraints need to be satisfied. However, it is not applicable to our system where the resources can be reused after a job is completed.

## III. problem model

***System Model and Job Information.*** Suppose the cloud service provider provides $K$ types of workers and $M$ types of parameter servers (PSs), and they are deployed on $S$ different physical servers. Let $[X]$ denote the integer set $\{1, 2, ...X\}$. $C_k$ ($C_m$) denotes the maximum number of available type-$k$ workers (type-$m$ PSs), $\forall k \in [K]$ ($\forall m \in [M]$). The system operates in discrete time slots $t = 1, 2, ..., T$. There are $N$ users arriving during the timespan and each user comes with a machine learning (ML) job to be processed. Each job needs to train a ML model over a large input dataset, using synchronous training method. Let $t_i$ denote the arrival time of job $i$. The configuration of job $i$ includes the following information: (i) the worker type $k_i$ and the PS type $m_i$; (ii) the number of type-$k_i$ workers (type-$m_i$ PSs) $d_{ik}$ ($d_{im}$); (iii) the size of the gradients/parameters $w_i$; (iv) required training epochs $\alpha_i$. Moreover, users usually have their budgets for completing jobs, which are private and will not be revealed to the cloud provider. We denote job $i$'s budget as $v_i$. Let $B_i$ denote the information of job $i$:$B_i = \{k_i, d_{ik}, m_i, d_{im}, w_i, \alpha_i\}$.

***Stochastic Assumptions.*** The budgets of users are usually related to their demands of resources. We assume that the budget and the demand of jobs which require same type of worker and PS follow a jointly unknown distribution. For each resource combination $(k, m)$, the $(demand, budget)$ pairs of users who request type-$k$ workers and type-$m$ PSs are independently and identically distributed, namely, $(d_{ik}, d_{im}, v_i)$ are i.i.d., and drawn from an unknown distribution $F_{k,m}$.

***Runtime of Jobs.*** In the parameter server architecture, the runtime of an epoch for a ML job consists of the following two parts: (i) *computation time*, which is the sum of computation time at the workers (*i.e.*, the data training time and gradients computation time) and at the PSs (*i.e.*, the parameters updating time); (ii) *transmission time*, which is the time for workers to push gradients to PSs and pull updated parameters from PSs. According to job $i$'s configuration as well as the historical knowledge, the computation time $\beta_i$ can be estimated. Next, we analyze job $i$'s transmission time. If a worker is deployed on a server where there is no PS, the data transmission time (*i.e.*, the worker exchanges gradients with all PSs) in an epoch is $2w_i/b_i$, where $b_i$ is the bandwidth between the PS and worker. Each type-$k$ worker (type-$m$ PS) reserves some bandwidth, which is denoted as $h_k$ ($H_m$). Hence, $b_i = \min(h_{k_i}, H_{m_i}/d_{ik})$. When all PSs and workers are located on the same server, the bandwidth to exchange gradients/parameters is abundant between them and the transmission time is negligible. Let $q_i$ represents whether all workers and PSs serving job $i$ are in the same server (1) or not (0). Hence, the runtime of job $i$:

$$\tau_i = \alpha_i \beta_i + \alpha_i (1 - q_i)(2w_i/b_i). \tag{1}$$

***Decision Variables.*** After receiving job $i$'s request, the cloud provider prices the resources and informs user the current unit prices $p_{ik}$ and $p_{im}$ for type-$k_i$ worker and type-$m_i$ PS. When its overall payment, *i.e.*, $p_{ik} \times d_{ik} + p_{im} \times d_{im}$, is no larger than its budget $v_i$, the user accepts the offered price and the provider need to decide how to place this job on available servers; otherwise, the user will leave without purchasing anything. Suppose the number of type-$k_i$ workers serving job $i$ on server $s$ is $x_{ski}$ and the number of type-$m_i$ PSs serving job $i$ on server $s$ is $z_{smi}$. Let $\mathcal{X}_{ski}$ ($\mathcal{Z}_{smi}$) denote the number of idle type-$k_i$ workers (type-$m_i$ PSs) on server $s$ when job $i$ arrives.

***Problem Formulation.*** To pursue the maximum overall profit over the system timespan, the cloud provider dynamically prices resources upon user arrives, and decides the placement for this job if the user accepts the price. This offline optimization problem can be formulated as the following mixed integer linear program (MILP):

$$\text{maximize} \sum_{i\in[N]} \Big( \sum_{k\in[K]} p_{ik} d_{ik} + \sum_{m\in[M]} p_{im} d_{im} \Big) f_i \tag{2}$$

subject to:

$$f_i = \mathbb{1}\{d_{ik} + y_k^i \le C_k, d_{im} + y_m^i \le C_m, \sum_{k\in[K]} p_{ik} d_{ik} + \sum_{m\in[M]} p_{im} d_{im} \le v_i, \forall k, \forall m\}, \tag{2a}$$

$$y_m^i = \sum_{\substack{j\in[i-1]:\\ t_j+\tau_j \ge t_i}} d_{jm} f_j, \forall m \in [M], \forall i \in [N], \tag{2b}$$

$$y_k^i = \sum_{\substack{j\in[i-1]:\\ t_j+\tau_j \ge t_i}} d_{jk} f_j, \forall k \in [K], \forall i \in [N], \tag{2c}$$

$$\sum_{s\in[S]} x_{ski} = d_{ik} f_i, \forall i \in [N], \tag{2d}$$

$$\sum_{s\in[S]} z_{smi} = d_{im} f_i, \forall i \in [N], \tag{2e}$$

$$0 \le x_{ski} \le \mathcal{X}_{ski}, \forall s \in [S], \forall i \in [N], \tag{2f}$$

$$0 \le z_{smi} \le \mathcal{Z}_{smi}, \forall s \in [S], \forall i \in [N], \tag{2g}$$

$$x_{ski} \in \mathbb{N}, z_{smi} \in \mathbb{N}, \forall s \in [S], \forall i \in [N], \tag{2h}$$

$$p_{ik}, p_{im} \ge 0, \forall i \in [N], \forall m \in [M], \forall k \in [K]. \tag{2i}$$

The $\mathbb{1}\{X\}$ is an indicator function, which equals 1 if $X$ is true and 0 otherwise. Variable $f_i$ in constraint (2a) indicates that job $i$ runs if there are enough resources and the user accepts the price. $y_k^i$ ($y_m^i$) in constraint (2b)/(2c) is the total number

of type-$k$ workers (type-$m$ PSs) that have been occupied at the time of job $i$'s arrival. Constraints (2d) and (2e) guarantee the number of type-$k$ workers (type-$m$ PSs) allocated to job $i$ is consistent with its request. The resource capacity of physical servers for running PSs and workers is formulated by constraints (2f) and (2g).

## IV. Algorithm Design and Analysis

Our learning-based algorithm consists of two subroutines. We introduce the pricing mechanism and placement strategy in Sec. IV-A. The theoretical analysis are presented in Sec. IV-B.

### A. Algorithm Design

#### 1) Dynamic Pricing Mechanism

***Design Rationale.*** In order to set prices to maximize the profit, the core idea is to estimate the likelihood that a user will accept the offered price without the knowledge of the (*demand*, *budget*) distribution as well as the runtime of jobs, so that the best prices can be set. We propose an online algorithm based on UCB (Upper Confidence Bound) to dynamically determine prices. Specifically, we learn the runtime and the distribution according to past jobs, and set prices for arriving jobs based on the learned knowledge.

Without loss of generality, we normalize $p_k$ and $p_m$ into $[0,1]$, *i.e.,* $p_k(p_m) \in [0,1]$. Suppose *fixed-price strategy* is adopted, *i.e.*, same prices $p_k$ and $p_m$ are offered to jobs requesting type-$k$ workers and type-$m$ PSs during the system timespan, which can be viewed as the expectation of realized prices. Let $Q_k(p_k)$ denote the expected number of type-$k$ workers sold at price $p_k$ to any job who requests type-$k$ workers, *i.e.,* $Q_k(p_k) = \mathbb{E}_{(d_{ik},d_{im},v_i)\sim F_{k,m}}[\hat{d}_{ik}]$, where $\hat{d}_{ik} = d_{ik}$ if $v_i \geq p_k d_{ik} + p_m d_{im}$ and $\hat{d}_{ik} = 0$ otherwise. We denote the total number of jobs requesting type-$k$ workers in $[1,T]$ as $n_k$. Similarly, we have $Q_m(p_m)$ and $n_m$ for type-$m$ PSs.

We first analyze the upper-bound of the overall profit under the *fixed-price strategy*. The analysis can be divided into two cases: (i) the resources are always sufficient to serve all jobs; (ii) the resources are insufficient, which means current running jobs occupy all resources. In the first case, the total expected profit of type-$k$ workers (type-$m$ PSs) with a fixed price $p_k$ ($p_m$) is $n_k p_k Q_k(p_k)$ ($n_m p_m Q_m(p_m)$). To simplify the description, we focus on workers in the following analysis. In the second case, at most $C_k$ type-$k$ workers are available at any time slot due to the resource capacity. In each time slot, if type-$k$ workers have been exhausted, the maximum expected number of type-$k$ workers which can be allocated to new jobs is $C_k(1 - \mathbb{E}_{i:k_i=k}[\tau_i]/T)$ (job $i$ runs in $\tau_i$ slots, then if we average its workload over $T$ slots, job $i$ runs $\tau_i/T$ slot in each slot). Thus, the average maximum total profit of type-$k$ workers with a fixed price $p_k$ is $p_k C_k(T - \mathbb{E}_{i:k_i=k}[\tau_i])$. We denote $T - \tau_i$ as $\mu_i$. Then, this expected profit can be formulated as $p_k C_k \overline{\mu}_k$, where $\overline{\mu}_k = \mathbb{E}_{i:k_i=k}[\mu_i]$. Let $\mathcal{A}(\boldsymbol{p}^K, \boldsymbol{p}^M)$ denote the expected overall profit under fixed prices $\boldsymbol{p}^K$ and $\boldsymbol{p}^M$, where $\boldsymbol{p}^K = \{p_1, p_2, \cdots, p_K\}$ and $\boldsymbol{p}^M = \{p_1, p_2, \cdots, p_M\}$. Under this price strategy, we have

$$\mathcal{A}(\boldsymbol{p}^K, \boldsymbol{p}^M) \leq \min\Big( \sum_{k\in[K]} p_k C_k \overline{\mu}_k + \sum_{m\in[M]} p_m C_m \overline{\mu}_m, \sum_{k\in[K]} n_k p_k Q_k(p_k) + \sum_{m\in[M]} n_m p_m Q_m(p_m) \Big). \quad (3)$$

To maximize the long-term profit, the prices that maximize the upper-bound, *i.e.,* RHS of (3), should be used. However, it is intractable to determine such prices, because both the budget distribution and the runtime are unknown in the online setting. Therefore, we design an online learning algorithm based on multi-armed bandit (MAB) to estimate the uncertain distributions and set dynamic prices to maximize the *profit upper-bound in expectation*. First, we discretize the price interval $[0,1]$, and get a candidate price set $\mathcal{P}_k$ ($\mathcal{P}_m$) for type-$k$ workers (type-$m$ PSs). Upon the arrival of job $i$, price $p_{k_i} \in \mathcal{P}_{k_i}$ and $p_{m_i} \in \mathcal{P}_{m_i}$ are chosen for this job. For each price $p_k \in \mathcal{P}_k$ ($p_m \in \mathcal{P}_m$), we define a reward contributing to the overall profit, and the prices with the highest reward are picked. We define the reward of price as follows:

$$\hat{R}_{ik}(p_k) = \min(n_k p_k Q^U_{ik}(p_k), p_k C_k \mu^U_{ik}) \quad (4)$$

$$\hat{R}_{im}(p_m) = \min(n_m p_m Q^U_{im}(p_m), p_m C_m \mu^U_{im}). \quad (5)$$

Intuitively, $\hat{R}_{ik}(p_k)$ and $\hat{R}_{im}(p_m)$ are estimates of the upper-bound of the expected profit of type-$k$ workers and type-$m$ PSs. Here, $Q^U_{ik}(p_k)$ ($Q^U_{im}(p_m)$) is the UCB of $Q_k(p_k)$ ($Q_m(p_m)$) estimated before job $i$ arrives; $\mu^U_{ik}$ ($\mu^U_{im}$) is the UCB of $\overline{\mu}_k$ ($\overline{\mu}_m$) estimated before job $i$ arrives, as defined below:

$$\mu^U_{ik} = \hat{\mu}_{ik} + r_i(\hat{\mu}_{ik}), \mu^U_{im} = \hat{\mu}_{im} + r_i(\hat{\mu}_{im}), \quad (6)$$

$$Q^U_{ik}(p_k) = \hat{Q}_{ik}(p_k) + r_i(\hat{Q}_{ik}(p_k)), \quad (7)$$

$$Q^U_{im}(p_m) = \hat{Q}_{im}(p_m) + r_i(\hat{Q}_{im}(p_m)), \quad (8)$$

where $\hat{\mu}_{ik}$, $\hat{\mu}_{im}$, $\hat{Q}_{ik}(p_k)$ and $\hat{Q}_{im}(p_m)$ are the current average values of their realizations of $\overline{\mu}_k$, $\overline{\mu}_m$, $Q_k(p_k)$ and $Q_m(p_m)$, respectively. These parameters can be computed as follows:

$$\hat{Q}_{ik}(p_k) = \frac{\textit{total \# of type-k workers sold at } p_k}{\textit{\# of times } p_k \textit{ has been used}}, \quad (9)$$

$$\hat{Q}_{im}(p_m) = \frac{\textit{total \# of type-m PSs sold at } p_m}{\textit{\# of times } p_m \textit{ has been used}}, \quad (10)$$

$$\hat{\mu}_{ik} = \frac{\sum_{i'<i:k_{i'}=k} \mu_{i'} f_{i'} \mathbb{1}(t_{i'} + \tau_{i'} < t_i)}{\sum_{i'<i:k_{i'}=k} f_{i'} \mathbb{1}(t_{i'} + \tau_{i'} < t_i)}, \quad (11)$$

$$\hat{\mu}_{im} = \frac{\sum_{i'<i:m_{i'}=m} \mu_{i'} f_{i'} \mathbb{1}(t_{i'} + \tau_{i'} < t_i)}{\sum_{i'<i:m_{i'}=m} f_{i'} \mathbb{1}(t_{i'} + \tau_{i'} < t_i)}. \quad (12)$$

And $r_i(X)$ is the *confidence radius* of the random variable $X$ such that for $X^U = \hat{X} + r_i(\hat{X})$, inequality $|X - \hat{X}| \leq r_i(X)$ holds with high probability. Therefore, suitable confidence radius needs to be designed, since a smaller confidence radius implies a more accurate estimate of the parameter $X$. Let $N^k_i(p_k)$ ($N^m_i(p_m)$) be the number of times that $p_k$ ($p_m$) has been used to price jobs requesting type-$k$ workers (type-$m$ PSs) before job $i$ arrives. We design the confidence radius[1] as:

[1]Only variables $r_i(\hat{Q}_{im}(p_m))$ and $r_i(\hat{\mu}_{im})$ are presented here since $r_i(\hat{Q}_{ik}(p_k))$ and $r_i(\hat{\mu}_{ik})$ are defined the same way.

$$r_i(\hat{Q}_{im}(p_m)) = \frac{\eta}{1+N_i^m(p_m)} + \sqrt{\frac{\eta \hat{Q}_{im}(p_m)}{1+N_i^m(p_m)}}, \tag{13}$$

$$r_i(\hat{\mu}_{im}) = \frac{\eta}{1+\sum_{i'<i:m_{i'}=m} f_{i'}\mathbb{1}(t_{i'}+\tau_{i'}<t_i)} + \sqrt{\frac{\eta \hat{\mu}_{im}}{1+\sum_{i'<i:m_{i'}=m} f_{i'}\mathbb{1}(t_{i'}+\tau_{i'}<t_i)}}, \tag{14}$$

where $\eta = \Theta(\log n_m)$.

**Algorithm 1** Dynamic Pricing Strategy *(DPS)*

**Input:** $K, M, T, \{C_k\}_{k\in[K]}, \{C_m\}_{m\in[M]}, \{n_k\}_{k\in[K]}, \{n_m\}_{m\in[M]}$
**Initialize:** $\theta_k = (TC_k \log n_k)^{\frac{2}{3}}/n_k, \theta_m = (TC_m \log n_m)^{\frac{2}{3}}/n_m, \delta_k \in (0,1), \delta_m \in (0,1), \mathcal{P}_k = \{\delta_k(1+\delta_k)^z \cap [0,1] : z \in \mathbb{Z}\}, \mathcal{P}_m = \{\delta_m(1+\delta_m)^z \cap [0,1] : z \in \mathbb{Z}\}$
**Upon:** job $i$ comes with its information $B_i$

1: Set $k = k_i, m = m_i$;
2: **if** $\sum_{i'=1}^{i} \mathbb{1}(k_{i'} = k) \le \theta_k n_k$ or $\sum_{i'=1}^{i} \mathbb{1}(m_{i'} = m) \le \theta_m n_m$ **then**
3: $p_{ik}, p_{im} = 0$;
4: $(\boldsymbol{x}_{ki}, \boldsymbol{z}_{ki}) = PA(B_i, \{\mathcal{X}_{ski}\}, \{\mathcal{Z}_{smi}\})$;
5: Update the number of occupied resource:
6: $y_k^{i+1} = y_k^i + d_{ik}, y_m^{i+1} = y_m^i + d_{im}$;
7: **else**
8: **if** $d_{ik} + y_k^i \le C_k$ and $d_{im} + y_m^i \le C_m$ **then**
9: Pick $p_{ik} \in \arg\max_{p_k \in \mathcal{P}_k} \hat{R}_{ik}(p_k)$;
10: Pick $p_{im} \in \arg\max_{p_m \in \mathcal{P}_m} \hat{R}_{im}(p_m)$;
11: Inform the user price $p_{ik}d_{ik} + p_{im}d_{im}$;
12: **if** *user accepts the offered price* **then**
13: $(\boldsymbol{x}_{ki}, \boldsymbol{z}_{ki}) = PA(B_i, \{\mathcal{X}_{ski}\}, \{\mathcal{Z}_{smi}\})$;
14: Compute runtime $\tau_i$ according to $(\boldsymbol{x}_{ki}, \boldsymbol{z}_{ki})$ and (1);
15: Update the number of occupied resource:
16: $y_k^{i+1} = y_k^i + d_{ik}, y_m^{i+1} = y_m^i + d_{im}$;
17: According to (6)-(14), update parameters:
18: $Q_{ik}^U(p_k), Q_{im}^U(p_m), \mu_{ik}^U, \mu_{im}^U$;
19: **end if**
20: **else**
21: Reject this user's request;
22: **end if**
23: **end if**

**Upon:** job $j$ is completed

1: Release and update the resource:
2: $y_{k_j}^{j+1} = y_{k_j}^{j+1} - d_{jk}, y_{m_j}^{j+1} = y_{m_j}^{j+1} - d_{jm}$;
3: $\{\mathcal{X}_{sk(j+1)} = \mathcal{X}_{sk(j+1)} + x_{skj}\}_{s\in[S]}$;
4: $\{\mathcal{Z}_{sm(j+1)} = \mathcal{Z}_{sm(j+1)} + z_{skj}\}_{s\in[S]}$;
5: Reshape the estimates $\hat{\mu}_{ik}$ and $\hat{\mu}_{im}$ according to (11)(12);

***Online Pricing Algorithm.*** Our dynamic pricing strategy DFS is summarized in Alg.1. In the initialization phase, we elaborately design $\delta_k$ and $\delta_m$ to discretize the prices interval and get sets of candidate prices. Note that parameters $\delta_k$ and $\delta_m$ have a significant impact on our algorithm performance, and we will illustrate this impact in Sec.V. Inspired by the trade-off between *exploration* and *exploitation* in classic MAB framework, we set nil prices for jobs in the beginning stage (lines 2-6), such that users can accept the price and we can obtain some information about job's runtime. The smaller $\theta_k$ and $\theta_m$, the shorter is the exploration time. Hence, parameters $\theta_k$ and $\theta_m$ indicate the balance between exploration and exploitation: a shorter exploration stage means less loss of profit but larger risk on the estimation error; in contrast, a longer exploration stage means larger loss of profit but smaller risk of estimation error. Here, $\theta_k$ and $\theta_m$ are derived carefully to reach a good balance between them. After the exploration phase, our algorithm starts the exploitation stage. If there are enough available resources to serve job $i$, the reward of each price in candidate sets is calculated based on the historical knowledge and the prices with the highest rewards are chosen (lines 7-11). If the user accepts the price, the placement algorithm *PA* is invoked (line 13) to decide how to deploy this job on servers, which is described in detail in next subsection. According to the placement strategy and the experiences of the computation time, job $i$'s runtime $\tau_i$ is approximately calculated in line 14. Meanwhile, the amount of occupied resources is updated and we update the estimated parameters $Q_{ik}^U(p_k), Q_{im}^U(p_m), \mu_{ik}^U$ and $\mu_{im}^U$ (lines 14-18), which will be used to calculate the rewards of prices when the next job arrives. Once a job is completed, the occupied resources are released and related resource parameters as well as the parameters (*i.e.*, $\hat{\mu}_{ik}$ and $\hat{\mu}_{im}$) related to the exact runtime are updated.

**Algorithm 2** Placement Algorithm *(PA)*

**Input:** $w_i, d_{ik}, d_{im}, H_{m_i}, h_{k_i}, \{\mathcal{X}_{ski}\}_{s\in[S]}, \{\mathcal{Z}_{smi}\}_{s\in[S]}$
**Initialize:** $\boldsymbol{x}_{ki} = \boldsymbol{0}, \boldsymbol{z}_{ki} = \boldsymbol{0}, c = 1$

1: Sort all servers in descending order of $\mathcal{X}_{ski}$ and $\mathcal{Z}_{smi}$, the result sequence is denoted as $\{s_1, s_2, \cdots, s_S\}$;
2: **for** $s = s_1, s_2, \cdots, s_S$ **do**
3: **if** $\mathcal{X}_{ski} \ge d_{ik}$ and $\mathbb{Z}_{smi} \ge d_{im}$ **then**
4: /* Deploy all workers and PSs on server $s$ */
5: $x_{ski} = d_{ik}, z_{smi} = d_{im}$;
6: /* Update current idle resources */
7: $\mathcal{X}_{sk(i+1)} = \mathcal{X}_{ski} - d_{ik}, \mathcal{Z}_{sm(i+1)} = \mathcal{Z}_{ski} - d_{im}$;
8: **Return** $\boldsymbol{x}_{ki}, \boldsymbol{z}_{ki}$
9: **end if**
10: **end for**
11: /* Multiple servers are used */
12: **while** $\sum_{j=1}^{c} \mathcal{X}_{s_jki} < d_{ik}$ **do**
13: $x_{s_cki} = \mathcal{X}_{s_cki}, \mathcal{X}_{s_ck(i+1)} = 0$;
14: $c = c + 1$;
15: **end while**
16: $x_{s_cki} = d_{ik} - \sum_{j=1}^{c-1} \mathcal{X}_{s_jki}, \mathcal{X}_{s_ck(i+1)} = \mathcal{X}_{s_cki} - x_{s_cki}$;
17: $c = 1$;
18: **while** $\sum_{j=1}^{c} \mathcal{Z}_{s_jmi} < d_{im}$ **do**
19: $z_{s_cmi} = \mathcal{Z}_{s_cmi}, \mathcal{Z}_{s_cm(i+1)} = 0$;
20: $c = c + 1$;
21: **end while**
22: $z_{s_cmi} = d_{im} - \sum_{j=1}^{c-1} \mathcal{Z}_{s_jmi}, \mathcal{Z}_{s_cm(i+1)} = \mathcal{Z}_{s_cmi} - z_{s_cmi}$;
23: **Return** $\boldsymbol{x}_{ki}, \boldsymbol{z}_{ki}$

*2) Placement Policy*

If user $i$ accepts the offered price, the cloud provider needs to decide how to place its job in physical servers so to minimize the runtime, since shorter runtime results in larger reward, leading to higher profit. The placement problem for job $i$ can be formulated as:

$$\text{minimize } \tau_i \tag{15}$$

subject to: $(2d) \sim (2h)$,

where $f_i = 1$. If there is a server having enough resources to serve job $i$, placing this job on the server results in the shortest runtime. We focus on another case, *i.e.,* $q_i = 0$. In this case, we try to use as few servers as possible to serve a job. As shown in Alg.2, all servers are sorted according to their current idle resources. Lines 2-10 determine whether there is a server on which all workers and PSs requested by job $i$ can be deployed. If there is no such server, workers and PSs are deployed in a greedy manner to serve job $i$ (lines 12-23).

### B. Theoretical Analysis

***Runtime.*** First, we analyze the runtime of DPS, which can be completed in polynomial time.

**Theorem 1.** *Our algorithm determines the price $p_{ik}$, $p_{im}$ and makes placement decision in $O[2(TC_{max}\log N)^{1/3}+S^2]$ time for each job, where $C_{max} = \max(C_k, C_m), \forall k \in [K], \forall m \in [M]$.*

*Proof.* See Appendix. A. □

***Regret Analysis.*** Now we theoretically analyze the *regret* of our algorithm. The benchmark used in our work is the best fixed-price strategy, which knows all information in advance and offers fixed unit prices for resources to all jobs with the maximal expected profit[2]. The *regret* is the difference between the expected overall profit obtained by our algorithm and that by the best fixed-price strategy. Theorem 2 below shows that the regret of DPS is *sub-linear* with both the timespan and the total job number.

Let $\boldsymbol{p}_*^K$ and $\boldsymbol{p}_*^M$ denote the price vectors of the best fixed-price mechanism. Therefore, the regret of our algorithm can be defined as follows:

$$\begin{aligned} Regret(\mathcal{L}) &= \mathcal{A}(\boldsymbol{p}_*^K, \boldsymbol{p}_*^M) - \mathbb{E}[\mathcal{A}(\mathcal{L})] \\ &= \sum_{k\in[K]} \mathcal{A}_k(p_*^k) + \sum_{m\in[M]} \mathcal{A}_m(p_*^m) - \mathbb{E}(\sum_{k\in[K]} \mathcal{A}_k(\mathcal{L}) + \sum_{m\in[M]} \mathcal{A}_m(\mathcal{L})) \\ &= \sum_{k\in[K]} [\mathcal{A}_k(p_*^k) - \mathbb{E}[\mathcal{A}_k(\mathcal{L})]] + \sum_{m\in[M]} [\mathcal{A}_m(p_*^m) - \mathbb{E}[\mathcal{A}_m(\mathcal{L})]], \end{aligned} \tag{16}$$

where $\mathcal{A}(\mathcal{L})$ is the total expected profit achieved by DPS. The regret $Regret(\mathcal{L})$ is derived in three steps: (i) we analyze the upper bound of the difference between the total expected profit of the best fixed candidate prices (namely, the best prices in candidate sets $\mathcal{P}_k$ and $\mathcal{P}_m$ in Alg.1) and the profit of our policy without considering resources capacity (namely, the condition in line 7 in Alg.1 is ignored); (ii) the upper bound of the difference between the best fixed candidate prices and our policy considering the resources capacity is derived; (iii) finally, the upper bound of $Regret(\mathcal{L})$ (namely, the gap between the best fixed prices and DPS) is obtained.

**Theorem 2.** *Let $\delta_k = (TC_k)^{-1/3}(\log n_k)^{2/3}$ and $\delta_m = (TC_m)^{-1/3}(\log n_m)^{2/3}$ in Alg.1. Then, the regret of DPS is $O[(K+M)((N\log N)^{1/2} + (TC_{max}\log N)^{2/3})]$.*

For ease of description, we denote the overall expected profit of the best fixed candidate prices as $\mathcal{A}(\boldsymbol{p}_*^{cK}, \boldsymbol{p}_*^{cM})$ and that of our policy without considering resources capacity is denoted as $\mathcal{A}(\mathcal{L}')$. In the rest of the proof, we mainly focus on PSs (the profit of workers can be analyzed the same way).

**Lemma 1** (The upper-bound of $\mathcal{A}_m(p_*^{cm}) - \mathcal{A}_m(\mathcal{L}')$)**.** *Let $\Delta(p_{im})$ denote the discrepancy between the expected profit per job requesting type-m PSs achieved by $p_*^{cm}$ and that achieved by offering our price $p_{im}$ for job i, namely, $\Delta(p_{im}) = \max\{\mathcal{A}_m(p_*^{cm})/n_m - p_{im}Q_m(p_m), 0\}$. We have :*

$$\begin{aligned} &\mathcal{A}_m(p_*^{cm}) - \mathcal{A}_m(\mathcal{L}') \\ &\le \theta_m n_m + \sum_{\substack{p_m\in\mathcal{P}_m:\\ \Delta(p_m)\ge\sigma_m}} \Delta(p_m)N(p_m) + \sum_{\substack{p_m\in\mathcal{P}_m:\\ \Delta(p_m)<\sigma_m}} \Delta(p_m)N(p_m) \\ &\le \sigma_m n_m + \theta_m n_m + |\mathcal{P}_m| O(\log n_m)(1 + C_m\mu_m^U/(\sigma_m n_m)), \end{aligned} \tag{17}$$

*where $\sigma_m = \delta_m C_m \bar{\mu}_m / n_m$, $N(p_m)$ is the number of times that $p_m$ has been picked during the whole timespan and $\mu_m^U$ is the UCB of $\bar{\mu}_m$ when price $p_m$ is picked at the last time.*

**Claim 1.1.** *With probability at least $1 - n_m^{-2}$ holds, for each job i with $m_i = m$:*

$$\mathcal{A}_m(p_*^{cm}) \le p_{im} \cdot \min(n_m Q'_m(p_m), C_m \bar{\mu}'_m), \tag{18}$$

*where $Q'_m(p_m) = Q_m(p_m) + 2r_i(\hat{Q}_{im}(p_m))$, $\bar{\mu}'_m = \bar{\mu}_m + 2r_i(\hat{\mu}_{im})$.*

*Proof.* See Appendix. B. □

In view of Claim 1.1, we have a straightforward corollary as shown in Claim 1.2.

**Claim 1.2.** *Let $p_{im}$ denote the price for job i designed by our algorithm without considering the resources capacity. We have*

$$\Pr[p_{im} \ge \mathcal{A}_m(p_*^{cm})/(C_m\bar{\mu}'_m)] \ge 1 - n_m^{-2}, \forall i : m_i = m. \tag{19}$$

As mentioned in Lemma 1, $\Delta(p_m)$ is defined at any candidate price $p_{im} = p_k$ and equals zero if price $p_m$ has never been chosen. Then, we have

$$\mathcal{A}_m(p_*^{cm}) - \mathcal{A}_m(\mathcal{L}') \le \sum_{p_m\in\mathcal{P}_m} \Delta(p_m)N(p_m). \tag{20}$$

Intuitively, if the distribution $Q_m(p_m)$ is accurately known for all $p_{im} \in \mathcal{P}_m, \forall i : m_i = m$, we can accurately estimate the term $n_m p_m Q_{im}^U(p_m)$ in (5). Then, $p_{im}n_m Q_m(p_m)$ can be used to upper bound $\mathcal{A}_m(p_*^{cm})$ (as shown in (18)). Hence, such an upper bound exactly equals $p_{im}Q_m(p_m)$. Namely, $\Delta(p_{im})$ will equal zero if $Q_m(p_m)$ is known to us, which means that the existence of non-zero $\Delta(p_{im})$ results from $Q_m(p_m)$'s incorrect estimate. Therefore, $\Delta(p_{im})$ is actually upper bounded by $r_i(\hat{Q}_{im}(p_m))$. Next, we upper bound $\Delta(p_{im})$ to further upper bound $\Delta(p_m)N(p_m)$ in the RHS of (20).

**Claim 1.3.** *For each job i, we have $\Delta(p_i m) \le p_{im} \cdot O(r_i(\hat{Q}_{im}(p_m)))$. Furthermore, we have*

$$\Delta(p_m)N(p_m) \le O(p_m \log n_m)(1 + C_m\mu_m^U/(n_m\Delta(p_m))). \tag{21}$$

*Proof.* See Appendix. C. □

Since the profit loss caused by DPS compared to $\mathcal{A}_m(p_*^{cm})$ consists of two parts: (i) $\sum_{p_m\in\mathcal{P}_m}\Delta(p_m)N(p_m)$ calculated by (21); (ii) prices are set to nil, in the exploration stage where the loss of profit can be upper bounded by $\theta_m n_m$. Combining them with Claim 1.2 and Claim 1.3, Lemma 1 is proofed. □

**Lemma 2** (The upper-bound of $\mathcal{A}_m(p_*^{cm}) - \mathbb{E}[\mathcal{A}_m(\mathcal{L})]$)**.** *Let $d_{max}^m$ denote the maximum number of type-m workers requested per job and $r_{max}(X)$ denote the maximum confidence radius on X after the exploration stage. We have $\mathcal{A}_m(p_*^{cm}) - \mathbb{E}[\mathcal{A}_m(\mathcal{L})]$*

[2]Such benchmark has been widely used in the regret analysis in online learning-based algorithm.

$$\leq \sigma_m n_m + \theta_m n_m + |\mathcal{P}_m| O(\log n_m)(1 + C_m \mu_m^U / (\sigma_m n_m))$$
$$+ O[\sqrt{n_m \log n_m} + C_m \overline{\mu}_m (\frac{2 r_{max}(\overline{\mu}_m)}{\overline{\mu}_m + 2 r_{max}(\overline{\mu}_m)} + \frac{d_{max}^m}{C_m})].$$

*Proof.* See Appendix. D. □

**Lemma 3** (the upper-bound of $\sum_{m\in[M]}[\mathcal{A}_m(p_*^m) - \mathbb{E}[\mathcal{A}_m(\mathcal{L})]]$)**.** *For each $\sigma_m > 0$, we have*

$$\sum_{m\in[M]} [\mathcal{A}_m(p_*^m) - \mathbb{E}[\mathcal{A}_m(\mathcal{L})]]$$
$$\leq \sum_{m\in[M]} [\sigma_m n_m + \theta_m n_m + |\mathcal{P}_m| O(\log n_m)(1 + C_m \mu_m^U / (\sigma_m n_m)) +$$
$$O(\sqrt{n_m \log n_m} + C_m \overline{\mu}_m / (1 + \frac{\overline{\mu}_m}{O(r_{max}(\overline{\mu}_m))}) + d_{max}^m \overline{\mu}_m + \delta_m C_m \overline{\mu}_m)].$$

*Proof.* See Appendix. E. □

Finally, we prove Theorem 2 based on Lemma3 , as shown in Appendix. F.

□

## V. performance evaluation

***Simulation Setup.*** We evaluate our algorithm over a timespan of 10000 time slots (*i.e.,* $T = 10000$) and each time slot is 5 minutes. The numbers of worker types and PS types are 15 and 10 respectively. The bandwidth of each type worker ranges between 100 Mbps and 5 Gbps and that of each type PS ranges between 5 Gbps and 20 Gbps. We assume there are 50 physical servers. The number of each type workers (PSs) deployed on each server is in $[0, 30]$ ($[0, 18]$). Then, the total resource capacity (*i.e.,* $C_k$ and $C_m$) can be calculated. The arrival time, resource demand and other information of jobs are set according to the real-world traces [17]. In particular, we analyze the users' preference of resources and their prices in the real-world traces to estimate and simulate budgets of users. The total number of arrived jobs is around 10000. We set the price of each type worker (PS) according to Amazon EC2 pricing [6] and normalize it into $[0, 1]$.

***Performance of Our Complete Algorithm DPS.*** We compare *DPS* with four alternatives:

- *Best fixed-price strategy (BFP)*: The optimal fixed unit price of each resource is set with the *priori* knowledge of all jobs' full information.
- *DPS-simple*: This is a variant of *DPS*, where the exploration stage (lines 2-6 in Alg.1) is omitted.
- *TOP*: It is adapted from an online pricing algorithm for cloud jobs [9]. Since this algorithm only involves pricing virtual instances, we slightly modify it to fit our system model and add placement module for it.
- *Random*: This algorithm randomly picks unit price from interval $[0, 1]$ upon each job's arrival, and making placement decision according to *PA*.

Fig. 2 shows that *DPS* outperforms other algorithms. In the first few time slots ($t < 1120$), the regret of *DPS* increases since the price is set nil in the exploration stage and the relation between users' budget and demands is unknown. After this, the regret of *DPS* decreases and equals zero at $t = 3140$, *i.e.,* the profit achieved by *DPS* is comparable to *BFP*'s. The negative regret means our algorithm exceeds *BFP* and this superiority grows over time. The regret of *DPS-simple* shows that the exploration stage plays an import role, which makes the estimation of job runtime accurate. The overall profits of algorithms are presented in Fig. 3. At the end of timespan, total profit achieved by *DPS* is 125%, 115%, 122% and 238% of *BFP*'s, *DPS-simple*'s, *TOP*'s and *Random*'s, respectively.

***The Impact of Parameters.*** The regret of *DPS* under different total job numbers (at different ratios, 0.1, 5 and 10 of the default $N$) is drawn in Fig. 4. At the beginning, the regret is smaller when $N$ is smaller. As time goes on, the larger $N$, the faster the regret decreases. Fig. 5 shows the effect of the value of $\delta$ (*i.e.,* $\delta_k$ and $\delta_m$) on *DPS*'s regret. When $\delta$'s value is too small, the number of candidate prices in $\mathcal{P}_k$ ($\mathcal{P}_m$) becomes larger. Hence, learning period gets longer. When $\delta$ is too large, the regret is growing. It is shown that the regret obtained by our choice of $\delta$ is the smallest. Then, we analyze the impact of $\theta$'s value on the performance of *DPS*. As shown in Fig. 6, when $\theta$ gets smaller, the regret is smaller at the beginning stage but it decreases more slowly in the later. A larger $\theta$ makes the exploration phase longer and leads to a larger regret in this phase. Our choice of $\theta$ shows a good trade-off between the exploration and exploitation.

***Performance of Placement Algorithm (PA).*** We compare our placement strategy with *random placement (RS)* algorithm to show its efficiency. The pricing mechanism in Alg.1 with random placement method (instead of *PA*) is used for comparison, which is denoted as *DPS-RS*. As shown in Fig. 7, the overall profit obtained by *DPS* is larger than that by *DPS-RS* and *BFP* when $T = 4000$, and the difference increases over time. The total runtime of all completed jobs under *PA* is always shorter than that under *RS*. Furthermore, the discrepancy between them become significantly larger as the number of completed jobs increases.

## VI. Conclusion

This paper is the first paper that addresses the dynamic pricing problem for distributed machine learning jobs, while jointly taking the placement into consideration. Our algorithm consists of two subroutines: (i) a dynamic pricing mechanism that determines the best price upon the arrival of each job, with a goal of maximizing provider's profit; (ii) a placement strategy that minimizes the runtime of accepted jobs. Through theoretical analysis, we show that our algorithm achieves a sub-linear regret with both the timespan and the total job number. Large-scaled simulation study based on real world data also verifies good performance of our algorithm, compared to state-of-the-art pricing mechanisms.

## Appendix

### A. Proof of Theorem 1

*Proof.* Lines 1-6 in Alg.1 can be done in a constant time. In Lines 9-10, our algorithm computes the reward over all the candidate prices in $\mathcal{P}_{k_i}$ and $\mathcal{P}_{m_i}$. Now, we focus on $\mathcal{P}_{k_i}$. Since $\mathcal{P}_k$ is initialized as $\{\delta_k(1+\delta_k)^z \cap [0,1] : z \in \mathbb{Z}\}$ for type-$k$

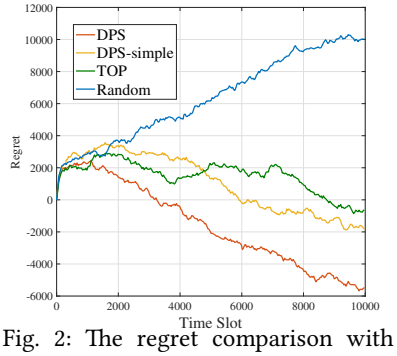


Fig. 2: The regret comparison with other algorithms .

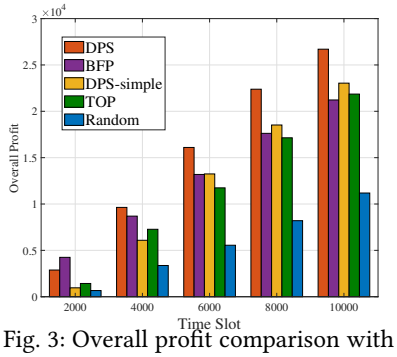


Fig. 3: Overall profit comparison with other algorithms.

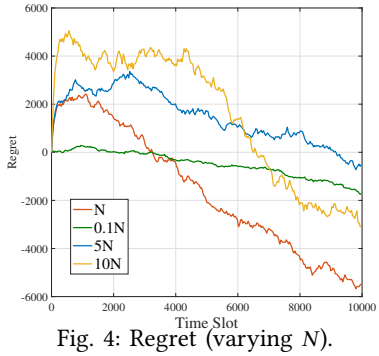


Fig. 4: Regret (varying $N$).

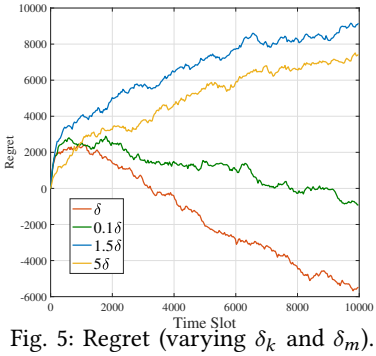


Fig. 5: Regret (varying $\delta_k$ and $\delta_m$).

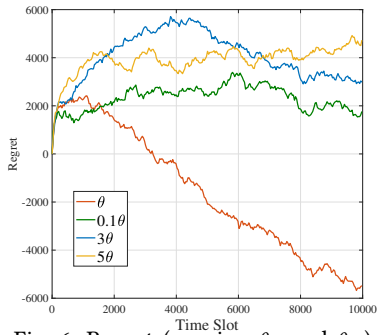


Fig. 6: Regret (varying $\theta_k$ and $\theta_m$).

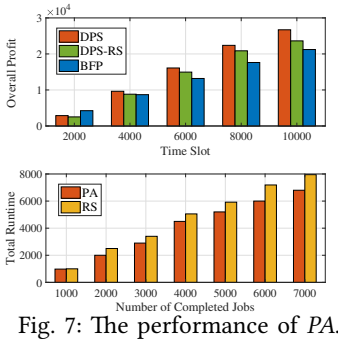


Fig. 7: The performance of *PA*.

worker, we have $\delta_k(1+\delta_k)^{|\mathcal{P}_k|} \geq 1$ and $\delta_k(1+\delta_k)^{|\mathcal{P}_k|-1} < 1$, which means $|\mathcal{P}_k| = \lceil \log_{1+\delta_k} \delta_k^{-1} \rceil$. Let $\delta_k = (TC_k)^{-1/3}(\log n_k)^{2/3}$, then we have $|\mathcal{P}_k| \leq \delta_k^{-1} \log n_k$. Thus, Line 9 in Alg.1 can be done in $O((TC_k \log n_k)^{1/3})$ time. Similarly, Line 10 can be done in $O((TC_m \log n_m)^{1/3})$ time. Next, we analyze the runtime of placement algorithm *PA* in Alg.2. In the worst case, the sorting time (line 1) is $O(S^2)$. Therefore, for each job, the runtime of our algorithm is $O[2(TC_{max} \log N)^{1/3} + S^2]$.

### B. Proof of Claim 1.1

*Proof.* As shown in Line 10 in Alg.1, the price with the highest reward $\hat{R}_{im}(p_m)$ in the candidate price set $\mathcal{P}_m$ is chosen in each round. Hence, for each user $i$ with $m_i = m$, we know $\hat{R}_{im}(p_{im}) = \max_{p_{im} \in \mathcal{P}_m} \hat{R}_{im}(p_m)$, which implies

$$\hat{R}_{im}(p_{im}) \geq \hat{R}_{im}(p_*^{cm}), \forall p_{im}, p_*^{cm} \in \mathcal{P}_m. \tag{22}$$

Morever, we know that the probability of inequalities $\mu_{im}^U \geq \overline{\mu}_{m_i}$ and $Q_{im}(p_{m_i})^U \geq Q_{m_i}(p_{m_i})$ holding is at least $1-n_m^{-2}$ based on the result in [18]. Therefore, we have $\hat{R}_{im}(p_*^{cm}) \geq \mathcal{A}_m(p_*^{cm})$ with high probability at least $1-n_m^{-2}$. Then, we have

$$\Pr[\hat{R}_{im}(p_{im}) \geq \mathcal{A}_m(p_*^{cm})] \geq 1-n_m^{-2}, \forall p_{im}, p_*^{cm} \in \mathcal{P}_m. \tag{23}$$

According to the definition of $\hat{R}_{im}(p_{im})$, we have that

$$\hat{R}_{im}(p_{im}) \leq p_{im} \cdot \min\{n_m Q'_m(p_m), C_m \bar{\mu}'_m\} \tag{24}$$

with probability at least $1-n_m^{-2}$. Combining (23) and (24), the Claim 1.1 follows.

### C. Proof of Claim 1.3

*Proof.* According to $\Delta(p_{im})$'s definition, if there is estimate error (namely, $\Delta(p_{im}) > 0$), we know$\mathcal{A}_m(p_*^{cm}) > n_m p_{im} Q_m(p_{im}), \forall m \in [M]$. Combining this inequality with the property of (19), we obtain

$$Q_m(p_{im}) < C_m \bar{\mu}'_m / n_m, \forall m \in [M]. \tag{25}$$

Let $\mathcal{J}_m$ denote the set of jobs requesting type-$m$ PSs. According to Claim 1.1, we know

$$\mathcal{A}_m(p_*^{cm}) \leq p_{im} n_m T(Q_m(p_{im}) + 2r_i(\hat{Q}_{im}(p_m))), \forall i \in \mathcal{J}_m.$$

Combining it with the definition of $\Delta(p_{im})$, we obtain $\Delta(p_i m) \leq 2p_{im} r_i(\hat{Q}_{im}(p_m))$, namely,

$$\Delta(p_i m) \leq p_{im} \cdot O(r_i(\hat{Q}_{im}(p_m))). \tag{26}$$

Then, we upper bound the confidence radii $r_i(\hat{Q}_{im}(p_m))$ and $r_i(\hat{\mu}_{im})$. According to the result in [18], when $\eta = \Theta(\log n_m)$, we know $r_i(\hat{X}) \leq 3\eta/(1+N_i(X)) + 3\sqrt{\eta \mathbb{E}[X]/(1+N_i(X))}$ holding with probability at least $1-n_m^{-2}$. Therefore, with high probability at least $1-n_m^{-2}$, we have

$$r_i(\hat{Q}_{im}(p_m)) \leq \max\{\frac{O(\log n_m)}{1+N_i^m(p_m)}, \sqrt{\frac{Q_m(p_m) O(\log n_m)}{1+N_i^m(p_m)}}\}, \tag{27}$$

$$r_i(\hat{\mu}_{im}) \leq \max\{\frac{O(\log n_m)}{1+\sum_{i'<i:m_{i'}=m} f_{i'} \mathbb{1}(t_{i'}+\tau_{i'} < t_i)}, \sqrt{\frac{\overline{\mu}_m O(\log n_m)}{1+\sum_{i'<i:m_{i'}=m} f_{i'} \mathbb{1}(t_{i'}+\tau_{i'} < t_i)}}\}. \tag{28}$$

Combining (26) (27) with (25), for all job $i \in \mathcal{J}_m$, we get

$$\Delta(p_{im}) \leq O(p_{im} \cdot \max(\frac{\log n_m}{1+N_i(p_{im})}, \sqrt{\frac{\log n_m Q_m(p_{im})}{1+N_i(p_{im})}})). \tag{29}$$

Removing its dependency on $i$ and rearranging this inequality, the Claim 1.3 follows.

### D. Proof of Lemma 2

*Proof.* The total profit of type-$m$ PSs (*i.e.*, $\mathcal{A}_m(p_{im})$) achieved by our DPS in expectation is $\mathbb{E}(\sum_{i\in\mathcal{J}_m} p_{im}d_{im}f_i)$ if the resource is always sufficient (namely, without considering the resources capacity). Taking the resources capacity into account, our DPS will stop offering resources when type-$m$ PSs are not enough to serve the job, even if the price for job $i$ is within the user's budget. Based on Azuma-Hoeffding inequality, we know that $\sum_{i\in\mathcal{J}_m}|p_{im}d_{im}f_i - p_{im}Q_m(p_{im})| \le O(n_m\log n_m)$ holds with high probability at least $1-n_m^{-2}$. Hence, we have

$$\mathbb{E}(\sum_{i\in\mathcal{J}_m} p_{im}d_{im}f_i) \ge \sum_{i\in\mathcal{J}_m} p_{im}Q_m(p_{im}) - O(n_m\log n_m). \tag{30}$$

Moreover, there exists such a case where the workload is high so that $\mathbb{E}(\sum_{i\in\mathcal{J}_m} d_{im}f_i) \ge \overline{\mu}_m(C_m - d^m_{max})$. We denote the set of jobs accepting the deal in the cases where the resource is sufficient as $\mathcal{J}'_m$. Then, we obtain

$$\begin{aligned}&\mathbb{E}[\sum_{i\in\mathcal{J}'_m} p_{im}d_{im}f_i | \sum_{i\in\mathcal{J}'_m} d_{im}f_i \ge \overline{\mu}_m(C_m - d^m_{max})]\\ &\ge \mathcal{A}_m(p^{cm}_*)(1 - O(\frac{2r_{max}(\overline{\mu}_m)}{\overline{\mu}_m + 2r_{max}(\overline{\mu}_m)} + \frac{d^m_{max}}{C_m})) - \theta_m n_m,\end{aligned} \tag{31}$$

where the last inequality holds due to Claim 1.2 and the definition of $\mu^U_m$. Combining (30) and (31), we can get the lower-bound of the expected profit obtained by our DPS for selling type-$m$ PSs:

$$\begin{aligned}\mathbb{E}[\mathcal{A}_m(\mathcal{L})] \ge \min\{&\mathcal{A}_m(p^{cm}_*)(1 - O(\frac{2r_{max}(\overline{\mu}_m)}{\overline{\mu}_m + 2r_{max}(\overline{\mu}_m)} + \frac{d^m_{max}}{C_m}))\\ &- \theta_m n_m, \mathcal{A}_m(\mathcal{L}') - O(\sqrt{n_m\log n_m})\}.\end{aligned}$$

Combining this inequality and Lemma 1, we have Lemma 2.

### E. Proof of Lemma 3

*Proof.* As shown in the initialization in Alg.1, the prices of type-$m$ PSs in the candidate set $\mathcal{P}_m$ are within the interval $[\delta_m, 1]$, *i.e.*, $p_m \in [\delta_m, 1], \forall p_m \in \mathcal{P}_m$. If $p^m_* \le \delta_m$, then we know $\mathcal{A}_m(p^m_*) - \mathcal{A}_m(p^{cm}_*) \le \delta_m C_m\overline{\mu}_m$. Let $p'_m$ denote the highest price in $\mathcal{P}_m$ that is no higher than the best fixed price $p^m_*$, which indicates $p'_m \ge p^m_*/(1+\delta_m)$. Hence, we have

$$\begin{aligned}&\sum_{m\in[M]}\mathcal{A}_m(p^{cm}_*) \ge \sum_{m\in[M]}\mathcal{A}_m(p'_m) \ge \sum_{m\in[M]}\mathcal{A}_m(p^m_*/(1+\delta_m))\\ &\ge \sum_{m\in[M]}\mathcal{A}_m(p^m_*)(1-\delta_m) \ge \sum_{m\in[M]}\mathcal{A}_m(p^m_*) - \sum_{m\in[M]}\delta_m C_m\overline{\mu}_m,\end{aligned}$$

where the last inequality holds because $Q_m(p_m)$ is a non-increasing function towards $p_m$. Combining the above inequality with Lemma 2, Lemma 3 is derived.

### F. Proof of Theorem 2

*Proof.* In the exploration stage in Alg.1, if the expected number of jobs requesting type-$m$ workers is denoted as $\Phi(\theta_m n_m)$, then we have $\Phi(\theta_m n_m) \ge \theta_m n_m - n_m\tau_{max}/T$. Further, we have $r_{max}(\overline{\mu}_m) \le O(\log n_m/(\tau_{max}\Phi(\theta_m n_m)))$. Due to $\tau_{max} \le (T^5 C^2_m \log^2 n_m)^{1/3}/(2n_m)$, we obtain $\Phi(\theta_m n_m) \ge (T^5 C^2_m \log^2 n_m)^{1/3}/2$. Moreover, we know $|\mathcal{P}|_m \le (\log n_m)/\delta_m$. Therefore, when $\delta_m = (TC_m)^{-1/3}(\log n_m)^{2/3}$ and $\sigma_m = \delta_m C_m\overline{\mu}_m/n_m$, according to Lemma 3, we have

$$\begin{aligned}&\sum_{m\in[M]}[\mathcal{A}_m(p^m_*) - \mathbb{E}[\mathcal{A}_m(\mathcal{L})]]\\ &\le \sum_{m\in[M]} O(\sqrt{n_m\log n_m} + \frac{\mu^U_m(TC_m\log n_m)^{\frac{2}{3}}}{\overline{\mu}_m} + C_m r_{max}(\overline{\mu}_m))\end{aligned} \tag{32}$$

$$\le O(\sum_{m\in[M]}\sqrt{n_m\log n_m} + (TC_m\log n_m)^{2/3}). \tag{33}$$

Inequality (32) holds since we assume $d^m_{max} \le T^{-1/3}(C_m\log n_m)^{2/3}$. Due to $r_{max}(\overline{\mu}_m) \le O(\frac{(1+\overline{\mu}_m)\log n_m}{1+\Phi(\theta_m n_m)})$, we know $C_m r_{max}(\overline{\mu}_m) \le (TC_m)^{2/3}(\log n_m)^{1/3}$. Moreover, $\mu^U_m/\overline{\mu}_m$ asymptotically approaches $O(1)$. Putting them together, the last inequality (33) can be established. Similarly, $\sum_{m\in[M]}[\mathcal{A}_k(p^k_*) - \mathbb{E}[\mathcal{A}_k(\mathcal{L})]]$ is derived. Thus, the regret $Regret(\mathcal{L})$ of our DPS algorithm is $O[(K+M)((N\log N)^{1/2} + (TC_{max}\log N)^{2/3})]$.

## References


[1] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Su, "Scaling distributed machine learning with the parameter server," in *Proc. of USENIX OSDI*, 2014.

[2] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing, "More effective distributed ML via a stale synchronous parallel parameter server," in *Proc. of NIPS*, 2013.

[3] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. of IEEE INFOCOM*, 2019.

[4] L. Mai, C. Hong, and P. Costa, "Optimizing network performance in distributed machine learning," in *Proc. of USENIX HotCloud*, 2015.

[5] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, "Firecaffe: near-linear acceleration of deep neural network training on compute clusters," in *Proc. of IEEE CVPR*, 2016.

[6] *Amazon EC2 Pricing*, 2019, https://aws.amazon.com/ec2/pricing/.

[7] *Google Cloud Pricing*, 2019, https://cloud.google.com/pricing/.

[8] *Linux Virtual Machines Pricing*, 2019, https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/.

[9] X. Zhang, C. Wu, Z. Huang, and Z. Li, "Occupation-oblivious pricing of cloud jobs via online learning," in *Proc. of IEEE INFOCOM*, 2018.

[10] W. Wang, B. Liang, and B. Li, "Revenue maximization with dynamic auctions in iaas cloud markets," in *Proc. of IEEE/ACM IWQoS*, 2013.

[11] 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/ACM IWQoS*, 2014.

[12] L. Mashayekhy, M. M. Nejad, D. Grosu, and A. V. Vasilakos, "An online mechanism for resource allocation and pricing in clouds," *IEEE Trans. Computers*, vol. 65, no. 4, pp. 1172–1184, 2016.

[13] J. R. Correa, P. Foncea, R. Hoeksma, T. Oosterwijk, and T. Vredeveld, "Posted price mechanisms for a random stream of customers," in *Proc. of ACM EC*, 2017.

[14] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[15] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

[16] M. Mahdavi, T. Yang, and R. Jin, "Efficient constrained regret minimization," *CoRR*, vol. abs/1205.2265, 2012.

[17] P. Minet, E. Renault, I. Khoufi, and S. Boumerdassi, "Analyzing traces from a google data center," in *Proc. of IEEE IWCMC*, 2018.

[18] R. Kleinberg, A. Slivkins, and E. Upfal, "Multi-armed bandits in metric spaces," in *Proc. of ACM STOC*, 2008.