This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

DPS: Dynamic Pricing and Scheduling for Distributed Machine Learning Jobs in Edge-Cloud Networks

Ruiting Zhou, *Member, IEEE,* Ne Wang, Yifeng Huang, Jinlong Pang, and Hao Chen

Abstract—5G and Internet of Things stimulate smart applications of edge computing, such as autonomous driving and smart city. As edge computing power increases, more and more machine learning (ML) jobs will be trained in the edge-cloud network, adopting the parameter server (PS) architecture. Due to the distinct features of the edge (low-latency and the scarcity of resources), the cloud (high delay and rich computing capacity) and ML jobs (frequent communication between workers and PSs and unfixed runtime), existing cloud job pricing and scheduling algorithms are not applicable. Therefore, how to price, deploy and schedule ML jobs in the edge-cloud network becomes a challenging problem. To solve it, we propose an auction-based online framework DPS. DPS consists of three major parts: job admission control, price function design and scheduling orchestrator. DPS dynamically prices workers and PSs based on historical job information and real-time system status, and decides whether to accept the job according to the deployment cost. DPS then deploys and schedules accepted ML jobs to pursue the maximum social welfare. Through theoretical analysis, we prove that DPS can achieve a good competition ratio and truthfulness in polynomial time. Large-scale simulations and testbed experiments show that DPS can improve social welfare by at least 95%, compared with benchmark algorithms in today's cloud system.

Index Terms—Distributed Machine Learning, Online Pricing, Online Scheduling, Parameter Server Architecture, Edge-Cloud Networks.

1 Introduction

With the substantial breakthroughs in deep learning (DL) algorithms, computing power and big data, the development of machine learning (ML) steps into an unprecedented booming era. Clearly, the success of ML applications benefits from massive training data. Due to the advances in the Internet of Things (IoT), data sources have recently undergone a radical shift from large-scale cloud datacenters to end devices. Traditionally, these data are transmitted to the cloud for training ML model via a wide area network (WAN) [1], whose monetary cost and transmission delay can be relatively high. To overcome this challenge, edge computing has been proposed to process data on edge nodes close to data sources [2], [3]. However, compared with cloud servers, edge nodes have fewer computing and storage resources [4]. Therefore, to facilitate distributed training of ML models, many studies turn to edge-cloud networks that combine the advantages of cloud and edge [5], [6].

Distributed training of ML models usually adopts a parameter server (PS) [7] architecture, even in edge networks [8], [9]. The PS architecture employs components called *workers* and *PSs* to co-train models, where the workers perform local updates using data sets and the PSs aggregates the gradients from the workers. Obviously, if all the workers and PSs of a ML job cannot be placed on the same server, the communication between them will consume a large amount of network bandwidth. This phenomenon is very common in edge networks, since the resources of edge nodes are relatively scarce. In practice, the PS architecture usually works with data parallelism [4], [10], where the large data sets of ML jobs are partitioned among multiple workers. Though such cooperation, the workflow of training ML models is shown as Fig. 1. In each iteration of model training, each worker generates gradients by training its data sets with local parameters and then pushes them to PSs. Next, the PSs update global model parameters using received gradients, and then the workers pull latest parameters from PSs. Regarding the parameter exchange, there are synchronous and asynchronous modes, where the former performs better in terms of model convergence and model accuracy [11].

The training of ML models is costly [12]. From the perspective of economic benefit, edge-cloud computing operators hope to maximize their revenues with their adopted pricing scheme, while customers aim to obtain the best quality of service (QoS) at a reasonable price. Therefore, to satisfy both parties, effective pricing methods and scheduling policies are required. However, pricing and scheduling ML jobs faces several challenges. *First*, the designed pricing approaches should be flexible enough to regulate the supply and demand of resources. When supply exceeds demand, resource prices should be low enough to stimulate

- *R. Zhou, Y. Huang and J. Pang are with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan, China. E-mail: {ruitingzhou, yifenghuang, jinlongpang}@whu.edu.cn.*
- *N. Wang is with the School of Computer Science, Wuhan University, Wuhan, China. E-mail: ne.wang@whu.edu.cn.*
- *H. Chen is with Huawei, Nanjing, China. E-mail: philips.chenhao@huawei.com*

Manuscript received Nov. 17, 2021; revised Apr. 2, 2022 and Jul. 1, 2022; accepted Jul. 22, 2022. This work is supported in part by the NSFC Grants (62072344 and U20A20177), Hubei Science Foundation (2020CFB195), Compact Exponential Algorithm Project of Huawei (YBN2020035131), Huawei Project (FA2019071041). (Corresponding author: Ne Wang.)

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

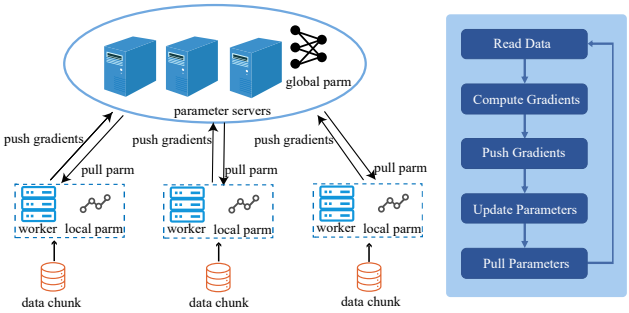

Fig. 1: The parameter server architecture.

customers' desire to consume. On the contrary, goods prices should be raised to discourage customers from buying, so as to reduce the pressure on operators. *Second*, the infrastructure of edge and cloud are quite heterogeneous. ML jobs are more inclined to train at the edge under the same computing power, since the edge is close to data sources and has low transmission delay. However, edge servers have much fewer resources than cloud servers. Consequently, the scheduling decision needs to consider the impact of both latency and computing resources [13].

Unfortunately, the existing pricing and scheduling policies for cloud services are not suitable for charging ML jobs in edge-cloud networks. In edge/cloud computing, operators, such as Amazon EC2 [14] and Windows Azure [15], widely employ pay-as-you-go pricing models, where operators first post real-time resource price and then customers buy them on demand. Such transactions require customers to have an accurate estimate of the runtime and resource usage of jobs. However, for the owners of ML jobs, this estimation is difficult to implement due to the distinct features of ML jobs. *First*, ML jobs are placement-sensitive, *i.e.*, whether all the workers and PSs are placed on the same servers (referring to the cloud servers) or not has a great impact on the running time of the ML jobs. *Second*, different ML models show heterogeneous performance for the same resource type. For instance, ResNet-50 model achieves a speedup of around 10x on V100 compared to a K80 GPU, whereas A3C model only sees a 2x speedup [16]. With this heterogeneity, ML users may have difficulties in choosing resource types. *Third*, the running speed of ML jobs is not linearly proportional to the amount of resources allocated to them. For example, the speed of training a ResNet-50 model reaches 0.089 steps/s with 12 workers and 0.092 steps/s with 16 workers [17]. As a result, it is not economic for ML users to estimate the amount of resources needed using simple linear pricing approaches [18]. With these factors, ML users are very likely to make wrong estimates. The underestimation of demands lowers the training progress, whereas overestimation leads to inefficiency in resource utilization.

To this end, this paper addresses the dynamic pricing and scheduling problem for distributed ML Jobs in edge-cloud networks. To the best of our knowledge, we are the *first* formal study that are focusing on the pricing and scheduling design for ML Jobs under the PS architecture in the edge-cloud system. We propose an action-based online algorithm, DPS, which determines whether or not to accept user's bids for training ML models, and if so, how to deploy and charge accepted ML jobs in the edge-cloud network. Auction mechanism can capture the dynamic fluctuation between supply and demand in markets, and allocate resources to users who value them most [19]. The work considers both ML service operator and ML users, and targets social welfare maximization for maximizing the efficiency of the edge-cloud system in the long run. We summarize our main contributions as follows.

First, we formulate the social welfare maximization problem into a mix integer nonlinear program (MINLP). The problem involves both conventional constraints (resource capacity constraints) and non-conventional constraints such as job start time, job completion time and data transmission delay. In addition, there exists the inherent structure feature called "triangular covering" [20], *i.e.*, variables are interrelated in our problem. The challenge further escalates when jobs arrive online and the scheduler must make decisions immediately without any future information. The problem is NP-hard even in the offline setting. To tackle these challenges, we first reformulate the problem into an integer linear program (ILP) with packing-type constraints by introducing an exponential number of variables, then we adopt the posted price mechanism and primal-dual framework to solve the reformulated problem.

Second, we propose an online pricing and scheduling algorithm DPS. DPS is decomposed into three key components – admission control, price function design and scheduling policy. For admission control, DPS considers the dual of reformulated problem by relaxing integer variables and introducing dual variables. By interpreting the dual variables as unit worker (PS) prices, DPS uses the subroutine $A_{bestsche}$ to calculate the estimated cost of each job. The algorithm accepts a job only if its estimated cost is lower than its bidding price. For price function design, we propose exponential functions to dynamically price various types of workers and PSs. The parameter of each price function is calculated based on past jobs and is continuously updated based on the real-time system status. For scheduling policy, $A_{bestsche}$ finds the optimal schedule and placement for each job, given the price of each type of unit worker (PS). Since the schedule is jointly determined by many factors including the type, number, placement and start time of workers and PSs, we adopt control variates to compute the optimal solution with highest payoff. In particular, considering the significant impact of different placement settings on communication time, we propose function $COST\_C$ for centralized placement and $COST\_D$ for distributed placement. Both algorithms greedily place workers and PSs.

Third, we conduct rigorous theoretical analysis to prove that DPS generates a feasible solution for social welfare maximization problem with a polynomial time complexity. We also analyze the competitive ratio of DPS. Moreover, we carry out both testbed experiment and large-scaled simulations to evaluate the practical performance of our algorithm. We first implement DPS as a scheduler in a well-designed testbed. The implemented testbed is based on MXNet [21] Framework with Kubernetes [22], which consists of nine physical machines. Especially, eight of them act as edge servers, and another one serves as both the remote cloud

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

and the central scheduler. The results of testbed illustrate that our algorithm DPS outperforms two benchmark algorithms *DRF* [23], [24], [25] and *FIFO* [26], [27] in terms of social welfare. Then simulations verify that DPS also achieves a much better performance than two baselines in the large scale of inputs. Particularly, DPS achieves a near-optimal social welfare with a small competitive ratio (less than 1.6) and increases social welfare by 95% and 259% compared with *DRF* and *FIFO*.

In the rest of the paper, Sec. 2 reviews related work. We introduce the system model in Sec. 3 and design the dynamic pricing and scheduling algorithm in Sec. 4. Sec. 5 introduces the testbed implementation, and Sec. 6 evaluates the performance of DPS through both testbed experiments and simulations. Sec. 7 concludes the paper.

2 Related Work

Dynamic Pricing for Cloud Resources. There are plenty of studies on dynamic pricing of cloud resources. Some focus on pricing virtual machines (VMs) [28] [19]. Mashayekhy *et al.* [28] design an auction-based mechanism for VMs pricing in clouds. Lu *et al.* [19] present a truthful double auction mechanism to price VMs in cloud markets. It first uses a matching process to determine the number of VMs required by users, then prices VMs in the double auction. Chen *et al.* [29] study how to allocate and price inter-VM bandwidth fairly for tenants in an IaaS cloud. Some researchers [30] [31] focus on pricing cloud services. Omotehinwa et al. [30] present a dynamic strategy-proof algorithm to price cloud services, which relies on the market history to predict a benchmark price for ensuring truthful valuation from participants. QDPSA [31] focuses on pricing and scheduling problem in wireless cloud computing. It dynamically prices cloud services according to the demand of users and system state. Zhang *et al.* [32] develop an occupation-oblivious online pricing method for cloud jobs, which does not require to pre-describe the duration of jobs. Zhang *et al.* [12] propose a dynamic pricing and placement mechanism for distributed ML jobs, which aims to maximize the profit of cloud provider and minimize the runtime of jobs. The above work does not capture the distinct features of ML jobs and cannot be applied to our model.

Job Scheduling and Resource Allocation in Edge-Cloud Systems. Tan *et al.* [13] propose the first online task dispatching and scheduling algorithm in edge-clouds. It considers both upload and download delays of jobs and focuses on minimizing the total weighted response time of all jobs. Xu *et al.* [33] study service caching and task offloading strategy in mobile edge computing system. Meng *et al.* [34] propose an online algorithm that jointly considers bandwidth and computing resources to dispatch and schedule deadline-aware tasks in edge computing. Zhang *et al.* [35] investigate online dispatching and scheduling jobs with different utility functions. The above work does not consider the pricing problem in edge-cloud systems.

Distributed Machine Learning Systems. Ghodsi *et al.* [25] design a fair allocation strategy for multiple resource types. Dorm [36] first partition a cluster, then run one application per partition, and finally dynamically resize the partitions with the goal of achieving resource efficiency and fairness. The above work does not consider resource allocation among multiple jobs. Chen *et al.* [37] propose a performance-aware fair scheduler to identify the demand elasticity of resources, which is used to improve average job performance. The above methods are designed for offline instead of online scenarios. Amiri *et al.* [38] propose two scheduling schemes that allocate tasks to workers and specify their scheduling order to minimize average completion time. Xiao *et al.* [39] design a new distributed graph engine for ML jobs represented by graph models. It preserves the benefits of graph computation while supporting efficient distributed ML training. Li *et al.* [40] study a ML broker service that aggregates geo-distributed ML jobs together to achieve volume discounts. It focuses on cost minimization. Bao *et al.* [5] propose an online algorithm to maximize overall utility of all jobs. It adjusts the number of workers and PSs for each uncompleted job over time. Zhang *et al.* [6] develop an online algorithm for ML jobs with elastic demand and different resource configurations, which aims to minimize the weighted average completion time of jobs. Different from the above literature, our work is the first work to joint optimize the problem of dynamic pricing and online scheduling ML jobs in the edge-cloud network, whose goal is to maximize the social welfare.

3 Problem Modeling

In this section, we first introduce the system setup in sec. 3.1, then model the optimization problem and outline some basic constraints in sec. 3.2.

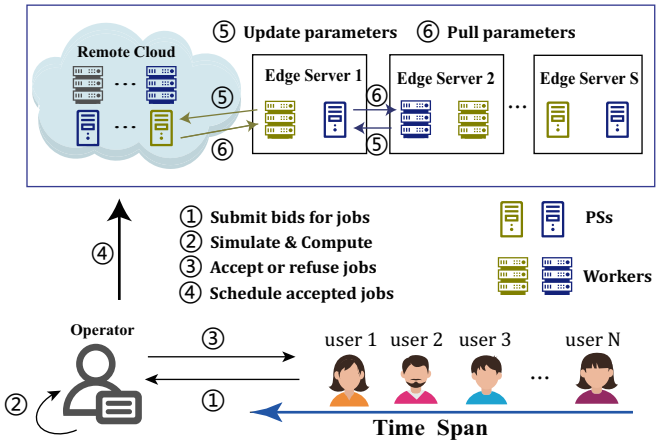

Fig. 2: An illustration of the pricing and scheduling process in the edge-cloud system.

3.1 Auction Setup and Agent Interactions

Auction Overview. As shown in Fig. 2, our model involves two types of entities interacting in an auction: the ML operator (auctioneer) and the user (bidder). The user-operator interaction consists of three steps. *First*, a user submits its bid once arrived. *Second*, the ML operator simulates an allocation and scheduling policy for the user and computes corresponding payment. If this solution does not violate the individual rationality (referred to non-negative user payoff), the two agents sign a contract and their interaction steps into the third stage, otherwise terminates. *Third*, the ML

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

operator truly allocates resources to the job according to the simulated schedule in step 2 and charges the user. Let $[X]$ represent an integer set $\{1, 2, ..., X\}$.

ML Operator. The ML operator serving as the owner and manager of resources, hosts plenty of heterogeneous edge servers and one cloud, denoted as set $[S]$. These servers make up an edge-cloud network, and each of them holds a pool of resources (*i.e.*, W types of workers and P types of PSs), which serve as goods and need leasing to users by an auction. There are C_{sw} type-w workers and C_{sp} type-p PSs pre-installed in each server $s \in [S]$. Each type-w worker (type-p PS) reserves b_w (b_p) bandwidth.

User. Users act as bidders. They arrive online and submit bids for training ML jobs. We assume that a user submits only one job at a time. Therefore, in the remainder of this paper, we do not differentiate between a user and a job unless otherwise stated. Let $[J]$ be the set of ML jobs. We use a tuple B_j to express job j's bid submitted at time r_j:

$$B_j = \{r_j, f_j(\cdot), \{D_j, M_j, E_j\}, \{\Delta_{js}^{\uparrow}\}_{\forall s}, \{k_j^{wp}\}_{\forall w, \forall p}\},$$

where $f_j(\cdot)$ is a job-dependent non-negative utility function, non-increasing with the job execution duration. The job utility function is customized for each user according to the characteristics of job workload, and $f_j(\cdot)$ is "claimed" by user j. Each job j holds large input data sets, which are usually organized in the form of D_j equal-sized data chunks. Moreover, each data chunk $d \in [D_j]$ is further divided into M_j equal-sized mini-batches. Due to the non-deterministic termination conditions of DL jobs [41], it is natural to ensure the termination of training process by specifying a maximum epoch number, denoted as E_j. Once job j wins the bid, each data chunk d is dispatched to one allocated worker on server s with a transmission delay $\Delta_{js}^{\uparrow}$. This delay is the same for all workers on the identical server. k_j^{wp} is the training time of one mini-batch in job j employing type-w worker(s) and type-p PS(s).

Next we discuss the computation of k_j^{wp} under PS architecture and synchronous training mode. The time of processing a mini-batch when job j employs type-w workers is e_j^w. The time of type-p PS to update job j's parameters is denoted as g_j^p, in analogy to the notion of e_j^w. Due to the same size of gradients and parameters, denoted as m_j, the time to push gradients and pull parameters can be represented by $\frac{m_j}{b_w}$. This time is negligible when all allocated workers and PSs are placed on the same server. We introduce and set an indicator variable $\psi_j = 1$ to reflect such placement, otherwise set $\psi_j = 0$. Thus, we have:

$$k_j^{wp} = \begin{cases} e_j^w + g_j^p + \dfrac{2m_j}{b_w}, & \text{if} \quad \psi_j = 0 \\ e_j^w + g_j^p, & \text{if} \quad \psi_j = 1 \end{cases} \tag{1}$$

Scheduling Decision. During the aforementioned interaction, the following decisions are made by the ML operator for job j: (i) $x_j \in \{0, 1\}$, which implies whether job j's bid is accepted ($x_j = 1$) or not ($x_j = 0$). (ii) $o_{jw}, o_{jp} \in \{0, 1\}$, which specify the type of workers and PSs that job j employs for training, respectively. (iii) $y_{jsw}(t), z_{jsp}(t) \in \{0, 1, 2, ...\}$, which are the number of workers and PSs allocated to job j by physical server s at time t, respectively. (iv) $p_j \geq 0$, which is the payment to ML operator if job j's bid is accepted.

Auction Preliminary. We next introduce some definitions in auction design. Let $v_j(\cdot)$ be the true valuation of job j's bid. And job j's payoff is $u_j(f_j(\cdot)) = v_j(\cdot) - p_j$ if $x_j = 1$, and is 0 if $x_j = 0$. The ML operator's revenue is the total payment of all winner jobs, *i.e.*, $\sum_j p_j x_j$. In practice, users are selfish and aim at maximizing respective payoff. Thus they may try to gain higher payoff by lying about their true valuations. In this work, we neither maximize the user's payoff nor the ML operator's revenue, and target *social welfare maximization*. Therefore, it is significant to elicit truthful bids. Important notations are listed in Table I.

Definition 1. (Truthful Auction)*: An auction is truthful if every user gains maximal payoff by reporting its true valuation,* i.e., $u_j(v_j(\cdot)) \geq u_j(f_j(\cdot)), \forall v_j(\cdot) \neq f_j(\cdot)$ *[42].*

Definition 2. (Social Welfare)*: The social welfare in the edge-cloud market is the aggregated user payoff* $\sum_j x_j(v_j(\cdot) - p_j)$ *plus the ML operator's revenue* $\sum_j p_j x_j$*,* i.e., $\sum_j x_j v_j(\cdot)$.

Definition 3. (Individual Rationality)*: An auction is individually rational if every user's payoff is always non-negative,* i.e., $u_j(f_j(\cdot)) \geq 0$ *[42].*

TABLE 1. Notations

Inputs	Descriptions
$[X]$	integer set $\{1, 2, ..., X\}$
J, S, T	# of jobs, physical servers and time slots
j, s, t	indexes of jobs, physical servers and time slots
W, P	# of worker and PS types, respectively
$r_j, f_j(\cdot)$	arrival time and utility function of job j
C_{sw}, C_{sp}	# of type-w workers and type-p PSs on server s
E_j, D_j, M_j	# of training epochs, data chunks and mini-batches in one data chunk of job j
$\Delta_{js}^{\uparrow}$	delay to transmit one data chunk of job j to server s
b_w, b_p	bandwidth of a type-w worker and type-p PS
Decisions	**Descriptions**
x_j	accept job j or not
o_{jw}, o_{jp}	whether job j employs type-w workers and type-p PSs
$y_{jsw}(t)$	# of type-w workers allocated to job j by s at t
$z_{jsp}(t)$	# of type-p PSs allocated to j by server s at time t
p_j	the payment that job j should pay if admitted
$a_j, \hat{t}_j$	start and completion time of j
ψ_j	whether a job's workers and PSs are placed together
k_j^{wp}	time to train one mini-batch in job j adopting type-w workers and type-p PSs

3.2 Problem Formulation

Constraints and Assumptions. DPS must obey some realistic assumptions and important constraints. We highlight them as below:

(1) *Admission, Homogeneous Worker (PS) Configuration:* Only after job j is admitted can the worker and PS type be allocated. Moreover, each job can only employ one type of worker (PS), as it is common to use homogeneous workers for synchronous training ML jobs in production [43] [44]. This constraint can be formulated as:

$$\sum_{w \in [W]} o_{jw} = \sum_{p \in [P]} o_{jp} = x_j, \forall j \tag{2}$$

(2) *Worker Allocation:* To keep job's worker type unchanged over the training course, we introduce constraint (3). We can easily observe that only if type w is picked (*i.e.*, $o_{jw} = 1$), can the scheduler allocate type-w workers to users (*i.e.*, $y_{jsw}(t) > 0$). Besides, the number of selected-type workers is limited to be at most the number of data chunks, D_j, to ensure that one data chunk is trained by at most one

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

worker [6] [5]. Furthermore, to avoid preemption overhead, each job occupies the assigned workers until its training is completed, shown as constraint (4).

$$\sum_{s\in[S]} y_{jsw}(t) \le D_j o_{jw}, \forall j, \forall w, \forall t \tag{3}$$

$$y_{jsw}(t) = y_{jsw}(t+1), \forall j, \forall s, \forall w, \forall t \in [a_j, \hat{t}_j - 1] \tag{4}$$

(3) *PS Allocation:* For the same reason, each job can use only one type of PS. But the number of allocated PSs can be as many as the total capacity of all servers. The above requirements can be modeled as constraint (5). In addition, if a job is running, the scheduler must assign at least one PS to the job to ensure that the training process works normally, shown as constraint (6). Likewise, in constraint (7), the PSs of a job will not be preempted over the job's training course.

$$\sum_{s\in[S]} z_{jsp}(t) \le o_{jp} \sum_{s\in[S]} C_{sp}, \forall j, \forall p, \forall t \tag{5}$$

$$\sum_{s\in[S]} \sum_{p\in[P]} z_{jsp}(t) \ge 1, \forall j, \forall t : \sum_s \sum_w y_{jsw}(t) > 0 \tag{6}$$

$$z_{jsp}(t) = z_{jsp}(t+1), \forall j, \forall s, \forall p, \forall t \in [a_j, \hat{t}_j - 1] \tag{7}$$

(4) *Sufficient Training Time Guarantee:* Once a job j admitted ($x_j = 1$), the job should procure sufficient workers and PSs to accomplish its training. But the training time of $E_j D_j M_j$ mini-batches varies with the placement of workers and PSs. The formal description is as below:

$$\sum_{t\in[T]} \sum_{s\in[S]} y_{jsw}(t) \ge E_j D_j M_j k_j^{wp}, \forall j \tag{8}$$

$$\psi_j = 1, \forall j : s = s', \forall y_{jsw}(t) > 0, \forall z_{js'p}(t) > 0 \tag{9}$$

(5) *Resource Capacity Constraint:* The resource capacity limit of physical servers is formulated by constraints (10) and (11). Constraint (12) guarantees that the bandwidth reservation for PSs of job j in server s should cover the total bandwidth of job j's workers placed on other servers.

$$\sum_{j\in[J]} y_{jsw}(t) \le C_{sw}, \forall s, \forall w, \forall t \tag{10}$$

$$\sum_{j\in[J]} z_{jsp}(t) \le C_{sp}, \forall s, \forall p, \forall t \tag{11}$$

$$\sum_{s'\in[S]\setminus s} \sum_{w\in[W]} y_{js'w}(t) b_w \le \sum_{p\in[P]} z_{jsp}(t) b_p, \forall j, \forall s, \forall t \tag{12}$$

(6) *Resource Allocation with Delay:* Constraint (13) implies that users can employ workers and PSs only after a job reaches server s, and release them after completing model training.

$$y_{jsw}(t) = z_{jsp}(t) = 0, \forall j, \forall s, \forall t < r_j + \Delta_{js}^{\uparrow} \vee t \notin [a_j, \hat{t}_j] \tag{13}$$

(7) *Job Start and Completion Time Computation:* Let a_j and $\hat{t}_j$ be the start and completion time of job j, and they are computed as:

$$a_j = \arg\min_{t\in[T]} (\sum_s \sum_w y_{jsw}(t) > 0), \forall j \tag{14}$$

$$\hat{t}_j = a_j + \lceil E_j D_j M_j k_j^{wp} / \sum_s \sum_w y_{jsw}(a_j) \rceil, \forall j \tag{15}$$

Social Welfare Maximization Problem. Under the assumption of truthful bidding, *i.e.*, $\forall v_j(\cdot) = f_j(\cdot)$, the social welfare maximization problem can be formulated as follows.

$$\text{maximize} \quad \sum_{j\in[J]} x_j f_j(\hat{t}_j - r_j) \tag{16}$$

$$\text{subject to:} \quad (2) - (15),$$

$$y_{jsw}(t), z_{jsp}(t) \in \{0, 1, 2, ...\}, x_j, o_{jw}, o_{jp}, \psi_j \in \{0, 1\},$$

$$a_j, \hat{t}_j \in [T], \forall j, \forall s, \forall w, \forall p, \forall t. \tag{16a}$$

Problem Difficulties. The main challenges of our online optimization problem stem from three aspects: (i) The jobs arrive online, and the scheduling decisions are made on the fly. It is non-trivial to tackle such an online optimization problem. Problem (16) is a mix integer nonlinear program (MINLP), and even its offline setting is proven to be NP-hard [45]. (ii) There are many factors that make job duration unpredictable, such as the types, number and placement of workers and PSs (*i.e.*, $o_{jw}, y_{jsw}(t), o_{jp}, z_{jsp}(t), \psi_j$), elastic allocation ($y_{jsw}(t)$ and $z_{jsp}(t)$ are not user-specified). (iii) Similar to the problem in [20], ours also exists the inherent structure feature called "triangular covering" (*i.e.*, x_j covers o_{jw}, o_{jp} while o_{jw}, o_{jp} cover $y_{jsw}(t), z_{jsp}(t)$ respectively), which further poses challenges in the algorithm design.

4 DPS Design

To tackle the problem difficulties and achieve the goal of maximizing total social welfare, DPS is decomposed into **three key components** – admission control, price function design as well as scheduling policy orchestrator. *First*, DPS encapsulates all decision variables into a new one, then equivalently reformulates the original problem into a packing one. Moreover, DPS formulates the dual problem of the reformulated problem and devises auction admission control rule based on duality theory. *Second*, once a job is admitted, it needs to pay the ML operator for training model. To compute the payment, DPS prices workers and PSs for the ML operator based on law of supply and demand in market economy. *Third*, beside of resource prices, the number of workers and PSs consumed must be known when computing expenditure. DPS schedules admitted jobs with a goal of maximizing user payoffs. The main idea of DPS is illustrated in Fig. 3.

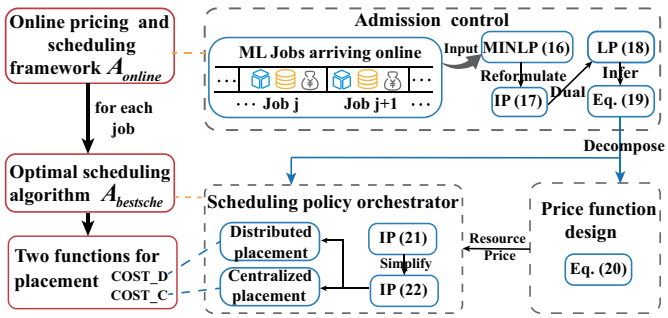

Fig. 3: The main structure of DPS.

i. In Sec. 4.1, when each job arrives online, its schedule is computed by solving MINLP (16). To circumvent the non-conventional constraints (14) and (15) of MINLP (16), we adopt the compact exponential technique [46] to reformulate MINLP (16) as an integer program (IP)

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

(17). However, such reformulation introduces potential exponential variables. To tackle this challenge, we formulate the dual problem (18) of IP (17). By well interpreting the dual variables and resorting to the complementary slackness, the admission control rule is developed and applied to our online framework DPS to deal with users' bids, *i.e.*, jobs with positive payoff are admitted, otherwise rejected. Next, we focus on the computation of jobs' payoff, which consists of two steps: resource pricing and resource allocation.

ii. In Sec. 4.2, we tailor-make price functions for workers and PSs in Eq. (20) according to the historical information of jobs and real-time state of our system, which complies with the market rules and can capture the relation between price and demand-supply.

iii. In Sec. 4.3, the resource allocation is computed by solving the income maximization problem (21) equivalent to LP (18) and IP (17), which is solved by algorithm $A_{bestsche}$. To deal with the heterogeneity of resources, we simplify problem (21) to problem (22) of minimizing the resource cost of each job. Finally, we solve problem (22) by designing functions COST_D and COST_C, which greedily allocate workers and PSs to the job under centralized and distributed placement, respectively.

We summarize the online algorithm framework DPS in Sec. 4.4 and analyze its performance in Sec. 4.5

4.1 Admission Control

Problem Reformulation. According to user-operator interaction, the schedule of one job plays an important role in the system admission control. A schedule is constituted by decision variables $o_{jw}, o_{jp}, y_{jsw}(t), z_{jsp}(t)$. For job j, let L_j denote the set of feasible schedules, *i.e.*, a feasible schedule satisfies constraints (2)-(9)(12)-(15)(16a). Moreover, we reshape the admission control variable x_j into $\chi_j^l, l \in L_j$. Here χ_j^l indicate whether job j is admitted and scheduled in accordance to schedule $l \in L_j$ ($\chi_j^l = 1$) or not ($\chi_j^l = 0$). Likewise, $\gamma_{jsw}^l(t)$ and $\zeta_{jsp}^l(t)$ denote the number of allocated workers and PSs at each time slot with schedule l, respectively. Similarly, $\hat{\tau}_j^l$ is the completion time with schedule l. Thus, MINLP (16) can be reformulated as the following IP (17).

$$\text{maximize} \quad \sum_{j\in[J]} \sum_{l\in L_j} \chi_j^l f_j(\hat{\tau}_j^l - r_j) \tag{17}$$

subject to:

$$\sum_{j\in[J]} \sum_{l\in L_j} \gamma_{jsw}^l(t)\chi_j^l \le C_{sw}, \forall s, \forall w, \forall t \tag{17a}$$

$$\sum_{j\in[J]} \sum_{l\in L_j} \zeta_{jsp}^l(t)\chi_j^l \le C_{sp}, \forall s, \forall p, \forall t \tag{17b}$$

$$\sum_{l\in L_j} \chi_j^l \le 1, \forall j \tag{17c}$$

$$\chi_j^l \in \{0,1\}, \forall j, \forall l \in L_j. \tag{17d}$$

Apparently, constraints (17a) and (17b) are equivalent to (10) and (11), respectively. Constraints (17c) and (17d) cover the remaining constraints in (16). In summary, we can solve problem (16) by solving the equivalent problem (17).

Dual Problem. The number of variables χ_j^l is potentially exponential. To tackle the challenges, we resort to its dual problem and formulate the dual (18) of problem (17) by relaxing integer variables $\chi_j^l \in \{0,1\}$ to $\chi_j^l \ge 0$ and introducing dual variables $\alpha_{sw}(t)$, $\beta_{sp}(t)$ and μ_j to constraints (17a), (17b) and (17c), respectively.

$$\begin{aligned}\text{minimize} \quad & \sum_{j\in[J]} \mu_j + \sum_{s\in[S]} \sum_{w\in[W]} \sum_{t\in[T]} \alpha_{sw}(t) C_{sw} \\ & + \sum_{s\in[S]} \sum_{p\in[P]} \sum_{t\in[T]} \beta_{sp}(t) C_{sp}\end{aligned} \tag{18}$$

subject to:

$$\begin{aligned}\mu_j \ge & f_j(\hat{\tau}_j^l - r_j) - \sum_{s\in[S]} \sum_{w\in[W]} \sum_{t\in[T]} \alpha_{sw}(t)\gamma_{jsw}^l(t) \\ & - \sum_{s\in[S]} \sum_{p\in[P]} \sum_{t\in[T]} \beta_{sp}(t)\zeta_{jsp}^l(t), \forall j, \forall l \in L_j\end{aligned} \tag{18a}$$

$$\mu_j, \alpha_{sw}(t), \beta_{sp}(t) \ge 0, \forall j, \forall s, \forall w, \forall p, \forall t. \tag{18b}$$

Admission Control and Behind Rationale. The dual variables $\alpha_{sw}(t)$ ($\beta_{sp}(t)$) can be interpreted as the unit price of type-w workers (type-p PSs) on server s at time t. As a result, $\sum_s \sum_w \sum_t \alpha_{sw}(t)\gamma_{jsw}^l(t) + \sum_s \sum_p \sum_t \beta_{sp}(t)\zeta_{jsp}^l(t))$ is the total charge that job j with schedule l should pay. The right hand side (RHS) of (18) is job j's utility minus the total resource cost. According to complementary slackness, if job j is admitted, *i.e.*, $\chi_j^l = 1$, the corresponding constraint (18a) is tight. In addition, we know $\mu_j \ge 0$ according to constraint (18b). Therefore, we have:

$$\mu_j = \max\{0, \max_{l\in L_j} \text{RHS of (18a)}\}. \tag{19}$$

μ_j can be interpreted as the payoff of user j when its job is admitted with optimal schedule l^*, where $l^* = \arg\max_{l\in L_j}$ RHS of (18a).

Consequently, the *admission control rule* is set as follows: if $\mu_j > 0$ with optimal schedule l^*, then the ML operator accepts job j, otherwise rejects this job's bid. The rational behind $\mu_j > 0$ is that job utility can afford to its resource usage, which is consistent with individual rationality in Definition 3. Furthermore, the ML operator wishes to accept jobs with larger utility and less resource consumption.

4.2 Price Function Design

To implement admission control, *i.e.*, the computation of μ_j, we need to solve the following sub-problems: (i) find the optimal schedule l^* for job j. This part will be discussed in Sec. 4.3. (ii) determine the marginal prices of workers and PSs, *i.e.*, dual variables $\alpha_{sw}(t)$ and $\beta_{sp}(t)$. We next focus on the price function design of workers and PSs.

In market economics, it is natural to increase resource prices when demand far exceeds supply in order to ease the pressure on resource suppliers. Conversely, when there is an oversupply, resource prices should be lowered to stimulate consumption. To this end, we develop the worker and PS price functions in line with such market rules. We define new variables $q_{sw}(t)$ ($h_{sp}(t)$) as the number of allocated type-w workers (type-p PSs) at time t on server s, and let marginal prices $\alpha_{sw}(t)$ ($\beta_{sp}(t)$) be a function of $q_{sw}(t)$ ($h_{sp}(t)$). Moreover, a good price function should own the following characteristics: (i) the prices have lower and upper bounds. They represent two extreme cases, *i.e.*, no resource usage and resource exhaustion. (ii) in general, $\alpha_{sw}(t)$ ($\beta_{sp}(t)$) increase with the growth of $q_{sw}(t)$ ($h_{sp}(t)$). (iii) the increment rate of the price of per type-w worker (type-p PS) is growing, in order to filter out jobs with low utilities and reserve resources for future jobs with higher utilities. In this work, we

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

choose exponential functions as the basis of price functions, and their formal expressions are as follows:

$$\alpha_{sw}(q_{sw}(t)) = (\eta_w)^{\frac{q_{sw}(t)}{C_{sw}}} - 1, \beta_{sp}(h_{sp}(t)) = (\eta_p)^{\frac{h_{sp}(t)}{C_{sp}}} - 1. \quad (20)$$

where $\eta_w = \max_j\{\frac{f_j(\hat{\tau}_j^l - r_j)}{\sum_s \sum_w \sum_t \gamma_{jsw}^l(t)}\} + 1$ and $\eta_p = \max_j\{\frac{f_j(\hat{\tau}_j^l - r_j)}{\sum_s \sum_p \sum_t \zeta_{jsp}^l(t)}\} + 1$, where $\frac{f_j(\hat{\tau}_j^l - r_j)}{\sum_s \sum_w \sum_t \gamma_{jsw}^l(t)}$ ($\frac{f_j(\hat{\tau}_j^l - r_j)}{\sum_s \sum_p \sum_t \zeta_{jsp}^l(t)}$) is the maximum possible utility per type-w worker (type-p PS) of job j.

We next discuss the behind rationale of their values. Our price functions are consistent with the aforementioned three characteristics. When there are no resources allocated, *i.e.*, $q_{sw}(t) = h_{sp}(t) = 0$, we have $\alpha_{sw}(t) = \beta_{sp}(t) = 0, \forall s, \forall w, \forall p, \forall t$ regardless of the values of η_w and η_p. In this case, any job can be admitted and scheduled. Such incentive mechanism can attract more users to participate in the auction, thereby increasing the probabilities of admitting jobs with high utility. Subsequently, we analyze the case where type-w workers or type-p PSs on server s are exhausted at time t, *i.e.*, $q_{sw}(t) = C_{sw}$ or $h_{sp}(t) = C_{sp}$. Consequently, $\alpha_{sw}(t) = \eta_w - 1$ or $\beta_{sp}(t) = \eta_p - 1$. Since there are no such resources to accommodate more jobs, $\eta_w - 1$ and $\eta_p - 1$ should be large enough to prevent jobs from being admitted, *i.e.*, $f_j(\hat{\tau}_j^l - r_j) - \sum_s \sum_w \sum_t (\eta_w - 1)\gamma_{jsw}^l(t) - \sum_s \sum_p \sum_t (\eta_p - 1)\zeta_{jsp}^l(t) < 0, \forall j$. Since we make no assumptions on the number of workers and PSs, there may exist the following case where type-w workers are exhausted while the number of available type-p PSs is infinite. As a result, $\sum_s \sum_p \sum_t (\eta_p - 1)\zeta_{jsp}^l(t) \to 0$ due to $\eta_p - 1 \to 0$ and limited $\zeta_{jsp}^l(t)$. To this end, to satisfy the inequality for rejecting jobs, we have $f_j(\hat{\tau}_j^l - r_j)/\sum_s \sum_w \sum_t \gamma_{jsw}^l(t) \leq \eta_w - 1, \forall j$. Thus, we can let $\eta_w - 1 = \max_j\{f_j(\hat{\tau}_j^l - r_j)/\sum_s \sum_w \sum_t \gamma_{jsw}^l(t)\}$. Similarly, we have $f_j(\hat{\tau}_j^l - r_j)/\sum_s \sum_p \sum_t \zeta_{jsp}^l(t) \leq \eta_p - 1, \forall j$ and $\eta_p - 1 = \max_j\{/f_j(\hat{\tau}_j^l - r_j)/\sum_s \sum_p \sum_t \zeta_{jsp}^l(t)\}$. Besides, such property can discourage users to submit bids and enable the ML operator to get rid of peak load as quickly as possible.

4.3 Scheduling Policy Orchestrator

So far how to price the worker and PS is investigated, we next discuss how to efficiently find the optimal schedule l^* for an admitted job j. This optimization problem is equivalent to the following one:

$$\max_{a_j, \hat{t}_j, o_{jw}, o_{jp}, \psi_j, \mathbf{y}, \mathbf{z}} f_j(\hat{t}_j - r_j) - \sum_{s\in[S]} \sum_{w\in[W]} \sum_{t\in[T]} \alpha_{sw}(t) y_{jsw}(t) - \sum_{s\in[S]} \sum_{p\in[P]} \sum_{t\in[T]} \beta_{sp}(t) z_{jsp}(t) \quad (21)$$

subject to:

$$q_{sw}(t) + y_{jsw}(t) \leq C_{sw}, \forall s, \forall w, \forall t \quad (21a)$$

$$h_{sp}(t) + z_{jsp}(t) \leq C_{sp}, \forall s, \forall p, \forall t \quad (21b)$$

Constraints $(2) - (9)(12) - (15)(16a)$, where $x_j = 1$.

In problem (21), we need to find a tuple of $o_{jw}, o_{jp}, y_{jsw}(t), z_{jsp}(t)$ to maximize the objective value. If we fix worker type w, PS type p, start time a_j and the number of workers needed, as a result, the completion time $\hat{t}_j$ is determined. Consequently, (21) is simplified to the following integer program (IP):

$$\min_{\psi_j, \mathbf{y}, \mathbf{z}} cost(a_j, \hat{t}_j, w, p) = \sum_{s\in[S]} \sum_{w\in[W]} \sum_{t\in[T]} \alpha_{sw}(t) y_{jsw}(t) + \sum_{s\in[S]} \sum_{p\in[P]} \sum_{t\in[T]} \beta_{sp}(t) z_{jsp}(t) \quad (22)$$

subject to:

$$y_{jsw}(t) \leq C_{sw} - q_{sw}(t), \forall s, \forall w, \forall t \in [a_j, \hat{t}_j] \quad (22a)$$

$$z_{jsp}(t) \leq C_{sp} - h_{sp}(t), \forall s, \forall p, \forall t \in [a_j, \hat{t}_j] \quad (22b)$$

$$y_{jsw}(t), z_{jsp}(t) \in \{0, 1, 2, ...\}, \forall s, \forall w, \forall p, \forall t \in [a_j, \hat{t}_j] \quad (22c)$$

Constraints $(3) - (9)(12)(13)$,

where $x_j = 1, o_{jw} = 1, o_{jp} = 1,$ specific $a_j, \hat{t}_j$

In problem (22), the key is to decide where to optimally place $\lceil E_j D_j M_j k_j^{wp}/(\hat{t}_j - a_j)\rceil$ workers and corresponding PSs to minimize the resource cost. For distributed and centralized placement, deployment solutions of workers and PSs are different. Thus, we come up with two greedy algorithms discussed in Alg. 3 and Alg. 4, to resolve the above two cases, respectively. The better one of these two results is the optimal solution of problem (22). These comparisons are concluded in Alg. 2 and specific steps are: enumerate worker type $w \in [W]$, PS type $p \in [P]$, start time a_j from r_j to T and the number of allocated workers from 1 to D_j; given above information, compute optimal deployment of workers and PSs via two greedy approaches.

4.4 Algorithm Design

Based on the aforementioned analysis, the online framework DPS is sketched in Alg. 1. In DPS, it simulates an optimal schedule l^* for each incoming job j by invoking Alg. 2 (line 3). If the job's payoff is positive, then the ML operator truly schedules this job and updates the usage and prices of workers and PSs according to l^* (lines 5-6), otherwise rejects this job (line 8).

Algorithm 1 Online Framework for Pricing and Scheduling ML Jobs: DPS

Input: $T, C_{sw}, C_{sp}, \forall s \in [S], \forall w \in [W], \forall p \in [P]$
Output: $x_j, o_{jw}, o_{jp}, y_{jsw}(t), z_{jsp}(t), \forall j \in [J], \forall s \in [S], \forall w \in [W], \forall p \in [P], \forall t \in [T]$

1: Initialize $y_{jsw}(t) = 0, z_{jsp}(t) = 0, q_{sw}(t) = 0, h_{sp}(t) = 0, \alpha_{sw}(t) = \alpha_{sw}(0), \beta_{sp}(t) = \beta_{sp}(0), \forall j, \forall s, \forall w, \forall p, \forall t$
2: **for** $j = 1, 2, ...$ **do**
3: Set $(l^*, \mu_j) = A_{bestsche}(r_j, \{q_{sw}(t)\}, \{h_{sp}(t)\})$
4: **if** $\mu_j > 0$ **then**
5: Update $q_{sw}(t) = q_{sw}(t) + \sum_j y_{jsw}(t), h_{sp}(t) = h_{sp}(t) + \sum_j z_{jsp}(t), \alpha_{sw}(t) = \alpha_{sw}(q_{sw}(t)), \beta_{sp}(t) = \beta_{sp}(h_{sp}(t)), \forall s, \forall w, \forall p, \forall t$
6: Set $x_j = 1$ and launch job j according to schedule l^*
7: **else**
8: Set $x_j = 0$ and reject job j
9: **end if**
10: **end for**

For job j, its optimal schedule is computed by Alg. 2. We enumerate worker type, PS type, start time and the number of allocated workers (line 2), and pass these data as arguments of functions COST_D and COST_C to compute minimal resource cost and corresponding schedule l under different placement assumptions (lines 3-4). Then from all candidate schedules, we select the one with highest payoff

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

Algorithm 2 Subroutine for Finding Optimal Schedule of Job j: $A_{bestsche}$

Input: $r_j, q_{sw}(t), h_{sp}(t), C_{sw}, C_{sp}, \forall s \in [S], \forall w \in [W], \forall p \in [P], \forall t \in [r_j, \hat{t}_j]$
Output: l^*, μ_j

1: Initialize $\mu_j = 0, l^* = \emptyset, o_{jw} = 0, o_{jp} = 0, y_{jsw}(t) = 0, z_{jsp}(t) = 0, \forall s, \forall w, \forall p, \forall t$
2: **for** each tuple $\{w, p, a_j, D_w\}, \forall w \in [W], p \in [P], a_j \in [r_j, T], D_w \in [1, D_j]$ **do**
3: Set $(cost, l) = \text{COST\_C}(w, p, a_j, D_w)$
4: Set $(cost', l') = \text{COST\_D}(w, p, a_j, D_w)$
5: **if** $f_j(\hat{\tau}_j^l) - cost < f_j(\hat{\tau}_j^{l'}) - cost'$ **then**
6: Set $(cost, l) = (cost', l')$
7: **end if**
8: $\mu_{jl} = f_j(\hat{\tau}_j^l) - cost$
9: **if** $\mu_{jl} > 0 \wedge \mu_{jl} > \mu_j$ **then**
10: $l^* \leftarrow \{w, p, l\}, \mu_j = \mu_{jl}$
11: **end if**
12: **end for**
13: **Return** l^*, μ_j

as the optimal schedule and record corresponding parameters (lines 5-11).

Functions COST_C and COST_D produce the optimal worker and PS deployment under centralized and distributed placement assumptions, respectively. For function COST_C, given the number of workers needed, the training duration d_j and completion time $\hat{t}_j$ are computed in line 3. Subsequently, we compute all feasible allocations and sort corresponding servers in non-decreasing order of resource cost in line 4. A feasible allocation should satisfy the following constraints. Note that only after job j arrived in server s at time t can it get workers from this server ($r_j + \Delta_{js} < a_j$ in line 4). Moreover, we only need one PS to cover the bandwidth of all workers due to centralized placement. So far the number of all needed workers and PSs is determined. Next we select the server which can accommodate all needed workers and PSs ($\min_{t\in[a_j,\hat{t}_j]}\{C_{sw} - q_{sw}(t)\} \geq D_w \wedge \min_{t\in[a_j,\hat{t}_j]}\{C_{sp} - h_{sp}(t)\} \geq 1$ in line 4) while its total resource cost is minimal.

Similarly, in function COST_D, we first compute the duration and completion time (lines 3). Next we sort servers in non-decreasing order of worker prices $\alpha_{sw}(t)$ (line 4), and deploy as many workers as possible starting from the cheapest server, while respecting capacity constraint (22a) and the upper bound D_j in constraint (2) (lines 5-8). Note that we can only use the servers that job j can reach at time a_j. Then, we divide the servers into two sets based on the last used server s_{s_h}, that is, $[s_1, s_h - 1]$ and $[s_h, H]$ respectively. According to constraint (12), only one server is needed to deploy PSs. As a result, we assume this server is from the above two sets respectively and discuss corresponding schedules in lines 14-19 and 20-35, respectively. For set $[s_1, s_h - 1]$, the number of PSs is computed respecting capacity constraint (22a) (line 16), bandwidth constraint (12) (line 15) and other constraints (5)-(7). We record resulted cost and corresponding allocation into set $\mathcal{L}$ (line 17). Subsequently, we analyze the another case. For any server $i_1 \in [s_h, H]$, if the resource cost after moving a worker form servers in $[s_1, s_h - 1]$ to i_1 decreases (line 24), then we move as many workers as possible to server i_1 (lines 23-34). Likewise, the results are stored into $\mathcal{L}$ (line 30). Finally, of all candidates in $\mathcal{L}$, the one with minimal cost is optimal (line 39).

Algorithm 3 Function for Scheduling Job j Under Distributed Placement: COST_D

1: **function** COST_D(w, p, a_j, D_w)
2: Initialize $min\_cost = \infty, min\_cost' = \infty, l = \emptyset, l' = \emptyset, cost = 0, cost_w = 0, y_{jsw}(t) = 0, z_{jsp}(t) = 0, \mathcal{L} = \emptyset, \forall s, \forall w, \forall p$
3: Compute $d_j = \lceil \frac{E_j D_j M_j k_j^{wp}}{D_w} \rceil, D'_w = D_w, \hat{t}_j = a_j + d_j$
4: Sort servers in $[H] = \{s | r_j + \Delta_{js} < a_j\}$ according to $\alpha_{sw}(t)$ in non-decreasing order into $s_1, s_2, ..., s_H, \hat{t}_j = a_j + d_j$
5: **for** $i = 1$ to H **do** ▷ deploy workers
6: Set $y_{js_iw}(t) = \min\{\min_{t\in[a_j,\hat{t}_j]}\{C_{s_iw} - q_{s_iw}(t)\}, D_j - \sum_{i'=1}^{i-1} y_{js_{i'}w}(t), D'_w\}, \forall t \in [a_j, \hat{t}_j]$
7: $D'_w \leftarrow D'_w - y_{js_iw}(a_j)$
8: **end for**
9: **if** $D'_w > 0$ **then** ▷ no enough workers
10: **Return** ∞, l
11: **end if**
12: Compute $cost_w = \sum_{t\in[a_j,\hat{t}_j]}\{\sum_s \alpha_{sw}(t) y_{jsw}(t)\}$
13: Record the last used server as s_{s_h}
14: **for** $i = 1$ to $s_h - 1$ **do**
15: Set $D_p = \lceil (D_w - y_{js_iw}(a_j)) b_w / b_p \rceil$
16: **if** $\max\{D_p, 1\} \leq \min_{t\in[a_j,\hat{t}_j]}\{C_{s_ip} - h_{s_ip}(t)\}$ **then**
17: Set $z'_{js_ip}(t) = D_p, \forall t \in [a_j, \hat{t}_j], cost = cost_w + \sum_{t\in[a_j,\hat{t}_j]}\{\sum_s \beta_{s_ip}(t) D_p\}, \mathcal{L} = \mathcal{L} \cup \{cost, l \leftarrow \{\mathbf{y}, \mathbf{z}'\}\}$
18: **end if**
19: **end for**
20: **for** $i_1 = s_h$ to H **do**
21: Set $D_w(s_{i_1}) = \min_{t\in[a_j,\hat{t}_j]}\{C_{s_{i_1}w} - q_{s_{i_1}w}(t) - y_{js_{i_1}w}(t)\}, D_p(s_{i_1}) = \min_{t\in[a_j,\hat{t}_j]}\{C_{s_{i_1}p} - h_{s_{i_1}p}(t)\}, \mathbf{y}' = \mathbf{y}, \mathbf{z}' = \emptyset$
22: Compute $cost = cost_w + \sum_{t\in[a_j,\hat{t}_j]}\{\beta_{s_{i_1}p}(t) b_w (D_w - y'_{js_{i_1}w}(t))/b_p\}$
23: **for** $i_2 = s_h$ to s_1 **do**
24: **if** $\alpha_{s_{i_1}w}(a_j) - \alpha_{s_{i_2}w}(a_j) - b_w/b_p \beta_{s_{i_1}p}(a_j) \leq 0 \wedge D_w(s_{i_1}) > 0$ **then**
25: Set $\Delta W = \min\{y'_{js_{i_2}w}(a_j), D_w(s_{i_1})\}, D_w(s_{i_1}) = D_w(s_{i_1}) - \Delta W$
26: Compute $cost = cost + (\alpha_{s_{i_1}w}(a_j) - \alpha_{s_{i_2}w}(a_j) - b_w/b_p \beta_{s_{i_1}p}(a_j)) \Delta W$
27: Set $y'_{js_{i_2}w}(t) = y'_{js_{i_2}w}(t) - \Delta W, y'_{js_{i_1}w}(t) = y'_{js_{i_1}w}(t) + \Delta W, z'_{js_{i_1}p}(t) = \lceil (D_w - y'_{js_{i_1}w}(t) - \Delta W) b_w / b_p \rceil, \forall t \in [a_j, \hat{t}_j]$
28: **else**
29: **if** $\sum_s \sum_p z'_{js_{i_1}p}(a_j) \leq D_p(s_{i_1})$ **then**
30: $\mathcal{L} \leftarrow \mathcal{L} \cup \{cost, l \leftarrow \{\mathbf{y}, \mathbf{z}\}\}$
31: **end if**
32: **break**
33: **end if**
34: **end for**
35: **end for**
36: **if** $\mathcal{L} = \emptyset$ **then** ▷ no enough PSs
37: **Return** ∞, l
38: **end if**
39: Sort $\mathcal{L}$ according to $cost$ in non-decreasing order and record the first element as $\mathcal{L}_1$
40: **Return** $\mathcal{L}_1$
41: **end function**

4.5 Theoretical Analysis

Theorem 1. (Correctness). *DPS generates a feasible solution of the original problem in (16).*

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

Algorithm 4 Function for Scheduling Job j Under Centralized Placement: COST_C

1: **function** COST_C(w, p, a_j, D_w)
2: Initialize $min\_cost = \infty, l = \emptyset, y_{jsw}(t) = 0, z_{jsp}(t) = 0, \forall s, \forall w, \forall p$
3: Compute $d_j = \lceil E_j D_j M_j k_j^{wp} / D_w \rceil, \hat{t}_j = a_j + d_j$
4: Sort servers in $[H] = \{s | r_j + \Delta_{js} < a_j \wedge \min_{t\in[a_j,\hat{t}_j]}\{C_{sw} - q_{sw}(t)\} \geq D_w \wedge \min_{t\in[a_j,\hat{t}_j]}\{C_{sp} - h_{sp}(t)\} \geq 1\}$ according to $\alpha_{sw}(a_j)D_w + \beta_{sp}(a_j)$ in non-decreasing order into $s_1, s_2, ..., s_H$
5: **if** $H == 0$ **then** ▷ no enough workers or PSs
6: **Return** ∞, l
7: **else**
8: Set $y_{js_1w}(t) = D_w, z_{js_1p}(t) = 1, \forall t \in [a_j, \hat{t}_j]$
9: Set $min\_cost = \sum_{t\in[a_j,\hat{t}_j]}\{\alpha_{s_1w}(t)D_w + \beta_{s_1p}(t)\}, l \leftarrow \{\mathbf{y}, \mathbf{z}\}$
10: **Return** min_cost, l
11: **end if**
12: **end function**

Proof. The solution of Alg. 1 consists of each job j' solution produced by Alg. 2. Thus, we only need to prove the feasibility of Alg. 2. Observing Alg. 2, it computes two solutions by invoking functions COST_C and COST_D given worker type, PS type, start time and the number of allocated workers. First of all, Alg. 2 must make sure that functions COST_C and COST_D are feasible. In both functions, parameter D_w is between $[1, D_j]$, which complies with constraint (3). When selecting servers to deploy D_w workers, we ensure the allocation not violate capacity constraint (22a), and job can reach these servers at time $t \in [a_j, \hat{t}_j]$ (satisfying constraint (13)). Subsequently we select enough PSs to cover the bandwidth of allocated workers (constraints (6) and (12)), and under the limitation of capacity (constraints (22b) and (5)). The deployment of workers and PSs is determined at time a_j, and remains unchanged later. So the allocation complies with constraints (4) and (7). And constraint (8) is apparently satisfied according the computation of duration d_j. Moreover, COST_C and COST_D are designed by assuming $\psi_j = 1$ and $\psi_j = 0$, respectively. In summary, the two functions produce feasible $\psi_j, \mathbf{y}, \mathbf{z}$. Back to Alg. 2, it only picks the better allocation in lines 8-10, *i.e.*, the optimal solution of problem (22). From lines 2-3, the solution of Alg. 2 apparently satisfies constraints (2), (3), (15) and (14). Consequently, Alg. 2 computes a feasible optimal solution for each job. In conclusion, Alg. 1 by calling Alg. 2 produces a feasible solution for our ML job pricing and scheduling problem. □

Theorem 2. (Optimality). *Alg. 2 produces an optimal solution of problem (21), where* COST_C *and* COST_D *compute an optimal solution of problem (22).*

Proof. To prove the optimality of Alg. 2, we first prove that functions COST_C and COST_D produce optimal solutions given the worker type, PS type, start time and the number of allocated workers, respectively.

1) Optimality of COST_C

In this case, all workers and PSs are placed in one server. We can only select from the candidate servers whose available resources can afford jobs' demand. The duration is determined by the given number of workers, and we only need one PS to satisfy constraints (5)(6)(12). And the number of workers and PSs is fixed and independent with servers. As a result, the server with the minimal value of $\alpha_{sw}(t)D_w + \beta_{sp}(t)$ is exactly the optimal one.

2) Optimality of COST_D

Note that $d_j = \lceil E_j D_j M_j k_j^{wp} / D_w \rceil$ is the duration by training with D_w workers. We select D_w workers from servers with the cheapest worker prices. Next we choose servers to deploy PSs for covering the bandwidth of allocated workers. According to constraint (12), it is apparently optimal to choose only one server since constraint (12) could be satisfied by any server that could deploy needed PSs, including the selected one. Thus, the key of finding the optimal schedule is to pick out one server to place PSs and part of workers, the remaining workers are naturally deployed from the cheapest servers. In Alg. 3, we divide candidate servers into two categories. For the former class, it is easy to calculate corresponding schedule. For any server i_2 in the other class, if we move one worker from $i_1 \in [s_1, s_h]$ to i_2, then i_2 can remove b_w/b_p PSs. As a result, the change in resource cost is computed as $\alpha_{i_2w}(a_j) - \alpha_{i_1w}(a_j) - b_w/b_p\beta_{i_2p}(a_j)$. And $\alpha_{i_2w}(a_j) - \alpha_{i_1w}(a_j) - b_w/b_p\beta_{i_2p}(a_j) < 0$, which means moving helps user reduce cost. So we move as many workers as possible in order to minimize the cost. When all candidate servers are traversed, the one with minimal cost is naturally the optimal schedule.

3) Optimality of Alg. 2

In Alg. 2, We enumerate all possible worker type, PS type, start time and the number of allocated workers, and update the best schedule only when the payoff of new schedule is smaller than the current one. The final result is naturally optimal. Then we can conclude this proof. □

Theorem 3. (Polynomial Time). *The time complexity of DPS is* $O(JWPTD_j(S \log S + S^2))$.

Proof. We first analyze the time complexity of functions COST_C and COST_D. For function COST_C, the running time is mainly produced by line 4 whose time complexity is $O(H \log H)$. The execution time of *loop* in line 5 is at most $O(H)$. In a word, the time complexity of function COST_C is $O(H \log H)$. For function COST_D, line 4 and 39 take $O(H \log H)$ steps for sorting and *loop* in line 5 and 14 spends at most $O(H)$ time. From line 20 to 35, it is a *double loop*. The iterations of outer *loop* in line 20 is $H - s_h$, and that of inner *loop* in line 23 is s_h. When $s_h = H/2$, the iterations is maximized, which is H^2. Consequently, the time complexity of this double loop is limited to $O(H^2)$. And other lines can be executed at constant time. Moreover, H is at most S. To sum up, the time complexity of function COST_D is also $O(S \log S + S^2)$.

Next, we talk about time complexity of Alg. 2. There is a *quadruple loop* in line 2, the time complexity of which is $O(WPTD_j)$. For each tuple of parameters, Alg. 2 calls functions COST_C and COST_D, and then compares their returned results and the temporary optimal objective value. These comparisons only spend constant time in each iteration. Thus, the time complexity of finding optimal schedule is $O(WPTD_j(S \log S + S^2))$.

As far as Alg. 1, each statement runs at most $O(TSW)$ or $O(TSP)$ times. In conclusion, the time complexity of Alg. 1 is $O(JWPTD_j(S \log S + S^2))$. □

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

Theorem 4. (Truthfulness). *The online auction in DPS is truthful.*

Proof. The parameters submitted by each job owner j include $\{r_j, f_j(\cdot), \{D_j, M_j, E_j\}, \{\Delta_{js}^{\uparrow}\}_{\forall s}, \{k_j^{wp}\}_{\forall w, \forall p}\}$. We demonstrate that our algorithm DPS can prevent users from reporting false parameters as follows.

(Truthfulness in the arrival time r_j). We first discuss this case where user j reports an earlier arrival time r'_j. If the start processing time a_j lies in $[r'_j, r_j)$, then the time span $[a_j, r_j)$ is wasted since the job cannot be processed before the actual arrival time r_j. As a result, the remaining time is insufficient to complete the job. Without enough training time, the model produced by the job cannot reach the desired accuracy. As a result, the job owner cannot obtain utility from the failed job but must pay the resource cost. If the start time a_j is not earlier than r_j, such misreporting will not affect the resource allocation and training process. Consequently, the job owner earns the same profit as being honest. Next, we discuss another case where the user delays submitting a job. According to Algorithms 2, 3 and 4, we compute an optimal schedule with maximum job profit, where the start time $a_j \in [r_j, T]$. If the user submits its job at a later time r'_j, the optimal schedule is extracted between $[r'_j, T]$. In the best case where the optimal start time a_j lies in $[r'_j, T]$, postponing job submission still generates the same schedule and thus the same profit as honest reporting. If $a_j \in [r_j, r'_j)$, misreporting the arrival time will incur a sub-optimal schedule, thus decreasing the user profit. Therefore, users will not misreport the arrival time of jobs.

(Truthfulness in utility functions $f_j(\cdot)$). Our auction DPS is one instance of posted pricing mechanisms [47]. For each job j, if it is admitted, then the payment paid to the ML operator is computed as $p_j = \sum_{s\in[S]}\sum_{w\in[W]}\sum_{t\in[T]}\alpha_{sw}(t)\gamma_{jsw}^{l}(t)$ $(\sum_{s\in[S]}\sum_{p\in[P]}\sum_{t\in[T]}\beta_{sp}(t)\zeta_{jsp}^{l}(t))$. We can observe that the formula's value is only related to the amount of resource consumed and resource price, not to the bidding price of j. In addition, the profit of j equals its valuation minus the payment, i.e., $u_j(f_j(\cdot)) = v_j(\cdot) - p_j$, which is also independent of j's bidding price. As a result, j cannot gain more profit by lying about its bidding price.

(Truthfulness in job workload $\{D_j, M_j, E_j\}$ and k_j^{wp}). If user j understates its workload/processing time ($\{D_j, M_j, E_j\}$) or the time k_j^{wp} to process a mini-batch, then the allocated resources and occupation time are not enough to complete the job. If the user overstates $\{D_j, M_j, E_j\}$ or k_j^{wp}, it will receive more resources than it actually needs. According to Theorem 2, we know that Algorithms 2-4 generate an optimal resource allocation for each job. Thus, allocating more resources to the job by overstating $\{D_j, M_j, E_j\}$ or k_j^{wp} is a sub-optimal choice for the job owner, *i.e.*, the resulting profit decreases due to non-increasing job utility and more resource cost. Therefore, we can conclude that misreporting the workload is not a dominant user strategy.

(Truthfulness in upload delay $\Delta_{js}^{\uparrow}$). Similar to the arrival time r_j, the upload delay $\Delta_{js}^{\uparrow}$ affects the job's start time a_j. As a result, the effects of misreporting $\Delta_{js}^{\uparrow}$ on job profit are similar to that of a false r_j. We will not repeat the discussion here.

In summary, the proposed framework DPS is truthful.

Theorem 5. (Individual Rationality). *The online auction in DPS is individually rational.*

Proof. According to the admission control in Sec. 4.1, a job is admitted if and only if $\mu_j > 0$. That is $f_j(\hat{\tau}_j^l - r_j) - (\sum_s\sum_w\sum_t \alpha_{sw}(t)\gamma_{jsw}^l(t) + \sum_s\sum_p\sum_t \beta_{sp}(t)\zeta_{jsp}^l(t)) > 0$, where the first item is the value of job j according to Theorem 4 and the second item is the payment that user j should pay the ML operator for resource usage. It is clear that μ_j is exactly $u_j(f_j(\cdot)) \geq 0$ in Definition 3, and our auction with the admission control is individually rational.□

Theorem 6. (Competitive Ratio). *The* competitive ratio *is the upper-bound ratio of the optimal social welfare to the social welfare computed by our online algorithm, DPS. For problem in (16), DPS is* $2(\log\eta_1 + \log\eta_2)$*-competitive, where* $\eta_1 = \max_w\{\eta_w\}, \eta_2 = \max_p\{\eta_p\}$.

Since the original problem (16) is equivalent to (17), we can achieve its competitive ratio by analyzing (17)'s. Next, we adopt the primal-dual theory to analyze the competitive ratio of (17). When $t = 0$, we can easily find that the objective values of (17) and (18) are both zero. If we can bound the ratio of the objective value difference of (18) to that of (17) by a constant, then the constant is exactly the competitive ratio of (17). If a job is rejected, the increments in the objective values of (17) and (18) are both zero. As a result, we only need to analyze the case that a job is accepted. By observing the objective of (17) and (18), we know that their values are significantly affected by the price changes before and after accepting a job. Therefore, we focus on the analysis of the price changes in the proof of our competitive ratio.

Proof. Let OPT be the optimal objective value of (16) and (17). And let P_j and D_j denote the objective value of (17) and (18), computed by Alg. 1 after handling job j. At the beginning of auction, *i.e.*, there are no jobs, their objective values are denoted by P_0 and D_0, respectively. Note that $P_0 = 0$ and $D_0 = 0$. After dealing with all jobs, the final objective values returned by Alg. 1 are denoted by P_J and D_J. Then we have the following lemmas and definitions.

Lemma 1. *If there exists a constant* η *such that* $P_j - P_{j-1} \geq \frac{1}{\eta}(D_j - D_{j-1}), \forall j \in [J]$, *and if* $P_0 = D_0 = 0$, *then Alg. 1 is* η*-competitive in total social welfare.*

Proof. Recalling the definition of P_J, we have $P_J = \sum_{j\in[J]}(P_j - P_{j-1})$. Similarly, $D_J - D_0 = \sum_{j\in[J]}(D_j - D_{j-1})$. Then we derive

$$P_J = \sum_{j\in[J]}(P_j - P_{j-1}) \geq \frac{1}{\eta}\sum_{j\in[J]}(D_j - D_{j-1}) = \frac{1}{\eta}(D_J - D_0)$$

In addition, we have $D_J \geq OPT \geq P_J$ in accordance to weak duality. Combining $P_0 = D_0 = 0$, we can derive $P_J \geq \frac{1}{\eta}D_J \geq \frac{1}{\eta}OPT$. Consequently, we can conclude the proof. □

Next we design a feasible η, which makes Alg. 1 satisfy $P_j - P_{j-1} \geq \frac{1}{\eta}(D_j - D_{j-1}), \forall j \in [J]$. For the case that job j is rejected, we have $P_j - P_{j-1} = D_j - D_{j-1} = 0$, then the inequality holds. We next discuss another case where job j is admitted with schedule l, *i.e.*, $\chi_j^l = 1$. We introduce new variables $\alpha_{sw}^j(t)$ $(\beta_{sp}^j(t))$ to denote the price of type-w workers (type-p PSs) on server s at time t after processing

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

job j. Let $q^j_{sw}(t)$ ($h^j_{sp}(t)$) be the number of type-w workers (type-p PSs) consumed on server s at time t after handing job j.

$$\alpha^j_{sw}(t) - \alpha^{j-1}_{sw}(t) = (\eta_w^{\frac{q^{j-1}_{sw} + \gamma^l_{jsw}(t)}{C_{sw}}}) - (\eta_w^{\frac{q^{j-1}_{sw}}{C_{sw}}}) = \eta_w^{\frac{q^{j-1}_{sw}}{C_{sw}}} (2^{\log \eta_w \frac{\gamma^l_{jsw}(t)}{C_{sw}}} - 1)$$

The inequality $2^x - 1 \leq x$ holds for any $x \in [0, 1]$, then we have

$$\alpha^j_{sw}(t) - \alpha^{j-1}_{sw}(t) \leq (\alpha^{j-1}_{sw}(t) + 1) \log \eta_w^{\frac{\gamma^l_{jsw}(t)}{C_{sw}}}$$

Similarly, we have

$$\beta^j_{sp}(t) - \beta^{j-1}_{sp}(t) \leq (\beta^{j-1}_{sp}(t) + 1) \log \eta_p^{\frac{\zeta^l_{jsp}(t)}{C_{sp}}}$$

Since $\chi^l_j = 1$, the constraint (18a) is tight according to complementary slackness. The increment of D_j is

$$\begin{aligned}
D_j - D_{j-1} &= \mu_j + \sum_s \sum_w \sum_t (\alpha^j_{sw}(t) - \alpha^{j-1}_{sw}(t)) C_{sw} \\
&\quad + \sum_s \sum_p \sum_t (\beta^j_{sp}(t) - \beta^{j-1}_{sp}(t)) C_{sp} \\
&\leq f_j(\hat{\tau}^l_j - r_j) - \sum_s \sum_w \sum_t \alpha^{j-1}_{sw}(t) \gamma^l_{jsw}(t) \\
&\quad - \sum_s \sum_p \sum_t \beta^{j-1}_{sp}(t) \zeta^l_{jsp}(t) \\
&\quad + \sum_s \sum_w \sum_t ((\alpha^{j-1}_{sw}(t) + 1) \log \eta_w^{\frac{\gamma^l_{jsw}(t)}{C_{sw}}}) C_{sw} \\
&\quad + \sum_s \sum_p \sum_t ((\beta^{j-1}_{sp}(t) + 1) \log \eta_p^{\frac{\zeta^l_{jsp}(t)}{C_{spw}}}) C_{sp} \\
&= f_j(\hat{\tau}^l_j - r_j) - \sum_s \sum_w \sum_t \alpha^{j-1}_{sw}(t) \gamma^l_{jsw}(t) \\
&\quad - \sum_s \sum_p \sum_t \beta^{j-1}_{sp}(t) \zeta^l_{jsp}(t) \\
&\quad + \sum_s \sum_w \sum_t (\alpha^{j-1}_{sw}(t) + 1) \log \eta_w \gamma^l_{jsw}(t) \\
&\quad + \sum_s \sum_p \sum_t (\beta^{j-1}_{sp}(t) + 1) \log \eta_p \zeta^l_{jsp}(t) \\
&= f_j(\hat{\tau}^l_j - r_j) \\
&\quad + (\log \eta_1 - 1) \sum_s \sum_w \sum_t \alpha^{j-1}_{sw}(t) \gamma^l_{jsw}(t) \\
&\quad + (\log \eta_2 - 1) \sum_s \sum_p \sum_t \beta^{j-1}_{sp}(t) \zeta^l_{jsp}(t) \\
&\quad + \log \eta_1 \sum_s \sum_w \sum_t \gamma^l_{jsw}(t) \\
&\quad + \log \eta_2 \sum_s \sum_p \sum_t \zeta^l_{jsp}(t)
\end{aligned}$$

where $\eta_1 = \max_w\{\eta_w\}, \eta_2 = \max_p\{\eta_p\}$. According to $f_j(\hat{\tau}^l_j - r_j) - \sum_s \sum_w \sum_t \alpha^{j-1}_{sw}(t) \gamma^l_{jsw}(t) - \sum_s \sum_p \sum_t \beta^{j-1}_{sp}(t) \zeta^l_{jsp}(t) = \mu_j \geq 0$, we have $f_j(\hat{\tau}^l_j - r_j) \geq \sum_s \sum_w \sum_t \alpha^{j-1}_{sw}(t) \gamma^l_{jsw}(t)$ and $f_j(\hat{\tau}^l_j - r_j) \geq \sum_s \sum_p \sum_t \beta^{j-1}_{sp}(t) \zeta^l_{jsp}(t)$. Combining assumptions $\frac{f_j(\hat{\tau}^l_j - r_j)}{\sum_s \sum_w \sum_t \gamma^l_{jsw}(t)} \geq 1$ and $\frac{f_j(\hat{\tau}^l_j - r_j)}{\sum_s \sum_p \sum_t \zeta^l_{jsp}(t)} \geq 1$, we can derive that $D_j - D_{j-1} \leq 2(\log \eta_1 + \log \eta_2) f_j(\hat{\tau}^l_j - r_j)$. Moreover, $P_j - P_{j-1} = f_j(\hat{\tau}^l_j - r_j)$. To sum up, we can conclude that Alg. 1 complies with Lemma 1, where $\eta = 2(\log \eta_1 + \log \eta_2)$.□

Next, we analyze the typical values of η_w and η_p in practice. Based on the discussion on Equation (20), we have $\eta_w = \max_j\{\frac{f_j(\hat{\tau}^l_j - r_j)}{\sum_s \sum_w \sum_t \gamma^l_{jsw}(t)}\} + 1$ and $\eta_p = \max_j\{\frac{f_j(\hat{\tau}^l_j - r_j)}{\sum_s \sum_p \sum_t \zeta^l_{jsp}(t)}\} + 1$. The values of η_w and η_p depend on the utility function $f_j(\hat{\tau}^l_j - r_j)$ and the resource demands $\gamma^l_{jsw}(t), \zeta^l_{jsp}(t)$. We first discuss the typical value of η_w. In practice, the training of ML jobs is conducted in GPU clusters, and a GPU is usually regarded as a worker. According to the real-world trace of deep learning workload [Philly][Helios], the practical GPU demands of ML jobs are typically within $\{1, 2, 4, 8, 16\}$, *i.e.*, $\gamma^l_{jsw}(t) \in \{1, 2, 4, 8, 16\}$. We might as well let $\gamma^l_{jsw}(t) = 2$, which is the most common number of GPUs required by jobs. Moreover, according to the Philly traces [Helios], the training time of deep learning jobs ranges from a few minutes to several days. Here, we may use the average training time for further discussion. Recall that we set a time slot to be one hour long, then the practical average training time can be around 3 time slots. As a result, $\sum_s \sum_w \sum_t \gamma^l_{jsw}(t) = 2 * 3 = 6$. Besides, we assume there is no queuing delay, resulting in the maximum possible value of $f_j(\hat{\tau}^l_j - r_j)$. In Sec. 6.2.2, we verify the algorithm performance using three kinds of utility functions. In the later analysis, we might as well select the sigmoid function, *i.e.*, $f_j(\hat{\tau}^l_j - r_j) = \frac{100\kappa}{1+e^{0.02(\hat{\tau}^l_j - r_j)}}, \kappa \in [1, 5]$. Let $\kappa = 3$, then $f_j(\hat{\tau}^l_j - r_j) \approx 145$. By summarizing the above results, $\eta_w \approx 145/6 + 1 \approx 25.17$. Using similar analysis, we have $\eta_p \approx 49.33$. Then the competitive ratio can be approximated as $\gamma = 2 * (log\eta_1 + log\eta_2) \approx 14.25$. This numerical value is the theoretical bound of the worse case. We have measured the actual bound in Fig. 10, which is lower than 1.6 and much better than the theoretical one.

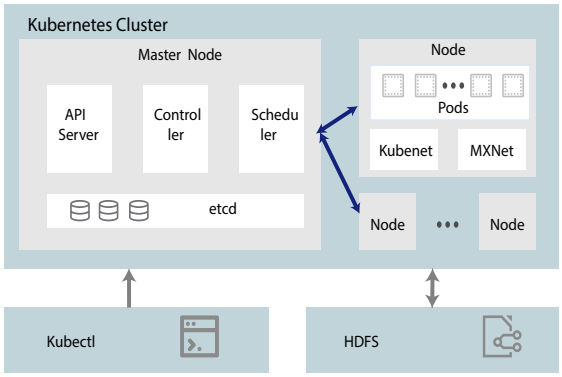

Fig. 4: Overall architecture of the testbed.

5 Implementation

We use Kubernetes 1.19 [22] to implement a distributed ML system based on MXNet [21] framework. The overall architecture of our system is depicted in Fig. 4. Our testbed is deployed on 8 physical machines (Slave Nodes) equipped with 1 GeForce RTX 2060 GPU, 12 CPU cores, 16GB RAM, 500GB HDDs and a dual-port 1GbE NIC, which serve as edge servers. Especially, another physical machine (Master Node) serves as both the remote cloud and a central scheduler. All training data and parameters exchanged between workers and PSs are stored by a shared Hadoop Distributed

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

File System (HDFS) [47]. Note that the size of one data chunk is set to 2MB. To achieve the dynamic deployment of workers, the model parameters of jobs in each time slot are checkpointed and stored in HDFS [48], [49]. All workers deployed in each time slot reload the recent checkpointed model parameters from HDFS before start training.

6 Evaluation

In this section, we first introduce the setup of implemented testbed, simulator and baselines in Sec. 6.1, and then present the performance results in Sec. 6.2.

6.1 Experimental Setup

Testbed Setup. As shown in Table. 2, we generate 6 types of ML jobs with three respective datasets CIFAR10, Caltech101 and ImageNet. Especially, we pick 7000 images (101.8MB) from ImageNet ILSVRC2012 dataset to form a smaller scale dataset, *i.e.*, ImageNet-12. Each type of job has a corresponding CNN model, *i.e.*, ResNet-50, ResNet-101, LeNet, GoogLeNet, AlexNet or Inception-BN. Note that all workers and PSs are implemented on Docker containers. We select five types of workers (W) and three types of PSs (P), each equipped with 0 to 1 GPU, 1 to 4 CPU cores and 3 to 6GB RAM. The processing capacity of job i with each type of worker (*i.e.*, e_j^w) is obtained by pre-training. We record and identify the value of processing capacity for each type of jobs with different types of workers. The number of epochs for jobs (E_j) is randomly picked from range $[20, 60]$. Each experiment runs for 30 time slots (*i.e.*, $T = 30$), and the length of each time slot is set to 20 minutes. In our experiments, the arrival pattern of ML jobs follows a uniform distribution.

TABLE 2. Machine learning jobs used for experiments

Model	Dataset	# of Data chunks/ Mini-batches
ResNet-50 [50]	CIFAR10 [51]	27 / 58
ResNet-101	CIFAR10	27 / 58
GoogLeNet [52]	Caltech101 [53]	115 / 58
LeNet [54]	Caltech101	115 / 58
AlexNet [55]	ImageNet-12 [56]	60 / 58
Inception-BN [57]	ImageNet-12	60 / 58

Simulator. We develop a simulator to evaluate the performance of proposed algorithm DPS. In the simulator, we generate the workload of jobs with similar settings in our testbed. For each job, E_j is set within [20, 60], D_j is in [27, 115], M_j is in [10, 58], e_j^w is in [0.001, 0.05] time slots, g_j^p is in [10, 100] milliseconds, m_j is in [30, 575] MB, and $\Delta_{js}^{\uparrow}$ is in [1, 4] time slots ([10, 15] time slots for cloud). In addition, we select a sigmoid function as the utility function, *i.e.*, $f_j(t) = \frac{100\kappa}{1+e^{0.02t}}$ [58], where κ is a latency-sensitive factor and its value is in [1, 5]. Then, we simulate an edge-cloud network running for $T = 150$ time slots (a time slot is one hour long), with $S = [50, 300]$ servers and one cloud. Each edge server contains [1, 5] workers and [1, 3] PSs, where the types of workers and PSs are set to be 5, respectively. Following the similar settings in [5], [6], [10]. The bandwidth of a type-w worker (b_w) and that of type-p PS (b_p) are set within [100, 5*1024] Mbps and [5*1024, 20*1024] Mbps.

Baselines. We compare DPS with two representative scheduling schemes:

- Dominant Resource Fairness Scheduling (DRF): default scheduler in YARN [23] and Mesos [24]. All jobs are accepted and scheduled according to their arrival time. Let workers be the dominant resources. For job j, we first calculate the maximal share of workers for job j as r_j (according to constraint (3). r_j means that D_j workers are allocated to job j). Then we compare r_j with historical maximal share r_m of workers. We choose the smaller one as the share of job j and derive the number of workers from it. The number of PSs are set according to constraint (12). The type of worker (PS) with minimal k_j^{wp} are chosen.
- Spark FIFO: default scheduler in Hadoop [26] and Spark [27]. All jobs are accepted and scheduled by the order of arrival, and the number of workers needed is specified by the job owners. In FIFO, the number of workers is fixed to D_j. The other settings are similar with DRF.

6.2 Performance Results

6.2.1 *Testbed Results.*

We first demonstrate the empirical convergence of DPS by training three different models with DPS and then observing their accuracy. The results are presented in Fig. 5, from which we can conclude that any model with DPS can converge regardless of its type.

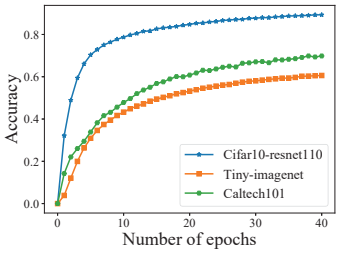

Fig. 5: Accuracy of three ML models.

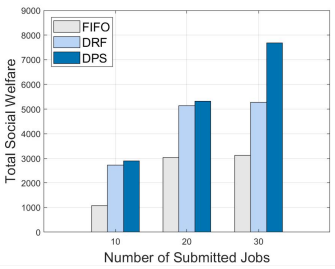

Fig. 6: Total social welfare.

Fig. 7: The number of completed jobs.

In our testbed experiments, the setting of utility functions is the same as our simulator, *i.e.*, $f_j(t) = \frac{100\kappa}{1+e^{0.02t}}$. We plot the social welfare of three algorithms under different J in Fig. 6. As observed from Fig. 6, we are confident that our algorithm DPS outperforms two baselines. The gap between DPS and two baselines becomes larger with the growth of the number of submitted jobs. In the next sub-section, we will discuss the performance of DPS in the case of a large-scale inputs though a simulator. Furthermore, Fig. 7

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

shows the corresponding number of completed jobs in our experiments. Compared with FIFO and DRF, we can see that DPS handles the most jobs. Especially, DRF completes as many jobs as DPS when submitting 10 or 20 jobs, which is why the results of DRF and DPS is similar. From Fig. 7, we can infer that the upper bound of the number of completed jobs for FIFO and DRF is close to 11 and 22, respectively. Therefore, it illustrates that DPS can effectively avoid resource starvation, and completes more jobs given the same resources.

6.2.2 Simulation Results.

Social Welfare. Fig. 8 and Fig. 9 present the social welfare achieved by different schemes under different number of jobs and servers, respectively. In both cases, DPS performs better than the other scheduling algorithms, respectively. In Fig. 8, the social welfare grows with the increase of the number of jobs since more jobs are accepted. Our algorithms performs up to 200% better than others. Since with the fixed number of servers ($S = 100$), resource usage has been saturated when J is larger than 150, so the objective value hardly increases. In Fig. 9, the social welfare grows with the increase of the number of servers as resources are richer.

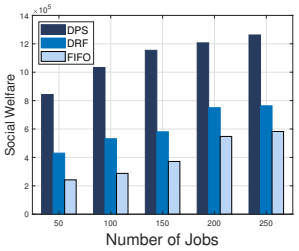

Fig. 8: Social welfare w.r.t. sigmoid utility function with different number of jobs.

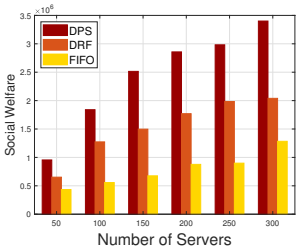

Fig. 9: Social welfare w.r.t. sigmoid utility function with different number of servers.

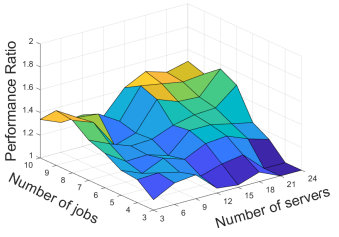

Fig. 10: Performance ratio of DPS.

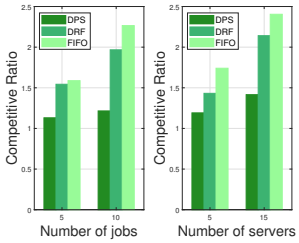

Fig. 11: Performance ratio under different number of jobs and servers.

Performance Ratio. Fig. 10 shows the performance ratio of DPS under different numbers of jobs and servers. The performance ratio is defined as the ratio of the social welfare computed by DPS to that returned by the offline optimal algorithm. We computed the performance ratio by dividing the social welfare of the offline optimal solution by that of DPS. Due to the high time complexity of finding the optimal offline solution, we set $T = 10, W = 1, P = 1$ and vary the workload of jobs. However, it still takes more than 30 hours to compute the optimal offline solution with 10 jobs. The performance ratio of DPS ranges from 1 to 1.6, showing the good performance of DPS. Fig. 11 compares the performance ratio of DPS with that of other algorithms. DPS performs better than baselines and is not affected greatly by the change of S and J.

Utility Function. To test the sensitivity of jobs' utility function, we adopt three types of utility functions as follows.

- Linear Function: $f_j^1(t) = -bt + aT, a, b > 0$,
  - $a \in [10, 20], b \in [1, 5]$.
- Inverse Function: $f_j^2(t) = aE_jD_jM_j * (T - t)/t$,
  - $a \in [1, 5]$.
- Sigmoid Function (default): $f_j^3(t) = \frac{100\kappa}{1+e^{0.02t}}$,
  - $\kappa \in [1, 5]$.

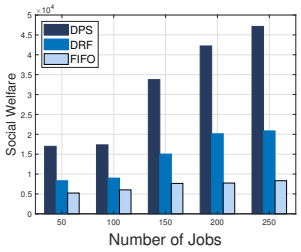

Fig. 12: Social welfare w.r.t. linear utility function under different number of jobs.

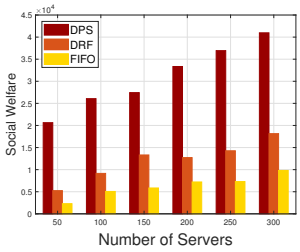

Fig. 13: Social welfare w.r.t. linear utility function under different number of servers.

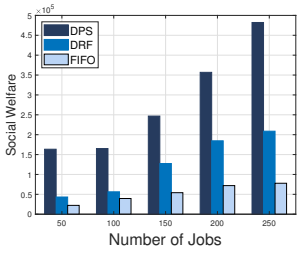

Fig. 14: Social welfare w.r.t. inverse utility function under different number of jobs.

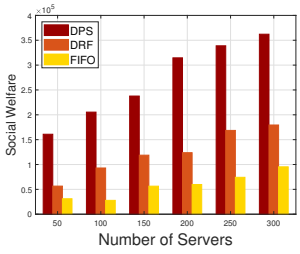

Fig. 15: Social welfare w.r.t. linear inverse function under different number of servers.

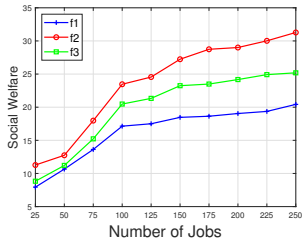

Fig. 16: Social welfare of DPS with different utility functions.

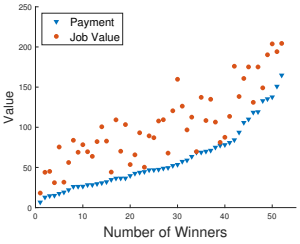

Fig. 17: Job value versus payment of winning bid.

Fig. 12 - Fig. 15 measure the algorithm performance w.r.t. linear and inverse utility functions for DPS and the baseline algorithms under different settings. Combining the results in Fig. 8 and Fig. 9, this experiment shows that DPS always performs the best under different utility functions. Besides, for different utility functions, the improvement of DPS compared with other baseline algorithms is roughly the same, indicating that DPS is less insensitive to utility functions compared to the baselines.

Fig. 16 shows the social welfare of DPS under different utility functions. Because there is a difference in the range of these functions, we normalize the result by dividing the *social welfare* by the maximum value of each price function.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

For each price function, the social welfare grows with the increase of the number of jobs. Among all the functions, f^2 performs best. Due to the resource capacity limitation of the servers, the social welfare hardly increases when J is larger than 150. As for f^1 and f^3, it is hard for them to accept jobs with long duration since they only consider the latency. However, f^2 considers both latency and the workload of jobs, which enables DPS to accept cost-efficient jobs, even the job takes a long time to finish. So the social welfare of f^2 keeps increasing when J is larger.

Individual Rationality. Fig. 17 compares the payment and the value of accepted jobs selected by DPS. We can see that the value of job is always larger than its corresponding payment. Therefore each accepted job's payoff is non-negative and the property of individual rationality can be satisfied.

Running Time. We measure the actual running time under different settings and plot the results in Fig. 18. The results show that the average time to compute a schedule for a job is less than 2 seconds. Compared to the long time to train an ML job, such decision time is acceptable.

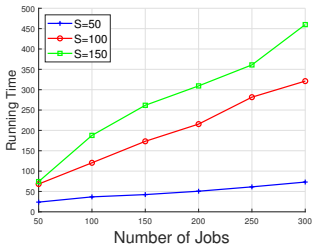

Fig. 18: Running Time of DPS under different settings.

7 Conclusion

This paper proposes an online algorithm DPS to price and deploy ML jobs on the edge-cloud network. As far as we know, this work is the first to study the dynamic pricing and deployment of ML jobs in the edge-cloud scenario. We first model the social welfare maximization problem, and then reformulate the problem to a classical packing-type integer program. When each job arrives online, DPS calculates the fee that the job owner needs to pay, decides whether to accept the job, and deploys the accepted ML job. DPS prices the workers and PSs based on historical job information, and then calculates the most profitable placement plan. We conduct theoretical analysis on the performance of DPS, and prove that DPS can achieve a close-to-optimal social welfare in polynomial time. Finally, the large-scale simulations based on real world data and testbed experiments also verify the excellent performance of DPS compared with two state-of-the-art algorithms.

References

[1] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing grouped aggregation in geo-distributed streaming analytics," in *Proc. of ACM HPDC*, 2015, pp. 133–144.

[2] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2020.

[3] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 3, pp. 515–529, 2020.

[4] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge Intelligence: paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[5] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. of IEEE INFOCOM*, 2018, pp. 495–503.

[6] Q. Zhang, R. Zhou, C. Wu, L. Jiao, and Z. Li, "Online scheduling of heterogeneous distributed machine learning jobs," in *Proc. of ACM Mobihoc*, 2020, pp. 111–120.

[7] 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, pp. 583–598.

[8] M. M. Amiri and D. Gündüz, "Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air," in *Proc. of IEEE ISIT*, 2019, pp. 2155–2169.

[9] H. Hu, D. Wang, and C. Wu, "Distributed machine learning through heterogeneous edge systems," in *Proc. of AAAI*, 2020, pp. 7179–7186.

[10] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. of USENIX OSDI*, 2014, pp. 571–582.

[11] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *Proc. of ACM SIGKDD*, 2015, pp. 1355–1364.

[12] X. Zhang, R. Zhou, J. C. Lui, and Z. Li, "Dynamic pricing and placement for distributed machine learning jobs," in *Proc. of BigCom*, 2020, pp. 152–160.

[13] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc. of IEEE INFOCOM*, 2017, pp. 1–9.

[14] *Amazon EC2 Pricing*, 2021, http://www.paddlepaddle.org.

[15] *Linux Virtual Machines Pricing*, 2021, https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/.

[16] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *Proc. of USENIX OSDI*, 2020, pp. 481–498.

[17] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *Proc. of ACM EuroSys*, 2018, p. 1–14.

[18] Z. Xiong, S. Feng, W. Wang, D. Niyato, P. Wang, and Z. Han, "Cloud/fog computing resource management and pricing for blockchain networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4585–4600, 2018.

[19] L. Lu, J. Yu, Y. Zhu, and M. Li, "A double auction mechanism to bridge users' task requirements and providers' resources in two-sided cloud markets," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 4, pp. 720–733, 2018.

[20] L. Pu, L. Jiao, X. Chen, L. Wang, Q. Xie, and J. Xu, "Online resource allocation, content placement and request routing for cost-efficient edge caching in cloud radio access networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1751–1767, 2018.

[21] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," in *Proc. of LearningSys*, 2016.

[22] *Kubernetes*, 2021, https://kubernetes.io/.

[23] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proc. of ACM SOCC*, 2013, pp. 1–16.

[24] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proc. of USENIX NSDI*, 2011, p. 295–308.

[25] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. of USENIX OSDI*, 2011, p. 323–336.

[26] *Hadoop: Fair Scheduler*, 2021, https://hadoop.apache.org/docs/r2.7.3/hadoop-yarn/hadoop-yarn-site/FairScheduler.html.

[27] *Job Scheduling*, 2021, http://spark.apache.org/docs/latest/job-scheduling.html.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

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

[29] L. Chen, Y. Feng, B. Li, and B. Li, "A case for pricing bandwidth: Sharing datacenter networks with cost dominant fairness," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1256–1269, 2021.

[30] J. Sadiku and T. O. Omotehinwa, "A dynamic strategy-proof algorithm for allocation and pricing of cloud services," *International Journal of Cloud Computing*, vol. 8, p. 166, 2019.

[31] Z. Wang, J. Wu, Y. Wu, S. Deng, and H. Huang, "QoS aware dynamic pricing and scheduling in wireless cloud computing," in *Proc. of IEEE CDC*, 2017, pp. 3702–3707.

[32] X. Zhang, C. Wu, Z. Huang, and Z. Li, "Occupation-oblivious pricing of cloud jobs via online learning," in *Proc. of IEEE INFOCOM*, 2018, pp. 2456–2464.

[33] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. of IEEE INFOCOM*, 2018, pp. 207–215.

[34] J. Meng, H. Tan, X. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1270–1286, 2020.

[35] C. Zhang, H. Tan, H. Huang, Z. Han, S. H.-C. Jiang, N. Freris, and X.-Y. Li, "Online dispatching and scheduling of jobs with heterogeneous utilities in edge computing," in *Proc. of ACM Mobihoc*, 2020, pp. 101–110.

[36] P. Sun, Y. Wen, T. N. B. Duong, and S. Yan, "Towards distributed machine learning in shared clusters: A dynamically-partitioned approach," in *Proc. of IEEE SMARTCOMP*, 2017, pp. 1–6.

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

[38] M. M. Amiri and D. Gunduz, "Computation scheduling for distributed machine learning with straggling workers," *IEEE Transactions on Signal Processing*, vol. 67, no. 24, pp. 6270–6284, 2019.

[39] W. Xiao, J. Xue, Y. Miao, Z. Li, C. Chen, M. Wu, W. Li, and L. Zhou, "Distributed graph computation meets machine learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1588–1604, 2020.

[40] 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 Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 948–966, 2020.

[41] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A gpu cluster manager for distributed deep learning," in *Proc. of USENIX NSDI*, 2019, pp. 485–500.

[42] N. Nisan, T. Roughgarden, Éva Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[43] Q. Ho, J. Cipar, H. Cui, J. K. Kim, S. Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," *Advances in Neural Information Processing Systems*, pp. 1223–1231, 2013.

[44] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. of VLDB Endow.*, vol. 5, no. 8, p. 716–727, 2012.

[45] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: A survey," *Surveys in Operations Research and Management Science*, vol. 17, no. 2, pp. 97 – 106, 2012.

[46] R. Zhou, Z. Li, C. Wu, and Z. Huang, "An efficient cloud market mechanism for computing jobs with soft deadlines," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 793–805, 2017.

[47] *HDFS*, 2021, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[48] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A {GPU} cluster manager for distributed deep learning," in *Proc. of USENIX NSDI*, 2019.

[49] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *Proc. of EuroSys*, 2018, pp. 1–14.

[50] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. of IEEE CVPR*, 2017.

[51] *The CIFAR-10 Dataset*, 2009, https://www.cs.toronto.edu/~kriz/cifar.html.

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. of IEEE CVPR*, 2015, pp. 1–9.

[53] *Caltech101 Dataset*, 2006, http://www.vision.caltech.edu/Image_Datasets/Caltech101/.

[54] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[55] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, p. 84–90, 2017.

[56] *ImageNet Dataset*, 2017, http://www.image-net.org.

[57] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. of IEEE CVPR*, 2016, pp. 2818–2826.

[58] Z. Huang, B. Balasubramanian, M. Wang, T. Lan, M. Chiang, and D. H. K. Tsang, "Need for speed: CORA scheduler for optimizing completion-times in the cloud," in *Proc. of IEEE INFOCOM*, 2015, pp. 891–899.

Ruiting Zhou received the PhD degree from the Department of Computer Science, University of Calgary, Canada, in 2018. She has been an associate professor with the School of Cyber Science and Engineering, Wuhan University since June 2018. Her research interests include cloud computing, machine learning and mobile network optimization. She has published research papers in top-tier computer science conferences and journals, including IEEE INFOCOM, ACM MobiHoc, ICDCS, IEEE/ACM Transactions on Networking, IEEE Journal on Selected Areas in Communications, IEEE Transactions on Mobile Computing. She also serves as a reviewer for journals and international conferences such as the IEEE Journal on Selected Areas in Communications, IEEE Transactions on Mobile Computing, IEEE Transactions on Cloud Computing, IEEE Transactions on Wireless Communications, and IEEE/ACM IWQOS.

Ne Wang received her B.E. and M.S. degrees in 2016 and 2019 from School of Computer Science and Technology, Wuhan University Of Technology, China. Since September, 2019, she has been a Ph.D student in School of Computer Science, Wuhan University, China. Her research interests are in the areas of edge computing, distributed machine learning optimization, and online scheduling.

Yifeng Huang is currently pursuing his Bachelor degree at School of Computer Science, Wuhan University, China. His research interests include edge computing, algorithm optimization, and online scheduling.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3195765

Jinlong Pang received a B.E. degree in School of Power and Machinery and a second B.E. degree in School of Computer both from Wuhan University, China. Now he is pursuing his M.E. degree in School of Cyber Science and Engineering at Wuhan University. His research interests includes distributed machine learning, federated learning, online learning and algorithm optimization.

Hao Chen received his Ph. D. degree in electronic science and engineering from Southeast University in 2012. Currently, he is a senior engineer at Huawei Nanjing Research Institute, Nanjing, China. His research interests include storage network, and resource management in cloud computing environment.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 29,2023 at 13:13:47 UTC from IEEE Xplore. Restrictions apply.