Dynamic Pricing and Placing for Distributed Machine Learning Jobs: An Online Learning Approach

Ruiting Zhou, Xueying Zhang, John C. S. Lui, *Fellow, IEEE*, and Zongpeng Li

***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. 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 the cloud service 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 can increase the provider's profit by 200%, and achieves a sub-linear regret with both the time horizon and the total job number, compared to benchmark pricing schemes. Extensive evaluations using real-world data also validates the efficacy of DPS.**

***Index Terms*— Machine learning, dynamic pricing, online placement.**

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].

This work targets distributed ML jobs, and designs dynamic pricing and placement frameworks to efficiently utilize cloud resources and maximize cloud provider's profit. 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.

However, existing cloud pricing mechanism is not suitable for distributed ML jobs. Today's cloud 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

Manuscript received 9 April 2022; revised 5 September 2022; accepted 30 November 2022. Date of publication 13 February 2023; date of current version 17 March 2023. This work was supported in part by NSFC under Grant 62072344 and Grant U20A20177 and in part by the National Key Research and Development Program of China under Grant 2022YFB2901300. The work of John C. S. Lui was supported in part by RGC's GRF under Grant 14215722. *(Corresponding author: Zongpeng Li.)*

Ruiting Zhou is with the School of Computer Science Engineering, Southeast University, Nanjing 211189, China (e-mail: ruitingzhou@seu.edu.cn).

Xueying Zhang is with the School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China (e-mail: snowyzhang@whu.edu.cn).

John C. S. Lui is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: cslui@cse.cuhk.edu.hk).

Zongpeng Li is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: zongpeng@tsinghua.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSAC.2023.3242707.

Digital Object Identifier 10.1109/JSAC.2023.3242707

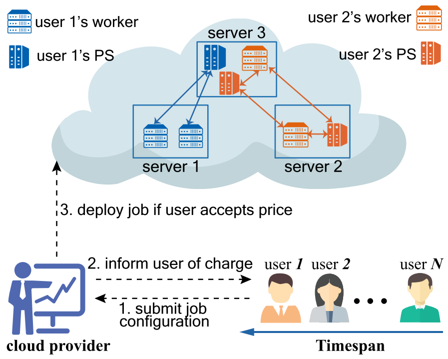


Fig. 1. An illustration of pricing and placement process.

plans and reserved instances [6]. Although above pricing strategies are simple to implement, they cannot be applied to distributed ML jobs directly, due to following reasons. *First*, different users have different budgets with heterogeneous demands. Fixed pricing fails to attract 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 the cloud service provider 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?*

In order to complement the existing cloud pricing models, we propose a novel mechanism, DPS, which integrates dynamic pricing and placement 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. We employ a multi-armed bandit (MAB) framework to design a two-phase algorithm, which is correlated in jointly obtaining pricing and placement decisions. If a user accepts the offered price in the first phase, the cloud provider deploys this job on its servers to minimize job runtime. The algorithm calculates the reward based on job runtime (phase two), to ensure that shorter runtime brings higher profit (phase one). The detailed technical contributions are as follows:

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, which makes it difficult to dynamically charge each job for higher profit. To tackle this issue, we design an online learning strategy based on the MAB framework, DPS. 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, we aim to reduce the time for parameter transmission among different physical servers. We propose a placement algorithm, PA, which deploys 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. In addition to the classic placement model, a more complex placement model is considered and discussed, where the bandwidth allocation is dynamic. We propose a new placement algorithm, UPA, to address the dynamic bandwidth allocation problem. The performance of two placement algorithms are studied and compared in simulations.

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. Furthermore, we demonstrate the performance of DPS by comparing it with existing state-of-the-art algorithms through extensive simulations. 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 and the extension to dynamic allocation of bandwidth are presented in Sec. IV and Sec. V, respectively. Simulations are conducted in Sec. VI. Sec. VII concludes the paper.

II. Related Work

A. 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]. Wang et al. [13] study how to minimize the cost of service providers by choosing different pricing options based on jobs' demands, which charges a job according to its priority (whether it is latency-sensitive). Baek et al. [14] analyze three dynamic pricing mechanisms for resource allocation of edge computing for IoT environment. Omotehinwa et al. [15] design a dynamic strategy-proof algorithm to price cloud services, which relies on the market history to forecast a benchmark price for ensuring truthful valuation from users. Those pricing strategies either focus on posted price mechanism [16] 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. Wang et al. [17] design an algorithm to price cloud resources for users according to their demands, the system state and queue length, which does not take the resource occupation time into account. Nambiar et al. [18] propose a "random price shock" algorithm that estimates price elasticity, while maximizing revenue. A pricing model for virtual cloud providers is presented in [19] to dynamically derive the energy costs per allocation unit and per work unit for each time period. Those pricing methods are not applied to our work, where the resource capacity should be carefully considered. An occupation-oblivious pricing method for cloud jobs is proposed in [9]. Yet, our work focuses on a completely different problem. First, we address distributed ML jobs based on the parameter server (PS) framework, in which users can request diverse combinations of workers and PSs. In addition, Zhang et al. in [9] assume that the runtimes of all type-k jobs (*i.e.,* running type-k VMs) are regarded as i.i.d, which may not be feasible. Second, the characteristic of PS framework is not captured by it, and our proposed algorithm has a different structure. Specifically, we design a two-phase algorithm, which is correlated in jointly obtaining pricing and placement decisions. If a user accepts the offered price in the first phase, the cloud provider deploys this job on its servers to minimize job runtime. The algorithm calculates the reward based on job runtime (phase two), to ensure that shorter runtime brings higher profit (phase one).

B. 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 [20], [21]. In an MAB problem, there is a set of arms and each of them has a reward (unknown before the arm is pulled). In each round, one arm is selected and its reward is revealed. Hence, MAB framework is usually adopted to achieve the maximal cumulative reward in the long term (*i.e.,* multiple rounds). Bubeck et al. [22] 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. Besbes et al. [23] complement this literature by developing a flexible non-parametric model for temporal uncertainty in the rewards. A dynamic bandit algorithm is proposed in [24] to improve the shortcoming of the basic MAB framework for quickly detecting trends early enough. Mahdavi et al. [25] extend the study of MAB where the learner aims to maximize total reward, given that some additional constraints need to be satisfied. However, they are not applicable to our system where the resources can be reused after a job is completed. Moreover, in our problem, resources are shared by jobs with different arrival times, which means our online decisions made at different times need to meet the global resource capacity constraints.

C. Distributed Machine Learning Systems

A considerable amount of research about ML system has been done during the last decade. Chen et al. [26] study and propose a distributed intelligent video surveillance system using deep learning algorithms and deploy it in an edge computing environment. They address the problems of parallel training, model synchronization and workload balancing. A parallel random forest algorithm is presented for big data in a spark cloud computing environment [27]. Some studies [28], [29], [30] focus on the resource allocation in distributed ML system. Ghodsi et al. [28] propose a fair allocation strategy of multiple resource types. An elastic ML framework is designed in [29], which computes the optimal number of nodes for a ML job to optimize its total runtime. Chen et al. [30] propose a performance-aware fair scheduler to identify resource demand elasticity, which is used to improve the average job performance. Some researchers [31], [32] [33], [34] focus on ML job scheduling and placement. Xu et al. [31] design a reinforcement learning-based job scheduling algorithm to minimize the cost of big data analytics on geo-distributed data centers. Mirhoseini et al. [32] propose a method to optimize device placement for TensorFlow computational graphs. Azar et al. [33] study online mechanisms for preemptive scheduling with deadlines. A scheduler based on reinforcement learning for a multi-resource cluster is proposed in [34] to address the diversity and heterogeneity of jobs and machines in modern cluster environments. Moreover, Chen et al. [35] introduce task decomposition and scheduling strategies with the objectives of thread-level load balancing and minimum waiting time for critical paths. Different from above work, we combine dynamic pricing and online placement to pursue the cloud provider profit maximization in the long term.

D. Placement, Scheduling and Task Offloading

In recent years, there has been an increasing amount of literature on the job placement, job scheduling and task offloading. Li et al. [36] study an ML broker service aggregating geo-distributed ML jobs into cloud data centers for volume discounts via dynamic online placement and scaling of workers and parameter servers for long-term cost minimization. In [37], a job scheduler is proposed based on a dynamic grouping integrated neighboring search strategy, which can balance the resource utilization and improve the performance and data locality in heterogeneous computing environments. Moreover, Lee et al. [38] propose a novel framework for online fog network formation and task distribution in a hybrid fog-cloud network. They have addressed the problem using an online optimization formulation whose goal is to minimize the maximum latency of the nodes in the fog network in presence of uncertainty about fog nodes' arrivals. A a probablistic user selection scheme is studied and proposed in [39], which jointly considers user selection and resource allocation for the minimization of federated learning convergence time while while optimizing the performance. In our work, we not only consider the placement of distributed ML jobs, but also integrate it with pricing. Based on this, an online learning algorithm is proposed.

III. Problem Model

A. 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 $\{1, 2, \ldots X\}$. 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 $t = 1, 2, \ldots, T$. 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 α_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:

$$B_i = \{k_i, d_{ik}, m_i, d_{im}, w_i, \alpha_i\}.$$

B. 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 $F_{k,m}$.

C. 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 β_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. We assume that each type-k worker (type-m PS) reserves bandwidth, and the amount of the reserved bandwidth is denoted as h_k (H_m). Hence, $b_i = \min(h_{k_i}, H_{m_i}/d_{ik})$. Note that the bandwidth b_i is simplified to be static here, and the dynamic allocation of bandwidth is discussed in Sec. V. 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:

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

D. 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.

E. 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):

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

$$\text{subject to: } f_i = \mathbb{1}\{d_{ik} + y_k^i \leq C_k, d_{im} + y_m^i \leq C_m,$$

TABLE I
SUMMARY OF NOTATIONS

K	# of worker types	M	# of PS types
T	system timespan	N	# of jobs
t_i	arrival time of job i	τ_i	runtime of i
w_i	parameter size of i	v_i	budget of i
$[X]$	set $\{1, 2, \cdots, X\}$	S	# of servers
$C_k(C_m)$	# of type-k worker (type-m PS)		
$k_i(m_i)$	type of worker (PS) requested by job i		
$d_{ik}(d_{im})$	the resource demands of i		
α_i	# of training epochs of job i		
β_i	the computation time of i in one epoch		
$p_{ik}(p_{im})$	the unit price of k_i worker (m_i PS) for i		
$\mathcal{X}_{ski}(\mathcal{Z}_{smi})$	# of idle type-k_i worker (type-m_i PS) of server s upon i's arrival		
$x_{ski}(z_{smi})$	# of k_i workers (m_i PSs) serving i on s		
$h_k(H_m)$	the bandwidth of type-k worker (type-m PS)		
b_i	$\min(h_{k_i}, H_{m_i}/d_{ik})$		

$$\sum_{k\in[K]} p_{ik}d_{ik} + \sum_{m\in[M]} p_{im}d_{im} \leq v_i, \quad \forall k,\ \forall m\}, \tag{2a}$$

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

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

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

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

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

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

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

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

$\mathbb{1}\{X\}$ 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. In constraint (2b)/(2c), we use j to refer jobs that are still running upon the arrival of job i. 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).

Theorem 1: MILP (2) is NP-hard.

Proof: When we delete variables z_{smi}, p_{ik}, p_{im} and their corresponding constraints and let $f_i = 1$, $d_{jk} = 1$ in MILP (2), the simplified MILP (2) is the classical knapsack problem which is NP-hard [40]. Therefore, we can reduce the classical knapsack problem to MILP (2), and MILP (2) is NP-hard. □

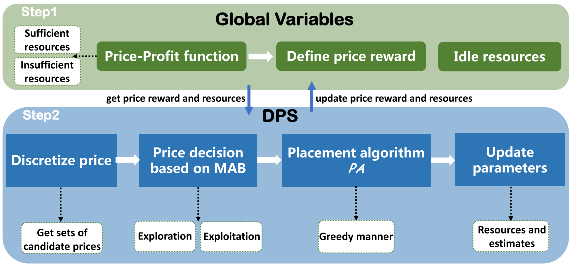


Fig. 2. An illustration of our algorithm structure.

F. Challenges

MILP (2) is an NP-hard problem even in the offline setting. In addition, the budgets of jobs are unknown to the cloud provider. Furthermore, when a job arrives, its runtime is stochastic, which is not only related to the resource configuration but also related to the placement of workers and PSs. To overcome these challenges, we design an online learning-based algorithm to integrate dynamic pricing and the placement of jobs for profit maximization.

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. Related discussion and the theoretical analysis are presented in Sec. IV-B and Sec. IV-C respectively.

Main Idea: Fig. 2 shows the main structure of our algorithm including two main steps. At step 1, we analyze the relationship between the total profit and prices of resources. Then, a *Price-Profit* function is derived. Based on this function, the reward of each price is defined. These variables will be used in next step. As you can see, step 2 is our learning-based algorithm (DPS), which is the core of this work. In the high level, DPS consists of the following parts: (i) discretizing price stage; (ii) price decision stage; (iii) placement algorithm; (iv) updating parameters stage. In the discretizing stage, the price interval is discretized and a set of candidate prices for each type worker (PS) is obtained. Each price is associated with a reward from step 1. In the price decision stage, a price is selected from the candidate set for the current job based on MAB framework, which makes a good balance between the exploration and exploitation phase. If the user accepts the price, a placement algorithm PA is involved to deploy the job. Furthermore, the reward of the selected price and the information of idle resources (workers and PSs) are updated, *i.e.,* global variables in step 1 are updated. Based on the online learning strategy, prices are dynamically offered to users according to the learned distribution of users' budgets.

A. Algorithm Design

1) Dynamic Pricing Mechanism:

a) 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 DPS based on UCB (Upper Confidence Bound) to dynamically determine prices. Specifically, DPS learns the runtime and the distribution of users' budgets according to past jobs, and sets prices for arriving jobs based on the learned knowledge.

Without loss of generality, p_k and p_m are normalized 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 τ_i slots, then if its workload is averaged over T slots, job i runs τ_i/T slot in each slot). Thus, the average maximum total profit of type-k workers with a fixed price p_k is $p_k C_k(T - \mathbb{E}_{i:k_i=k}[\tau_i])$. Let $\mu_i = T - \tau_i$. Then, the 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

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

To maximize the long-term profit, the prices that maximize the upper-bound, *i.e.,* RHS of (3), should be set. 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*. At the beginning, the price interval $[0,1]$ is discretized, and DPS gets 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 $p_k \in \mathcal{P}_k$ ($p_m \in \mathcal{P}_m$), there is an associated reward contributing to the overall profit. The prices with the highest reward are selected for the current job. The reward of price is defined as follows:

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

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

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; μ^U_{ik} (μ^U_{im}) is the UCB of $\overline{\mu}_k$ ($\overline{\mu}_m$) estimated before job i arrives, as defined below:

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

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

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

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:

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

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

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

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

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. The confidence radius[1] is:

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

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

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

b) Online pricing algorithm: Our dynamic pricing strategy, DPS, is summarized in Alg. 1. In the initialization phase, DPS elaborately designs δ_k and δ_m to discretize the prices interval and gets sets of candidate prices. Note that parameters δ_k and δ_m have a significant impact on our algorithm performance, and we will illustrate this impact in Sec. VI. Inspired by the trade-off between *exploration* and *exploitation* in classic MAB framework, nil prices are set for jobs at the

[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.

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** $\sum_{i'=1}^{i} \mathbb{1}(k_{i'} = k) \leq \theta_k n_k$ or $\sum_{i'=1}^{i} \mathbb{1}(m_{i'} = m) \leq \theta_m n_m$ **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** $d_{ik} + y_k^i \leq C_k$ and $d_{im} + y_m^i \leq C_m$ **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 τ_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 (2d)-(5), 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 (2i)(3);

beginning stage (lines 2-6). Here, "nil prices" means that the unit price of the resource is temporarily set to zero such that users can accept the price. In the beginning, we have no any knowledge about users' budgets and the runtime of different distributed ML jobs. Therefore, we set zero price to make sure that each user will accept the deal in this stage, so that DPS can get more information about system environment and make more correct estimation as soon as possible. This is conducive to improving the learning rate of our algorithm. The smaller θ_k and θ_m, the shorter is the exploration time. Hence, parameters θ_k and θ_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, θ_k and θ_m are derived carefully to reach a good balance between them. After the exploration phase, DPS 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 τ_i is approximately calculated in line 14. Meanwhile, the estimated parameters $Q_{ik}^U(p_k), Q_{im}^U(p_m), \mu_{ik}^U$ and μ_{im}^U (lines 14-18) as well as the amount of occupied resources are updated, which will be used to calculate the rewards of prices when the next job arrives. Once a job is completed and the occupied resources are released, DPS updates parameters (*i.e.*, $\hat{\mu}_{ik}$ and $\hat{\mu}_{im}$) related to the exact runtime.

2) Placement Policy: The correlation between DPS and the placement algorithm can be confirmed by Eq. (2a). Specifically, we design a two-phase algorithm, which is correlated in jointly obtaining pricing and placement decisions. If user i accepts the offered price in the first phase, the cloud provider deploys this job on its servers to minimize job runtime. The algorithm calculates the reward based on job runtime (phase two), to ensure that shorter runtime brings higher profit by determine pricing strategy (phase one). The placement problem for job i can be formulated as:

$$\begin{aligned} &\text{minimize } \tau_i \\ &\text{subject to: constraints } \quad (2d) \sim (2h), \end{aligned} \tag{15}$$

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. Discussion

Note that parameters n_k and n_m are used as input in our Alg. 1, which are not known in the online setting. However, they can usually be estimated by studying and analyzing the historical arrived jobs. Namely, we estimate n_k (n_m) by calculating the empirical arrival rate of jobs needing type-k worker (type-m PS) based on history and multiplying the arrival rate by T. In practice, it is found that this estimation method can make relatively accurate predictions, and the difference between estimated values and actual values does not have a big impact on the length of the exploration period in simulations. Moreover, a set of candidate prices in our pricing mechanism are provided to choose. Therefore, the optimal prices of workers and PSs may not fall into our sets. There maybe a gap between the profit achieved by DPS and the maximum profit. The gap is analyzed in the following subsection.

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: $\boldsymbol{x}_{ki} = \mathbf{0}, \boldsymbol{z}_{ki} = \mathbf{0}, c = 1$

Sort all servers in descending order of $\mathcal{X}_{ski}$ and $\mathcal{Z}_{smi}$, the result sequence is denoted as $\{s_1, s_2, \cdots, s_S\}$;
for $s = s_1, s_2, \cdots, s_S$ **do**
  if $\mathcal{X}_{ski} \geq d_{ik}$ and $\mathbb{Z}_{smi} \geq d_{im}$ **then**
    /* Deploy all workers and PSs on server s */
    $x_{ski} = d_{ik}, z_{smi} = d_{im}$;
    /* Update current idle resources */
    $\mathcal{X}_{sk(i+1)} = \mathcal{X}_{ski} - d_{ik}, \mathcal{Z}_{sm(i+1)} = \mathcal{Z}_{ski} - d_{im}$;
    Return $\boldsymbol{x}_{ki}, \boldsymbol{z}_{ki}$
  end if
end for
/* Multiple servers are used */
while $\sum_{j=1}^{c} \mathcal{X}_{s_j ki} < d_{ik}$ **do**
  $x_{s_c ki} = \mathcal{X}_{s_c ki}, \mathcal{X}_{s_c k(i+1)} = 0$;
  $c = c + 1$;
end while
$x_{s_c ki} = d_{ik} - \sum_{j=1}^{c-1} \mathcal{X}_{s_j ki}, \mathcal{X}_{s_c k(i+1)} = \mathcal{X}_{s_c ki} - x_{s_c ki}$;
$c = 1$;
while $\sum_{j=1}^{c} \mathcal{Z}_{s_j mi} < d_{im}$ **do**
  $z_{s_c mi} = \mathcal{Z}_{s_c mi}, \mathcal{Z}_{s_c m(i+1)} = 0$;
  $c = c + 1$;
end while
$z_{s_c mi} = d_{im} - \sum_{j=1}^{c-1} \mathcal{Z}_{s_j mi}, \mathcal{Z}_{s_c m(i+1)} = \mathcal{Z}_{s_c mi} - z_{s_c mi}$;
Return $\boldsymbol{x}_{ki}, \boldsymbol{z}_{ki}$

C. Theoretical Analysis

1) Runtime: First, we analyze the time complexity of DPS. We prove that DPS runs in polynomial time.

Theorem 2: DPS 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. □

2) Regret Analysis: In order to verify the efficiency of our algorithm, now we theoretically analyze the *regret* of DPS, which is an important evaluation criterion for measuring algorithms. 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]. Hence, the *regret* denotes the difference between the expected overall profit obtained by our algorithm and that by the best fixed-price strategy. Theorem 3 below shows that the regret of DPS is *sub-linear* with both the timespan and the total job number, which means our algorithm is learning the best pricing strategy from the history as time goes on or the number of jobs increases. When T or N is large enough, DPS has an asymptotically optimal performance. Theorem 3 indicates that our algorithm can provide the best pricing policy and bring the highest profit for the cloud service provider.

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

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:

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

where $\mathcal{A}(\mathcal{L})$ is the total expected profit achieved by DPS. Combining (16) with (3), we can find that $Regret(\mathcal{L})$ is not only affected by resource prices, but also related to users' bugdgets. However, this information is private for us and we can not know it in advance. Hence, it's difficult to directly get the exact value of $Regret(\mathcal{L})$. Therefore, we derive the upper bound of it by decomposing the process. Specifically, the derivation process of regret $Regret(\mathcal{L})$ is divided into the following 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 3: 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:

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

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

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

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

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, we have a straightforward corollary as shown in Claim 2, which implies that the profit achieved by selling type-m PSs with our algorithm DPS is at least $\mathcal{A}_m(p_*^{cm})$ if the actual sold number of type m PSs is at least $\max_{i\in[n_m]} C_m \bar{\mu}'_m$. This conclusion will be used later.

Claim 2: Let p_{im} denote the price for job i designed by our algorithm without considering the resources capacity. We have

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

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

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

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 3: For each job i, we have $\Delta(p_i m) \leq p_{im} \cdot O(r_i(\hat{Q}_{im}(p_m)))$. Furthermore, we have

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

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 2 and Claim 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

$$\begin{aligned} &\mathcal{A}_m(p_*^{cm}) - \mathbb{E}[\mathcal{A}_m(\mathcal{L})] \\ &\quad \leq \sigma_m n_m + \theta_m n_m + |\mathcal{P}_m| O(\log n_m)(1 + C_m \mu_m^U / (\sigma_m n_m)) \\ &\qquad + O[\sqrt{n_m \log n_m} + C_m \overline{\mu}_m (\frac{2r_{max}(\overline{\mu}_m)}{\overline{\mu}_m + 2r_{max}(\overline{\mu}_m)} + \frac{d_{max}^m}{C_m})]. \end{aligned}$$

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

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

Proof: See Appendix. E. □

Finally, we prove Theorem 3 based on Lemma 3, as shown in Appendix. F. □

V. Extension to Dynamic Allocation of Bandwidth

In this section, we extend the static bandwidth allocation in Sec. III to dynamic bandwidth allocation, where the minimum transmission bandwidth between PSs and workers is not only relevant to the resource configuration but also to the way of current job's placement. The transmission time in the dynamic allocation of bandwidth model is analyzed in Sec. V-A and the placement policy is presented in Sec. V-B.

A. Transmission Time

Since parameters are usually evenly distributed on PSs in the PS architecture, the size of gradients/parameters transmitted between a PS and a worker is w_i/d_{im}. If a PS and a worker are located on different servers, the data transmission time between them in an epoch is $w_i/(d_{im} \cdot b_i)$, where b_i is the minimal bandwidth between a worker and a PS among all servers. Hence, $b_i = \min_{s\in[S]}\{h_{ki}/(d_{im} - z_{smi}), H_{mi}/(d_{ik} - x_{ski})\}$, where $d_{im} - z_{smi}$ is the number of PSs which are not placed on server s. So the bandwidth that is reserved for a type-k_i worker on server s, h_{ki}, is shared by all remote PSs, to communicate between each PS-worker pair. The bandwidth allocated to each remote PS is $h_{ki}/(d_{im} - z_{smi})$. Similarly, $H_{mi}/(d_{ik} - x_{ski})$ is the bandwidth between each remote worker and a type-m_i PS on server s. Therefore, the runtime of job i can be denoted as:

$$\tau_i = \alpha_i \beta_i + \frac{2\alpha_i(1 - q_i)w_i}{d_{im} \cdot b_i}. \tag{22}$$

Then, if the offered price for job i is accepted by the user, *i.e.*, $f_i = 1$, its placement problem (ILP (15) in Sec. IV) can be reformulated as:

$$\text{minimize } \alpha_i \beta_i + \frac{2\alpha_i w_i}{d_{im} b_i} \tag{23}$$

$$\text{subject to: } b_i \leq \frac{h_{ki}}{d_{im} - z_{smi}}, \quad \forall s \in [S], \tag{23a}$$

$$b_i \leq \frac{H_{mi}}{d_{ik} - x_{ski}}, \quad \forall s \in [S], \tag{23b}$$

$$(2d) \sim (2h).$$

B. Placement Algorithm

As shown in (23), this is a Min-Max Regret Integer Linear Programming problem (MMR-ILP). Since the min-max regret has, at least, the same complexity of its deterministic counterpart [41], MMR-ILP is also an NP-hard optimization problem. To minimize the runtime of jobs, apart from the PA proposed in Sec. IV, we design another placement policy UPA as shown in Alg. 3. The core idea is to use the smallest number of servers to host the job, such that the number of PSs (workers) are deployed on each of these servers is as equal as possible. The performance evaluation of UPA and PA is presented in Sec. VI-C. The simulations show that UPA performs better when resources are scarce, and PA has better performance with sufficient resources.

Algorithm 3 Uniform Placement Algorithm (UPA)

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: $\boldsymbol{x}_{ki} = \mathbf{0}, \boldsymbol{z}_{ki} = \mathbf{0}, c = 1$

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** $\mathcal{X}_{ski} \geq d_{ik}$ and $\mathbb{Z}_{smi} \geq d_{im}$ **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: /* Place job on multiple servers uniformly */
12: **while** $\sum_{j=1}^{c} \mathcal{X}_{s_j ki} < d_{ik}$ **do**
13:     $c = c + 1$;
14: **end while**
15: **for** $c' = c, c-1, \cdots, 1$ **do**
16:     $x_{s_{c'}ki} = \lfloor d_{ik}/c \rfloor$;
17:     $\mathcal{X}_{s_{c'}k(i+1)} = \mathcal{X}_{s_{c'}ki} - x_{s_{c'}ki}$;
18: **end for**
19: $c = 1$;
20: **while** $\sum_{j=1}^{c} \mathcal{Z}_{s_j mi} < d_{im}$ **do**
21:     $c = c + 1$;
22: **end while**
23: **for** $c' = c, c-1, \cdots, 1$ **do**
24:     $z_{s_{c'}mi} = \lfloor d_{im}/c \rfloor$;
25:     $\mathcal{Z}_{s_{c'}m(i+1)} = \mathcal{Z}_{s_{c'}mi} - z_{s_{c'}mi}$;
26: **end for**
27: **Return** $\boldsymbol{x}_{ki}, \boldsymbol{z}_{ki}$

Similarly, we first sort all servers in descending order of their current resource availability (available PSs and workers). Then our algorithm determines whether there exists a server having enough resources to serve the current job (lines 1-8). If there is a such server, it is deployed to run the job. Otherwise, the job is placed on multiple servers uniformly (lines 12-27). For each job, we check whether the resources on the first c servers are sufficient to serve the job (starting with $c = 1$) as shown in lines 12-14 and lines 19-22. If so, we place parameter servers and workers in the job evenly on the c servers (lines 15-18); otherwise, we check the first $c+1, c+2, \cdots$ servers until enough servers are found to place the job. Then current idle resources of servers are updated.

VI. Performance Evaluation

To evaluate the performance of DPS, we conduct extensive numerical simulations based on real-world data. The simulation setup and four benchmark algorithms are introduced in Sec. VI-A, and the simulation results are analyzed in Sec. VI-B.

A. 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 [42]. 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 $[0, 1]$.

Benchmarks: 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. The best fixed price is the fixed prices (*i.e.,* p_k, p_m) which maximize the right hand side of (3) (overall profit), and also consider the available resource capacity as shown in (3). In the offline setting (assuming the distribution of users' budgets and the average of jobs' runtime are known in advance), the best fixed prices are computed by Nelder-Mead Simplex algorithm [43] based on RHS of (3).
- *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.

B. Simulation Result

1) Performance Metric: Our performance metric is the *regret* of algorithms, which is the cumulative profit of *BFP* minus the cumulative profit of the algorithm.

Fig. 3 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

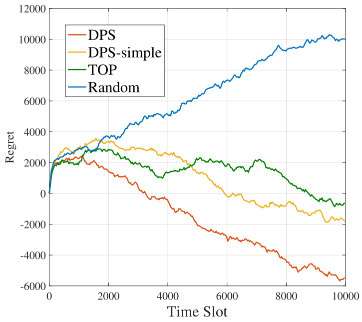


Fig. 3. The regret comparison with other algorithms.

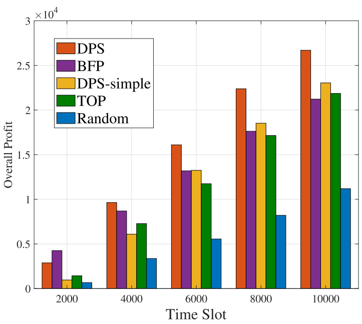


Fig. 4. Overall profit comparison with other algorithms.

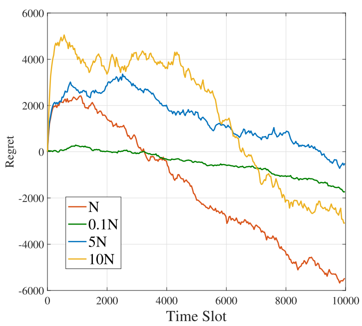


Fig. 5. Regret of DPS under different parameter N.

between users' budget and demands is unknown. Note that this growth curve is concave at the beginning, which implies that our algorithm is gradually learning from the past and makes better pricing decisions. 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. 4. 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.

2) The Impact of Parameters: Next, we investigate the impact of different 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. 5. At the beginning, the

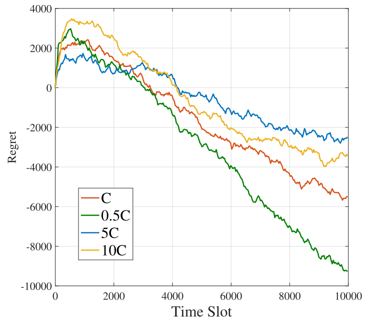


Fig. 6. Regret of DPS while C_k and C_m varying.

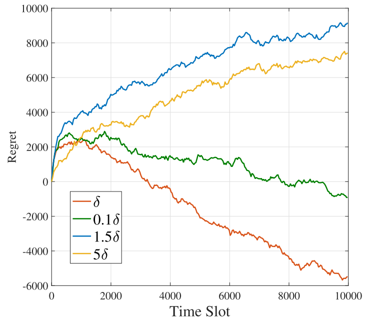


Fig. 7. Regret of DPS while δ_k and δ_m varying.

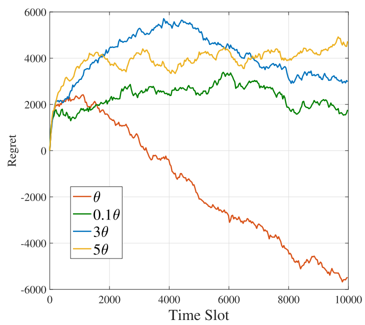


Fig. 8. Regret of DPS while θ_k and θ_m varying.

regret is smaller when N is smaller. Intuitively, a smaller N indicates a lower job arrival rate per time slot, which results in a smaller regret. As time goes on, the larger N, the faster the regret decreases. Similarly, Fig. 6 shows the regret of DPS under different resource capacities ($0.5C$, $5C$ and $10C$).[3] The smaller C, the smaller the regret is, which means that our algorithm performs better when resources are more scarce. These are consistent with our Theorem 3.

Fig. 7 shows the effect of the value of δ (*i.e.,* δ_k and δ_m) on DPS's regret. When δ'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 δ is too large, the regret is growing. It is shown that the regret obtained by our choice of δ is the smallest. Then, we analyze the impact of θ's value on the performance of DPS. As shown in Fig. 8, when θ gets smaller,

[3]Here, $0.5C$ means $0.5C_k$ and $0.5C_m, \forall k \in [K], m \in [M]$.

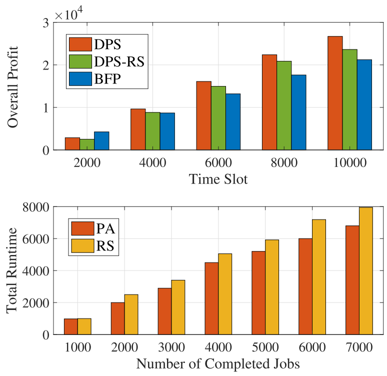


Fig. 9. The performance of placement strategy.

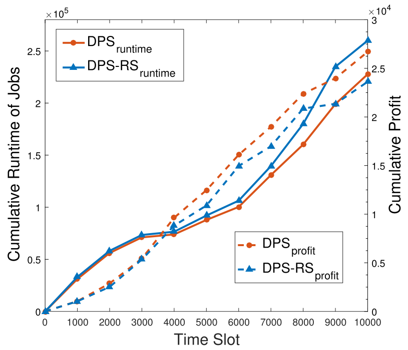
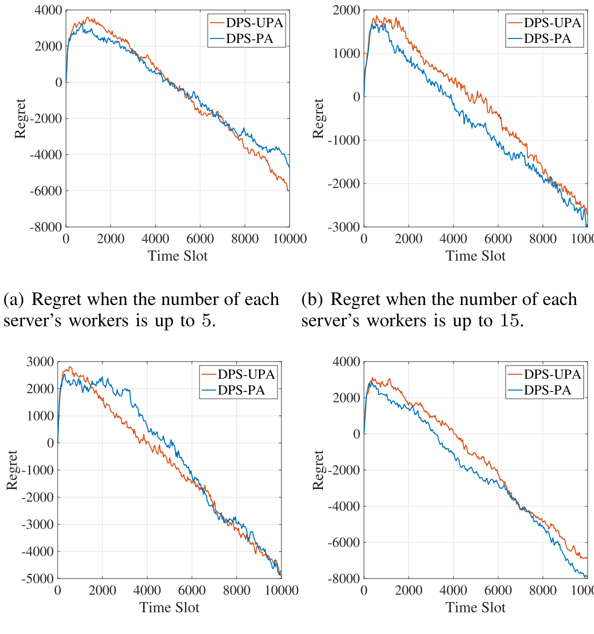


Fig. 10. The cumulative runtime and profit of jobs as time goes.

the regret is smaller at the beginning stage but it decreases more slowly in the later. A larger θ makes the exploration phase longer and leads to a larger regret in this phase. Our choice of θ shows a good trade-off between the exploration and exploitation.

3) 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. 9, 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. Moreover, combining Fig. 4 and Fig. 9, we know dynamic pricing strategy usually performs better than static pricing method.

To present the influence of DPS on jobs' runtime, Fig. 10 shows the cumulative runtime and cumulative profit of DPS and *DPS-RS* as time goes. In the initial phase, the difference between cumulative runtime of completed jobs caused by DPS and *DPS-RS* is relatively small. However, we can find that cumulative runtime of DPS is far shorter than that of *DPS-RS* with the arrival of subsequent jobs. It means that our algorithm DPS has good effect in minimizing the overall runtime of



(a) Regret when the number of each server's workers is up to 5.

(b) Regret when the number of each server's workers is up to 15.

(c) Regret when the number of each server's workers is up to 20.

(d) Regret when the number of each server's workers is up to 30.

Fig. 11. Regrets of UPA and PA under different resources intervals.

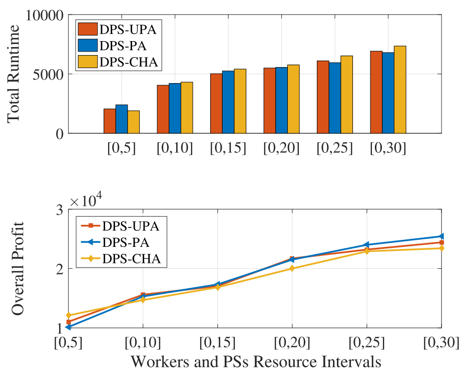


Fig. 12. The performance of UPA, PA and PA*CHA* under different resources capacity.

jobs. Meanwhile, observing the curves about runtime as well as profit, it shows that shorter runtime brings higher profit for cloud provider.

C. Simulation for Dynamic Bandwidth Allocation

We compare three placement algorithms: UPA in Alg. 3, PA in Alg. 2, and consistent hashing algorithm (CHA) [44]. Here, CHA is a cloud-optimized consistent hashing algorithm that takes the advantage of cloud environments: scaling management and auto-healing. First, CHA performs hash according to the type and amount of resources requested. If the server's resources matched by the hash are insufficient, the next server will be selected.

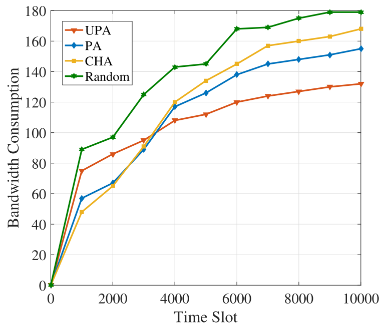


Fig. 13. The bandwidth consumption of UPA, PA, CHA and Random as time goes.

Fig. 11 analyzes the regret while resource capacity of each server changes. As shown in Fig. 11(a), when the number of each server's workers is up to 5, *i.e.,* $\mathcal{X}_{sk0} \in [0, 5]$, the regrets of UPA and PA in the early stage are similar. After $t = 890$, UPA's regret reduces faster. And the regret of UPA is smaller than PA's. This implies that UPA performs better than PA when resources are scarce, which can also be validated in Fig. 12. In Fig. 12, the number of each type worker deployed on each server, *i.e.*, $\mathcal{X}_{sk0}$, is in the range of $[0, 5]$, $[0, 15]$, $[0, 20]$ and $[0, 30]$ respectively. Correspondingly, the number of each type PS deployed on each server ranges in $[0, 3]$, $[0, 9]$, $[0, 12]$ and $[0, 18]$ respectively. When $\mathcal{X}_{sk0} \in [0, 5]$ and $\mathcal{X}_{sk0} \in [0, 10]$, the total runtime of processed jobs under PA is shorter than UPA. Similarly, profit obtained by the former is higher than the latter. Combining Fig. 11(b), Fig. 11(c) and Fig. 12, we find that the performance of UPA and PA is comparable when the amount of available resource slightly increases. When the total available resource continues to increase, PA has less regret and achieves higher profit, as shown in Fig. 11(d) and Fig. 12. This indicates that PA performs better with sufficient resources. In addition, compare to CHA, both UPA and PA achieve smaller profits at the beginning. Nevertheless, after that, their profits are larger than that of CHA.

Finally, we analyze the bandwidth consumption of UPA, CHA, CHA and Random algorithm. Fig. 13 shows their bandwidth consumptions at different time slots, and the unit is GB. At the beginning, UPA's bandwidth consumption is slightly larger than that of PA and CHA. However, as time goes on, it consumes smaller bandwidth than other algorithms. Moreover, the bandwidth consumption of UPA becomes more and more stable, which indicates that it provides an efficient placement strategy.

VII. 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 proposed algorithm estimates unknown input by leveraging a multi-armed bandit online learning framework, and calculates rewards based on feedback of job runtime. 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 2

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 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. If user accepts the price, the algorithm updates parameters (Lines 15-18), which can also be operated in a constant time. Hence, upon job i arrives, our algorithm can produce price for it in a polynomial runtime, *i.e.*, $O(T^{1/3}(C_{k_i}^{1/3} \log^{1/3} n_{k_i} + C_{m_i}^{1/3} \log^{1/3} n_{m_i}))$. Next, we analyze the runtime of placement algorithm *PA* in Alg. 2. In the worst case, the sorting time (line 1) is $O(S^2)$. And the rest of the algorithm (lines 2-23) can be completed in $O(S)$ time. 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

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

$$\hat{R}_{im}(p_{im}) \geq \hat{R}_{im}(p_*^{cm}), \quad \forall p_{im}, p_*^{cm} \in \mathcal{P}_m. \tag{24}$$

Moreover, 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 [20]. 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

$$\Pr[\hat{R}_{im}(p_{im}) \geq \mathcal{A}_m(p_*^{cm})] \geq 1 - n_m^{-2}, \quad \forall p_{im}, p_*^{cm} \in \mathcal{P}_m. \tag{25}$$

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

$$\hat{R}_{im}(p_{im}) \leq p_{im} \cdot \min\{n_m Q_m'(p_m), C_m \bar{\mu}_m'\} \tag{26}$$

with probability at least $1 - n_m^{-2}$. Combining (25) and (26), the Claim 1 follows.

C. Proof of Claim 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

$$Q_m(p_{im}) < C_m \bar{\mu}_m' / n_m, \quad \forall m \in [M]. \tag{27}$$

There is an attractive property behind (27): if the profit achieved by the best candidate price is larger than the expected profit obtained by our DPS without considering the resources capacity, the expected sold amount under our strategy must be relatively small. Let $\mathcal{J}_m$ denote the set of jobs requesting type-m PSs. According to Claim 1, we know

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

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,

$$\Delta(p_i m) \leq p_{im} \cdot O(r_i(\hat{Q}_{im}(p_m))). \tag{28}$$

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 [20], 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

$$r_i(\hat{Q}_{im}(p_m)) \leq \max\{\frac{O(\log n_m)}{1+N_i^m(p_m)}, \sqrt{\frac{Q_m(p_m)O(\log n_m)}{1+N_i^m(p_m)}}\}, \tag{29}$$

$$r_i(\hat{\mu}_{im}) \leq \max\{\frac{O(\log n_m)}{1+\sum_{i'<i:m_{i'}=m} f_{i'}\mathbb{1}(t_{i'}+\tau_{i'}<t_i)}, \sqrt{\frac{\overline{\mu}_m O(\log n_m)}{1+\sum_{i'<i:m_{i'}=m} f_{i'}\mathbb{1}(t_{i'}+\tau_{i'}<t_i)}}\}. \tag{30}$$

Combining (28) (29) with (27), for all job $i \in \mathcal{J}_m$, we get

$$\Delta(p_{im}) \leq O(p_{im} \cdot \max(\frac{\log n_m}{1+N_i(p_{im})}, \sqrt{\frac{\log n_m Q_m(p_{im})}{1+N_i(p_{im})}})). \tag{31}$$

Removing its dependency on i and rearranging this inequality, the Claim 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 $\sum_{i\in\mathcal{J}_m} |p_{im} d_{im} f_i - p_{im} Q_m(p_{im})| \leq O(n_m \log n_m)$ holds with high probability at least $1 - n_m^{-2}$. Hence, we have

$$\mathbb{E}(\sum_{i\in\mathcal{J}_m} p_{im} d_{im} f_i) \geq \sum_{i\in\mathcal{J}_m} p_{im} Q_m(p_{im}) - O(n_m \log n_m). \tag{32}$$

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

$$\begin{aligned}
&\mathbb{E}[\sum_{i\in\mathcal{J}'_m} p_{im} d_{im} f_i | \sum_{i\in\mathcal{J}'_m} d_{im} f_i \\
&\geq \overline{\mu}_m(C_m - d_{max}^m)] \\
&\geq \sum_{i\in\mathcal{J}'_m} \mathcal{A}_m(p_*^{cm}) f_i/(C_m \mu_m^U - \theta_m n_m) \\
&\geq \mathcal{A}_m(p_*^{cm})(1 - O(\frac{2r_{max}(\overline{\mu}_m)}{\overline{\mu}_m + 2r_{max}(\overline{\mu}_m)} + \frac{d_{max}^m}{C_m})) - \theta_m n_m,
\end{aligned} \tag{33}$$

where the last inequality holds due to Claim 2 and the definition of μ_m^U. Combining (32) and (33), we can get the lower-bound of the expected profit obtained by our DPS for selling type-m PSs:

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

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 $p_*^m \leq \delta_m$, then we know $\mathcal{A}_m(p_*^m) - \mathcal{A}_m(p_*^{cm}) \leq \delta_m C_m \overline{\mu}_m$. Let p'_m denote the highest price in $\mathcal{P}_m$ that is no higher than the best fixed price p_*^m, which indicates $p'_m \geq p_*^m/(1+\delta_m)$. Hence, we have

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

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 3

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 $\Phi(\theta_m n_m) \geq \theta_m n_m - n_m \tau_{max}/T$. Further, we have $r_{max}(\overline{\mu}_m) \leq O(\log n_m/(\tau_{max}\Phi(\theta_m n_m)))$. Due to $\tau_{max} \leq (T^5 C_m^2 \log^2 n_m)^{1/3}/(2n_m)$, we obtain $\Phi(\theta_m n_m) \geq (T^5 C_m^2 \log^2 n_m)^{1/3}/2$. Moreover, we know $|\mathcal{P}|_m \leq (\log n_m)/\delta_m$. 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

$$\sum_{m\in[M]} [\mathcal{A}_m(p_*^m) - \mathbb{E}[\mathcal{A}_m(\mathcal{L})]]$$

$$\leq \sum_{m\in[M]} O(\theta_m n_m + (TC_m)^{-1/3}\log^{2/3} n_m\overline{\mu}_m + \sqrt{n_m \log n_m} + (TC_m \log n_m)^{2/3}\mu_m^U/\overline{\mu}_m + C_m r_{max}(\overline{\mu}_m) + \overline{\mu}_m d_{max}^m)$$

$$\leq \sum_{m\in[M]} O(\sqrt{n_m \log n_m} + \frac{\mu_m^U (TC_m \log n_m)^{\frac{2}{3}}}{\overline{\mu}_m} + C_m r_{max}(\overline{\mu}_m)) \tag{34}$$

$$\leq O(\sum_{m\in[M]} \sqrt{n_m \log n_m} + (TC_m \log n_m)^{2/3}). \tag{35}$$

Inequality (34) holds since we assume $d_{max}^m \leq T^{-1/3}(C_m \log n_m)^{2/3}$. Due to $r_{max}(\overline{\mu}_m) \leq O(\frac{(1+\overline{\mu}_m)\log n_m}{1+\Phi(\theta_m n_m)})$, we know $C_m r_{max}(\overline{\mu}_m) \leq (TC_m)^{2/3}(\log n_m)^{1/3}$. Moreover, $\mu_m^U/\overline{\mu}_m$ asymptotically approaches $O(1)$. Putting them together, the last inequality (35) 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, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Operating Syst. Design Implement. (OSD)*, Aug. 2014, pp. 583–598.

[2] Q. Ho et al., "More effective distributed ML via a stale synchronous parallel parameter server," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 1–9.

[3] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 505–513.

[4] L. Mai, C. Hong, and P. Costa, "Optimizing network performance in distributed machine learning," in *Proc. USENIX HotCloud*, 2015, pp. 1–7.

[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. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2592–2600.

[6] (2019). *Amazon EC2 Pricing*. [Online]. Available: https://aws.amazon.com/ec2/pricing/

[7] (2019). *Google Cloud Pricing*. [Online]. Available: https://cloud.google.com/pricing/

[8] (2019). *Linux Virtual Mach. Pricing*. [Online]. Available: 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. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 2456–2464.

[10] W. Wang, B. Liang, and B. Li, "Revenue maximization with dynamic auctions in IaaS cloud markets," in *Proc. IEEE/ACM 21st Int. Symp. Quality Service (IWQoS)*, Jun. 2013, pp. 1–6.

[11] W. Shi, C. Wu, and Z. Li, "RSMOA: A revenue and social welfare maximizing online auction for dynamic cloud resource provisioning," in *Proc. IEEE 22nd Int. Symp. Quality Service (IWQoS)*, May 2014, pp. 41–50.

[12] L. Mashayekhy, M. M. Nejad, D. Grosu, and A. V. Vasilakos, "An online mechanism for resource allocation and pricing in clouds," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1172–1184, Apr. 2015.

[13] X. Wang et al., "Maximizing the profit of cloud broker with priority aware pricing," in *Proc. IEEE 23rd Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2017, pp. 511–518.

[14] B. Baek, J. Lee, Y. Peng, and S. Park, "Three dynamic pricing schemes for resource allocation of edge computing for IoT environment," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4292–4303, May 2020.

[15] T. O. Omotehinwa and J. S. Sadiku, "A dynamic strategy-proof algorithm for allocation and pricing of cloud services," *Int. J. Cloud Comput.*, vol. 8, no. 2, pp. 166–182, 2019.

[16] J. Correa, P. Foncea, R. Hoeksma, T. Oosterwijk, and T. Vredeveld, "Posted price mechanisms for a random stream of customers," in *Proc. ACM Conf. Econ. Comput.*, Jun. 2017, pp. 169–186.

[17] Z. Wang, J. Wu, Y. Wu, S. Deng, and H. Huang, "QoS aware dynamic pricing and scheduling in wireless cloud computing," in *Proc. IEEE 56th Annu. Conf. Decis. Control (CDC)*, Dec. 2017, pp. 3702–3707.

[18] M. Nambiar, D. Simchi-Levi, and H. Wang, "Dynamic learning and pricing with model misspecification," *Manage. Sci.*, vol. 65, no. 11, pp. 4980–5000, Nov. 2019.

[19] P. Liu, G. Bravo, and J. Guitart, "Energy-aware dynamic pricing model for cloud environments," in *Proc. GECON*, 2019, pp. 71–80.

[20] R. Kleinberg, A. Slivkins, and E. Upfal, "Multi-armed bandits in metric spaces," in *Proc. 14th Annu. ACM Symp. Theory Comput.*, May 2008, pp. 681–690.

[21] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, nos. 2–3, pp. 235–256, 2002.

[22] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Found. Trends Mach. Learn.*, vol. 5, no. 1, pp. 1–122, 2012.

[23] O. Besbes, Y. Gur, and A. Zeevi, "Optimal exploration-exploitation in a multi-armed bandit problem with non-stationary rewards," *Stochastic Syst.*, vol. 9, no. 4, pp. 319–337, 2019.

[24] S. Nobari, "DBA: Dynamic multi-armed bandit algorithm," in *Proc. AAAI*, 2019, pp. 9869–9870.

[25] M. Mahdavi, T. Yang, and R. Jin, "Efficient constrained regret minimization," 2012, *arXiv:1205.2265*.

[26] J. Chen, K. Li, Q. Deng, K. Li, and P. S. Yu, "Distributed deep learning model for intelligent video surveillance systems with edge computing," *IEEE Trans. Ind. Informat.*, early access, Apr. 4, 2019, doi: 10.1109/TII.2019.2909473.

[27] J. Chen et al., "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 919–933, Apr. 2017.

[28] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. USENIX NSDI*, 2011, pp. 1–14.

[29] B.-G. Chun et al., "Dolphin: Runtime optimization for distributed machine learning," in *Proc. ICML ML Syst. Workshop*, 2016, pp. 1–6.

[30] C. Chen, W. Wang, and B. Li, "Performance-aware fair scheduling: Exploiting demand elasticity of data analytics jobs," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 504–512.

[31] C. Xu, K. Wang, P. Li, R. Xia, S. Guo, and M. Guo, "Renewable energy-aware big data analytics in geo-distributed data centers with reinforcement learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, pp. 205–215, Jan. 2020.

[32] A. Mirhoseini et al., "Device placement optimization with reinforcement learning," in *Proc. ACM ICML*, 2017, pp. 2430–2439.

[33] Y. Azar, I. Kalp-Shaltiel, B. Lucier, I. Menache, J. Naor, and J. Yaniv, "Truthful online scheduling with commitments," in *Proc. 16th ACM Conf. Econ. Comput.*, Jun. 2015, pp. 715–732.

[34] M. Cheong, H. Lee, I. Yeom, and H. Woo, "SCARL: Attentive reinforcement learning-based scheduling in a multi-resource heterogeneous cluster," *IEEE Access*, vol. 7, pp. 153432–153444, 2019.

[35] J. Chen, K. Li, K. Bilal, X. Zhou, K. Li, and P. S. Yu, "A bi-layered parallel training architecture for large-scale convolutional neural networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 965–976, May 2018.

[36] X. Li, R. Zhou, L. Jiao, C. Wu, Y. Deng, and Z. Li, "Online placement and scaling of geo-distributed machine learning jobs via volume-discounting brokerage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 948–966, Apr. 2020.

[37] C.-T. Chen, L.-J. Hung, S.-Y. Hsieh, R. Buyya, and A. Y. Zomaya, "Heterogeneous job allocation scheduler for Hadoop MapReduce using dynamic grouping integrated neighboring search," *IEEE Trans. Cloud Comput.*, vol. 8, no. 1, pp. 193–206, Jan. 2020.

[38] G. Lee, W. Saad, and M. Bennis, "An online optimization framework for distributed fog network formation with minimal latency," *IEEE Trans. Wireless Commun.*, vol. 18, no. 4, pp. 2244–2258, Apr. 2019.

[39] M. Chen, H. V. Poor, W. Saad, and S. Cui, "Convergence time optimization for federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 4, pp. 2457–2471, Apr. 2021.

[40] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *J. Heuristics*, vol. 4, no. 1, pp. 63–86, Jun. 1998.

[41] H. Aissi, C. Bazgan, and D. Vanderpooten, "Min–max and min–max regret versions of combinatorial optimization problems: A survey," *Eur. J. Oper. Res.*, vol. 197, no. 2, pp. 427–438, 2009.

[42] P. Minet, E. Renault, I. Khoufi, and S. Boumerdassi, "Analyzing traces from a Google data center," in *Proc. 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2018, pp. 1167–1172.

[43] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence properties of the Nelder-Mead simplex method in low dimensions," *SIAM J. Optim.*, vol. 9, no. 1, pp. 112–147, 1998.

[44] Y. Nakatani, "Structured allocation-based consistent hashing with improved balancing for cloud infrastructure," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 9, pp. 2248–2261, Sep. 2021.

Ruiting Zhou received the Ph.D. degree from the Department of Computer Science, University of Calgary, Canada, in 2018. She is currently an Associate Professor with the School of Computer Science Engineering, Southeast University. She has published research papers in top-tier computer science conferences and journals, including IEEE INFOCOM, ACM MOBIHOC, IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, and IEEE TRANSACTIONS ON MOBILE COMPUTING. Her research interests include cloud computing, machine learning, and mobile network optimization. She serves as the TPC Chair for INFOCOM Workshop-ICCN (2019–2023). She also serves as a Reviewer for international conferences and journals, such as IEEE ICDCS, IEEE/ACM IWQoS, IEEE SECON, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON MOBILE COMPUTING, and IEEE TRANSACTIONS ON CLOUD COMPUTING.

Xueying Zhang received the B.E. degree from the School of Computer Science, Wuhan University, China, in 2018, where she is currently pursuing the master's degree with the School of Cyber Science and Engineering. Her research interests include the areas of network optimization, online learning, and online scheduling.

John C. S. Lui (Fellow, IEEE) received the Ph.D. degree in computer science from UCLA. He is currently the Choh-Ming Li Chair Professor with the Department of Computer Science and Engineering (CSE), The Chinese University of Hong Kong (CUHK). After his graduation, he joined the IBM Laboratory and participated in research and development projects on file systems and parallel I/O architectures. He joined the CSE Department, CUHK. His research interests include films and general reading. His current research interests include online learning algorithms and applications (e.g., multi-armed bandits and reinforcement learning), machine learning on network sciences and networking systems, large scale data analytics, network/system security, network economics, large scale storage systems, and performance evaluation theory. He received various departmental teaching awards and the CUHK Vice-Chancellor's Exemplary Teaching Award. He also received the CUHK Faculty of Engineering Research Excellence Award (2011–2012). He is an Elected Member of the IFIP WG 7.3, a fellow of ACM, a Senior Research Fellow of the Croucher Foundation, and a fellow of the Hong Kong Academy of Engineering Sciences (HKAES), and was the Past Chair of the ACM SIGMETRICS (2011–2015).

Zongpeng Li received the B.Sc. degree in computer science and technology from Tsinghua University in 1999 and the Ph.D. degree from the University of Toronto in 2005. He is currently a Professor at Tsinghua University. His research interests include computer networks, network coding, network algorithms, and cyber security. He received the Outstanding Young Computer Science Researcher Award from the Canadian Association of Computer Science in 2015. He received best paper awards from five international conferences.