Toward Efficient Online Scheduling for Distributed Machine Learning Systems

Menglu Yu, *Graduate Student Member, IEEE*, Jia Liu, *Senior Member, IEEE*, Chuan Wu, *Senior Member, IEEE*, Bo Ji, *Senior Member, IEEE*, and Elizabeth S. Bentley, *Member, IEEE*

***Abstract*—Recent years have witnessed a rapid growth of distributed machine learning (ML) frameworks, which exploit the massive parallelism of computing clusters to expedite ML training. However, the proliferation of distributed ML frameworks also introduces many unique technical challenges in computing system design and optimization. In a networked computing cluster that supports a large number of training jobs, a key question is how to design efficient scheduling algorithms to allocate workers and parameter servers across different machines to minimize the overall training time. Toward this end, in this paper, we develop an online scheduling algorithm that jointly optimizes resource allocation and locality decisions. Our main contributions are three-fold: i) We develop a new analytical model that considers both resource allocation and locality; ii) Based on an equivalent reformulation and observations on the worker-parameter server locality configurations, we transform the problem into a mixed packing and covering integer program, which enables approximation algorithm design; iii) We propose a meticulously designed approximation algorithm based on randomized rounding and rigorously analyze its performance. Collectively, our results contribute to the state of the art of distributed ML system optimization and algorithm design.**

***Index Terms*—Online resource scheduling, distributed machine learning, approximation algorithm.**

I. Introduction

FUELED by the rapid growth of data analytics and machine learning (ML) applications, recent years have witnessed an ever-increasing hunger for computing power. However, with hardware capability no longer advancing at the pace of the Moore's law, it has been widely recognized that a viable solution to sustain such computing power needs is to exploit *parallelism* at both machine and chip scales. Indeed, the recent success of deep neural networks (DNN) is enabled by the use of distributed ML frameworks, which exploit the massive parallelism over computing clusters with a large number of GPUs. These distributed ML frameworks have significantly accelerated the training of DNN for many applications (e.g., image and voice recognition, natural language processing, etc.). To date, prevailing distributed ML frameworks include TensorFlow [1], MXNet [2], PyTorch [3], Caffe [4], to name just a few.

However, the proliferation of distributed ML frameworks also introduces many unique technical challenges on large-scale computing system design and network resource optimization. Particularly, due to the decentralized nature, at the heart of distributed learning system optimization lies the problem of scheduling ML jobs and resource provisioning across different machines to minimize the total training time. Such scheduling problems involve dynamic and combinatorial worker and parameter server allocations, which are inherently NP-hard. Also, the allocations of workers and parameter servers should take *locality* into careful consideration, since co-located workers and parameter servers can avoid costly network communication overhead. However, locality optimization adds yet another layer of difficulty in scheduling algorithm design. Exacerbating the problem is the fact that the future arrival times of training jobs at an ML computing cluster are hard to predict, which necessitates *online* algorithm design without the knowledge of future job arrivals. So far in the literature, there remains a lack of holistic theoretical studies that address all the aforementioned challenges. Most of the existing scheduling schemes are based on simple heuristics without performance guarantee (see Section II for detailed discussions). This motivates us to fill this gap and pursue efficient online scheduling designs for distributed ML resource optimization, which offer *provable* performance guarantee.

The main contribution of this paper is that we develop an online scheduling algorithmic framework that *jointly* yields resource scheduling and locality optimization decisions with strong competitive ratio performance. Further, we reveal interesting insights on how distributed ML frameworks affect online resource scheduling optimization. Our main technical results are summarized as follows:

- By abstracting the architectures of prevailing distributed ML frameworks, we formulate an online resource scheduling optimization problem that: i) models the training of ML jobs based on the parameter server (PS) architecture and stochastic gradient descent (SGD)

Manuscript received November 1, 2020; revised June 25, 2021; accepted July 26, 2021. Date of publication August 13, 2021; date of current version June 27, 2022. This work has been supported in part by NSF grants CA- REER CNS-2110259, CNS-2102233, CCF-2110252, ECCS-2140277, CNS- 2112694, HKU-17204619, HKU-17208920, and a Google Faculty Research Award. Recommended for acceptance by Dr. Jiangchuan Liu. *(Corresponding author: Jia Liu.)*

Menglu Yu is with the Department of Computer Science, Iowa State University, Ames, IA 50011 USA (e-mail: mengluy@iastate.edu).

Jia Liu is with the Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH 43210 USA (e-mail: liu@ece.osu.edu).

Bo Ji is with the Department of Computer Science, Virginia Tech, Blacksburg, VA 24061 USA (e-mail: boji@vt.edu).

Chuan Wu is with the Department of Computer Science, University of Hong Kong, Hong Kong 999077, Hong Kong (e-mail: cwu@cs.hku.hk).

Elizabeth S. Bentley is with the Air Force Research Laboratory, Information Directorate, Rome, NY 13441 USA (e-mail: elizabeth.bentley.3@us.af.mil).

Digital Object Identifier 10.1109/TNSE.2021.3104513

method; and ii) explicitly takes *locality* optimization into consideration. We show that, due to the heterogeneous internal (between virtual machines or containers) and external (between physical machines) communications, the locality-aware scheduling problem contains *non-deterministic* constraints and is far more complex compared to the existing works that are locality-oblivious (see, e.g., [5], [6]).

- To solve the locality-aware scheduling problem, we develop an equivalent problem reformulation to enable subsequent developments of online approximation algorithms. Specifically, upon carefully examining the locality configurations of worker-server relationships, we are able to transform the original problem to a special-structured integer nonlinear program with mixed cover/packing-type constraints, and the low-complexity approximation algorithm design with provable performance can be further entailed.
- To tackle the integer nonlinear problem with mixed cover/packing-type constraints, we propose an approximation algorithm based on a meticulously designed randomized rounding scheme and then rigorously prove its performance. We note that the results of our randomized rounding scheme are general and could be of independent theoretical interest. Finally, by putting all algorithmic designs together, we construct a primal-dual online resource scheduling (PD-ORS) scheme, which has an overall competitive ratio that only *logarithmically* depends on ML job characteristics (e.g., required epochs, training samples).

Collectively, our results contribute to a comprehensive and fundamental understanding of distributed machine learning system optimization. The remainder of this paper is organized as follows. In Section II, we review the literature to put our work in comparative perspectives. Section III introduces the system model and problem formulation. Section IV presents our algorithms and their performance analysis. Section V presents numerical results and Section VI concludes this paper.

II. Related Work

As mentioned in Section I, due to the high computational workload of ML applications, many distributed ML frameworks (e.g., TensorFlow [1], MXNet [2], PyTorch [3], Caffe [4]) have been proposed to leverage modern large-scale computing clusters. A common distributed training architecture implemented in these distributed ML frameworks is the PS architecture [7], [8], which employs multiple workers and PSs (implemented as virtual machines or containers) to collectively train a global ML model. Coupled with the *iterative* ML training based on stochastic gradient descent (SGD), the interactions between machines in the distributed ML cluster are significantly different from those in traditional cloud computing platforms (e.g., MapReduce [9] and Dryad [10] and references therein). For example, a MapReduce job usually partitions the input data into independent chunks, which are then processed by the *map* step in a parallel fashion. The output of the maps are then fed to the *reduce* step to be aggregated to yield the final result. Clearly, the data flows in MapReduce are a "one-way traffic," which is unlike those iterative data flows in ML training jobs whose completions highly depend on the ML job's convergence property. As a result, existing job scheduling algorithms for cloud systems are not suitable for distributed ML frameworks.

Among distributed ML system studies, most of the early attempts (see, e.g., [7], [8] and references therein) only considered static allocation of workers and parameter servers. To our knowledge, the first work on understanding the performance of distributed ML frameworks is [11], where Yan *et al.* developed analytical models to quantify the impacts of model-data partitioning and system provisioning for DNN. Subsequently, Chun *et al.* [5] developed heuristic dynamic system reconfiguration algorithms to allocate workers and parameter servers to minimize the runtime, but without providing optimality guarantee. The first dynamic distributed scheduling algorithm with optimality guarantee was reported in [12], where Sun *et al.* used standard mixed integer linear program (MILP) solver to dynamically compute the worker-parameter server partition solutions. Due to NP-hardness of the MILP, the scalability of this approach is limited. The most recent work [13] proposed an online scheduling algorithm to schedule synchronous training jobs in ML clusters with the goal to minimize the weighted completion time. However, the consecutive time slots were allocated for each training job, and the numbers of workers and parameter servers could not be adjusted.

Another line of the research is to leverage the learning-based approach to do the resource scheduling. There are a number of recent works using deep reinforcement learning (DRL) for resource allocation, device placement, and video streaming. For example, Mao *et al.* [14] and Chen *et al.* [15] designed a multi-resource cluster scheduler using DRL with the goal to minimize average job slowdown. The proposed scheduler picks one or more of the waiting jobs in the queue and allocate to machines at each time slot, and the resource demand of each job is unknown until after its arrival. Later, Mao *et al.* [16], [17] used DRL to heuristically train scheduling policies for graph-based parallel jobs by setting both parallelism level and execution order. Meanwhile, Mirhoseini *et al.* [18], [19] utilized DRL to design a model for efficient placement of computational graphs onto hardware devices, aiming at minimize the running time of each individual TensorFlow job. Although various performance gains have been empirically reported, these DRL-based studies do not offer optimality performance guarantee due to the lack of theoretical foundation of DRL as of today.

The most relevant work to ours is [6], where Bao *et al.* developed an online primal-dual approximation algorithm, OASiS, to solve the scheduling problem for distributed ML systems. Our work differs from [6] in the following key aspects: 1) In [6], the workers and parameter servers are allocated on two *strictly separated* sets of physical machines, i.e., *no* worker and parameter server can share the same physical

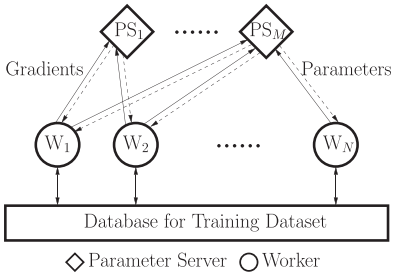


Fig. 1. Illustration of distributed training with the PS architecture.

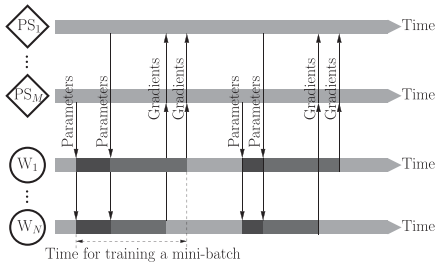


Fig. 2. The workflow of iterative training.

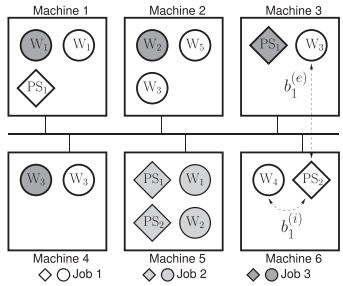


Fig. 3. Colocated parameter servers and workers on physical machines.

machine, which significantly simplified the underlying optimization problem. In this work, we consider the cases that workers and parameter servers can be *co-located* on the same physical machine, which is the common practice in existing ML systems (see, e.g., [20], [21]). Such co-location can significantly reduce inter-server communication, expedite training, and improve resource utilization efficiency between workers and parameter servers. However, as will be shown later, the co-location setting leads to an integer *non-convex* optimization problem with *non-deterministic* constraints, which is much harder and necessitates new algorithm design. 2) Ref. [6] advocates dynamic worker number adjustment, but does *not* guarantee the same global batch size across the training iterations. According to recent literature [22], maintaining a consistent global batch size is important for ensuring convergence of DNN training, when the worker number varies. We ensure a consistent global batch size in our model. We note that the co-location setting was considered in [23]. However, the scheduling algorithm therein is a heuristic and does not provide performance guarantee. This motivates us to develop new algorithms with provable performance to fill this gap.

III. System Model and Problem Formulation

In this section, we first provide a quick overview on the architecture of distributed ML frameworks to familiarize readers with the necessary background. Then, we will introduce our analytical models for ML jobs and resource constraints, as well as the overall problem formulation.

1) Distributed Machine Learning: A Primer. As illustrated in Fig. 1, the key components of a PS-based distributed ML system include parameter servers, workers, and the training dataset, which are usually implemented over a connected computing cluster that contains multiple physical machines. The training dataset of an ML job is stored in distributed data storage (e.g., HDFS [24]) and usually divided into equal-sized data chunks. Each data chunk further contains multiple equal-sized mini-batches.

To date, one of the most widely adopted training algorithms in distributed ML frameworks is the *stochastic gradient descent method* (SGD) [25]. With SGD, the interactions between workers and parameter servers are illustrated in Fig 2. A worker is loaded with the DNN model (we focus on data parallel training) with current values of the model parameters (e.g., the weights of a DNN) and retrieves a new data chunk from the data storage. In each training iteration, a worker processes one mini-batch from its data chunk to compute gradients (i.e., directions and magnitudes of parameter changes).[1] Upon finishing a mini-batch, the worker sends the gradients to the parameter servers, receives updated global parameters, and then continues with the next training iteration to work on the next mini-batch. On the parameter server side, parameters are updated as: $\mathbf{w}[k] = \mathbf{w}[k-1] + \alpha_k \mathbf{g}[k]$, where $\mathbf{w}[k]$, α_k, and $\mathbf{g}[k]$ denote the parameter values, step-size, and stochastic gradient in the k-th update, respectively.

2) Modeling of Learning Jobs: We consider a time-slotted system. The scheduling time-horizon is denoted as $\mathcal{T}$ with $|\mathcal{T}| = T$. We use $\mathcal{I}$ to represent the set of training jobs and let a_i denote the arrival time-slot of job $i \in \mathcal{I}$. As shown in Fig. 3, parameter servers and workers could spread over multiple physical machines. We let $\mathcal{H}$ represent the set of physical machines. For each job i, we use $w_{ih}[t], s_{ih}[t] \geq 0$ to represent the allocated numbers of workers and parameter servers on machine $h \in \mathcal{H}$ in each time-slot $t \geq a_i$, respectively. Further, we let $\mathcal{P}_i[t] \triangleq \{h \in \mathcal{H} | s_{ih}[t] > 0\}$ and $\mathcal{W}_i[t] \triangleq \{h \in \mathcal{H} | w_{ih}[t] > 0\}$ denote the sets of physical machines that contain parameter servers and workers for job i in time-slot t, respectively.

We use a binary variable $x_i \in \{0, 1\}$ to indicate whether job i is admitted ($x_i = 1$) or not ($x_i = 0$). We use τ_i to denote the training for each sample of job i. We let $b_i(h, p)$ denote the data rate of the link between a worker for job i (on machine h) and a parameter server (on machine p). Each worker or parameter server is exclusively assigned to some job i, and $b_i(h, p)$ is reserved and decided by the user upon job submission, which is common to ensure the data transfer performance [6]. Note that the value of $b_i(h, p)$ is *locality-dependent* where the slowest worker will become the bottleneck since we focus on

[1] As an example, in a DNN model, gradients can be computed by the well-known "back-propagation" approach.

Bulk-Synchronous-Parallel (BSP) scheme [26]. Specifically, we have:

$$b_i(h,p) = \begin{cases} b_i^{(i)}, & \text{if } h = p, \\ b_i^{(e)}, & \text{otherwise,} \end{cases}$$

where $b_i^{(i)}$ and $b_i^{(e)}$ denote the internal and external communication link rates, respectively. For example, as shown in Fig. 3, since Job 1's worker W_4 and parameter server PS_2 are both on the same machine, they communicate at the internal link rate $b_1^{(i)}$. On the other hand, since Job 1's worker W_3 and parameter server PS_2 are on different physical machines, they communicate at the external link rate $b_1^{(e)}$. In practice, it usually holds that $b_i^{(e)} \ll b_i^{(i)}$.

Next, we calculate the amount of time for a worker on machine h to process a sample. We use F_i to denote the global batch size of job i, which is a fixed constant across all time-slots.[2] We assume F_i is equally divided among workers, i.e., the local batch size at each worker is: $F_i / \sum_{h' \in \mathcal{H}} w_{ih'}[t]$.[3]

We assume symmetric link speed in both directions between a worker and a PS. Let g_i denote the size of gradients/parameters of job i. Then, from a worker's perspective, to push gradients to and pull updated parameters from the PSs for job i, the combined uplink/downlink communication time can be computed as: $(2g_i / \sum_{h' \in \mathcal{H}} s_{ih'}[t]) / (\min_{p \in \mathcal{P}_i[t]} b_i(h,p))$, where the numerator term $g_i / \sum_{h' \in \mathcal{H}} s_{ih'}[t]$ follows from the assumption of even parameter distribution among the PSs, and the denominator is due to the fact that push/pull time is decided by the slowest link among all connections from the worker to all PSs (i.e., $\min_{p \in P_i[t]} b_i(h,p)$). Hence, the average computation and communication time to process a sample on machine $h \in \mathcal{W}_i[t]$ for job i in time slot t can be computed as:

$$\underbrace{\tau_i}_{\substack{\text{Training time} \\ \text{per sample}}} + \underbrace{\left(\frac{2g_i / \sum_{h' \in \mathcal{H}} s_{ih'}[t]}{\min_{p \in \mathcal{P}_i[t]} b_i(h,p)}\right) \Big/ \left(\frac{F_i}{\sum_{h' \in \mathcal{H}} w_{ih'}[t]}\right)}_{\text{Communication time per sample}}.$$

Recall that we focus on the BSP scheme, where all workers are synchronized before they proceed to the next iteration. In other words, the total number of samples trained on machine $h \in \mathcal{W}_i[t]$ for job i in time slot t is determined by the slowest link among all connections from all workers to all PS (i.e., $\min_{p \in \mathcal{P}_i[t], h' \in \mathcal{W}_i[t]} b_i(h,' p)$). It then follows that the *number of samples trained* on machine $h \in \mathcal{W}_i[t]$ for job i in time-slot t can be computed as:

$$\frac{w_{ih}[t]}{\tau_i + \left(\frac{2g_i / \sum_{h' \in \mathcal{H}} s_{ih'}[t]}{\min_{p \in \mathcal{P}_i[t], h' \in \mathcal{W}_i[t]} b_i(h,' p)}\right) \Big/ \left(\frac{F_i}{\sum_{h' \in \mathcal{H}} w_{ih'}[t]}\right)}. \quad (1)$$

Note that in practice, ML users usually specify a fixed ratio between worker number and PS number (e.g., often 1:1[4]) when launching their training jobs to ensure appropriate coordinations between workers and PSs in terms of channel bandwidth, memory allocation, etc. To model this practice, we define the ratio of worker number to PS number for each job i as:

$$\gamma_i \triangleq \frac{\sum_{h' \in \mathcal{H}} w_{ih'}[t]}{\sum_{h' \in \mathcal{H}} s_{ih'}[t]}, \quad \forall i, t. \quad (2)$$

With γ_i, we can rewrite (1) as:

$$\frac{w_{ih}[t]}{\tau_i + \frac{\gamma_i}{F_i} \frac{2g_i}{\min_{p \in \mathcal{P}_i[t], h' \in \mathcal{W}_i[t]} b_i(h,' p)}}.$$

Suppose that, for job i, there are K_i data samples in its training dataset. In practice, $K_i \gg F_i$. In ML systems, an epoch is defined as a round of training that exhausts all data samples. We let E_i denote the number of epochs needed by job i. In this paper, we assume that the epoch of each job is predetermined. This is because it is often difficult to estimate the required number of epochs for SGD-type methods' convergence. Therefore, most SGD-type algorithms in practice stop after a fixed number of iterations (i.e., fixed number of epochs, see, e.g., [30] and references therein) to avoid excessive training delay.

Then, the total number of samples to be processed for job i over the entire training process is $E_i K_i$. To make sure that there are sufficient workers allocated for job i over the entire training horizon, we have:

$$\sum_{t \in \mathcal{T}} \sum_{h \in \mathcal{H}} \frac{w_{ih}[t]}{\tau_i + \frac{\gamma_i}{F_i} \frac{2g_i}{\min_{p \in \mathcal{P}_i[t], h' \in \mathcal{W}_i[t]} b_i(h,' p)}} \geq x_i E_i K_i, \forall i \in \mathcal{I}. \quad (3)$$

We note that, with co-located workers and parameter servers on each machine, (3) is *non-deterministic* due to the existence of the $\min\{\cdot\}$ operator. As will be shown later, this non-deterministic constraint makes the scheduling design far more complicated than related works [5]–[8].

To model the fact that the largest number of assigned concurrent workers is no more than the global batch size (otherwise, some workers will be idle), we have:

$$\sum_{h \in \mathcal{H}} w_{ih}[t] \leq x_i F_i, \quad \forall i \in \mathcal{I}, a_i \leq t \leq T. \quad (4)$$

3) Resource Constraint Modeling: We let $\mathcal{R}$ denote the set of resources (e.g., CPU/GPU, memory, storage, etc.). Let α_i^r

[2] We note that this fixed global batch size requirement is *compliant* with the standard SGD implementation [27] and important for ensuring convergence [22]. In contrast, the global batch size in some existing works on dynamic ML resource allocation (e.g., [6]) could be time-varying, which necessitates time-varying dynamic learning rate adjustments to offset correspondingly and further complicates the SGD implementation.

[3] Most distributed ML frameworks (e.g., Tensorflow [28]) set the same local batch size to each worker for the distributed training.

[4] In practice, the ratio between numbers and parameter servers are specified by the user upon the job's submission (e.g., 1:1 in Kubernetes [29]).

TABLE I
NOTATION

$\mathcal{I}/\mathcal{T}$	The set of jobs/System timespan
$\tilde{t}_i/a_i$	Completion time of job i /Arrival time of job i
$u_i(\cdot)/K_i$	Job $i's$ utility function/Number of samples in i
$\mathcal{R}/\mathcal{H}$	The set of resource types/The set of machines
E_i/F_i	# of training epochs/Global batch size for job i
x_i	Admission decision variable to accept job i or not
C_h^r	Capacity of type-r resource on server h
α_i^r	Type-r resource required by a worker in job i
β_i^r	Type-r resource required by a PS in job i
$w_{ih}[t]$	Number of workers of job i on server h in t
$s_{ih}[t]$	Number of PSs of job i on server h in t
$b_i(h,p)$	Bandwidth consumed by a worker of job i, where $b_i(h,p)=b_i^{(e)}$, if $h\neq p$ or $b_i^{(i)}$, otherwise.
τ_i	Time to train a sample for job i
g_i	Size of gradients and parameters for job i
$\mathcal{W}_i[t]$	Set of physical machines containing workers for job i in t
$\mathcal{P}_i[t]$	Machines containing parameter servers for job i in t
x_{π_i}	Binary decision variable to select schedule π for job i or not
$\tilde{t}_{\pi_i}$	The completion time slot of job i with schedule π
$w_{ht}^{\pi_i}$	# of workers on server h in t for job i in schedule π
$s_{ht}^{\pi_i}$	# of PSs on server h for schedule π in t
Π_i	Set of all feasible schedules for job i
γ_i	The ratio of worker number to PS number for job i
$\rho_h^r[t]$	Allocated type-r resource on machine h in time t
$Q_h^r(\cdot)$	Price function for type-r resource on machine h

and β_i^r be the amount of type-r resource required by a worker and a parameter server for job i, respectively. Let C_h^r be the capacity of type-r resource on machine h. To ensure the resources do not exceed type-r's limit, we have:

$$\sum_{i\in\mathcal{I}}(\alpha_i^r w_{ih}[t]+\beta_i^r s_{ih}[t])\leq C_h^r, \forall t\in\mathcal{T}, r\in\mathcal{R}, h\in\mathcal{H}. \quad (5)$$

Note that for job i, its completion time $\tilde{t}_i$ corresponds to the latest time-slot where there remain some active workers allocated for it. Therefore, we have:

$$\tilde{t}_i=\arg\max_{t\in\mathcal{T}}\left\{\sum_{h\in\mathcal{H}} w_{ih}[t]\ >\ 0\right\}, \quad \forall i\in\mathcal{I}. \quad (6)$$

To ensure that no workers and parameter servers are allocated before job i's arrival, we have:

$$w_{ih}[t]=s_{ih}[t]=0, \quad \forall i\in\mathcal{I}, h\in\mathcal{H}, t\ <\ a_i. \quad (7)$$

4) Objective Function and Problem Statement: Let $u_i(\tilde{t}_i-a_i)$ be the utility function for job i, which is non-increasing with respect to the training time $\tilde{t}_i-a_i$. The utility functions could play the role of various performance metrics based on job completion times (e.g., fairness). In this paper, our goal is to maximize the overall utility for all jobs. Putting all constraints and the objective function together, the offline (with knowledge of $a_i, \forall i$) distributed ML resource scheduling problem (DMLRS) can be formulated as:

$$\textbf{DMLRS}: \underset{\mathbf{x},\mathbf{w},\mathbf{s}}{\text{Maximize}} \sum_{i\in\mathcal{I}} x_i u_i(\tilde{t}_i-a_i)$$
$$\text{subject to Constraints}(3)-(7).$$

Problem DMLRS is an integer nonlinear program, which is NP-hard in general [31]. Also, Problem DMLRS involves two *non-deterministic* constraints in (3) and (6), which are not amenable for conventional optimization techniques. Moreover, the arrivals $\{a_i, \forall i\}$ are often unknown in practice, which necessitates *online optimization*. Overcoming these challenges constitutes the rest of this paper. To conclude this section, we summarize the key notation used in this paper in Table I for easy reference.

IV. Online Scheduling Algorithm Design

In this section, we structure the key components of our online scheduling algorithm design for solving Problem DMLRS into three steps from Sections IV-A to IV-C. We state our main theoretical performance results in Section IV-D.

A. Reformulation for Non-Deterministic Constraint (6)

The first challenge in solving Problem DMLRS stems from the non-deterministic "argmax" structure in constraint (6). To address this challenge, we let Π_i be the set of all feasible schedules for job $i\in\mathcal{I}$ that satisfy constraints (3), (4). Each schedule $\pi_i\in\Pi_i$ is defined by the numbers of workers $w_{ht}^{\pi_i}$ and parameter servers $s_{ht}^{\pi_i}$ allocated for job i on machine h in each time-slot t, i.e., $\pi_i\triangleq\{w_{ht}^{\pi_i}, s_{ht}^{\pi_i}, \forall t\in\mathcal{T}, h\in\mathcal{H}\}$. Note that $w_{ht}^{\pi_i}$ and $s_{ht}^{\pi_i}$ are constants, not to be confused with decision variables $w_{ih}[t]$ and $s_{ih}[t]$. We define a binary variable $x_{\pi_i}\in\{0,1\}$ that is equal to 1 if job i is admitted and scheduled under schedule π_i, or 0, otherwise. Clearly, due to the combinatorial nature, $|\Pi_i|$ is exponential. We let $\tilde{t}_{\pi_i}$ denote job i's completion time under schedule π_i. Then, one can equivalently reformulate Problem DMLRS as:

$$\textbf{R}-\textbf{DMLRS}:$$
$$\underset{\mathbf{x}}{\text{Maximize}} \sum_{i\in\mathcal{I}}\sum_{\pi_i\in\Pi_i} x_{\pi_i}u_i(\tilde{t}_{\pi_i}-a_i)$$
$$\text{subject to} \sum_{i\in\mathcal{I}}\sum_{\pi_i\in\Gamma(t,h)}(\alpha_i^r w_{ht}^{\pi_i}+\beta_i^r s_{ht}^{\pi_i})x_{\pi_i}\leq C_h^r, \quad \forall t\in\mathcal{T}, r\in\mathcal{R}, h\in\mathcal{H}, \quad (8)$$
$$\sum_{\pi_i\in\Pi_i} x_{\pi_i}\leq 1, \quad \forall i\in\mathcal{I},$$
$$x_{\pi_i}\in\{0,1\}, \quad \forall i\in\mathcal{I}, \pi_i\in\Pi_i, \quad (9)$$

where we use $\Gamma(t,h)$ to represent the set of feasible schedules that use machine h to deploy workers or parameter servers in time-slot t. Constraint (8) guarantees that, in any time-slot t and on any machine h, the total amount of consumed type-r resources will not exceed the capacity limit C_h^r. Constraint (9) ensures that, for each job i, at most one feasible schedule from Π_i will be selected. Note that Problem R-DMLRS is an integer linear program (ILP) and a feasible solution to Problem R-

Algorithm 1: Primal-Dual Online Resource Scheduling (PD-ORS).

Initialization:

1. Let $w_{ih}[t]=0$, $s_{ih}[t]=0$, $\forall i,t,h$. Let $\rho_h^r[t]=0$, $\forall h,r,t$. Choose some appropriate initial values for $p_h^r[0]$.

Main Loop:

2. Upon the arrival of job i, determine a schedule π_i^* to maximize the RHS of (11) and its corresponding payoff λ_i using **Algorithm 2** (*to be specified*).
3. If $\lambda_i > 0$, set $x_i = 1$. Set $w_{ih}[t]$ and $s_{ih}[t]$ according to schedule π_i^*, $\forall t \in \mathcal{T}(\pi_i^*)$, $h \in \mathcal{H}(\pi_i^*[t])$.
   Update $\rho_h^r[t] \leftarrow \rho_h^r[t] + \alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t]$, $\forall t \in \mathcal{T}(\pi_i^*)$, $h \in \mathcal{H}(\pi_i^*[t])$, $r \in \mathcal{R}$.
   Update $p_h^r[t] = Q_h^r(\rho_h^r[t])$, $\forall t \in \mathcal{T}(\pi_i^*)$, $h \in \mathcal{H}(\pi_i^*[t])$, $r \in \mathcal{R}$.
   Schedule job i based on π_i^* and go to Step 2.
4. If $\lambda_i \le 0$, set $x_i = 0$ and reject job i and go to Step 2.

DMLRS has a corresponding feasible solution to the original Problem DMLRS, and vice versa. Notice that the non-deterministic constraint (6) *no longer exists* in Problem R-DMLRS. Further, if relaxing binary x_{π_i}-variables to real-valued, Problem R-DMLRS is a linear program (LP). However, it remains difficult to solve Problem R-DMLRS since it has an *exponential* number of x_{π_i}-variables due to the combinatorial nature of feasible schedules. We will address this challenge in the next subsection.

B. An Online Primal-Dual Framework for R-DMLRS

In what follows, we adopt a primal-dual online algorithmic framework to reduce the number of binary variables, which is an effective approach to address this kind of challenge in the literature (see, e.g., [6], [32]). Note that, in the dual of Problem R-DMLRS, the number of dual variables is polynomial. Meanwhile, although there are an exponential number of constraints in the dual problem, one only needs to be concerned with the set of active (binding) constraints, which are easier to deal with. To see this, we associate dual variables $p_h^r[t] \ge 0$, $\forall t \in \mathcal{T}$, $h \in \mathcal{H}$, $r \in \mathcal{R}$ and $\lambda_i > 0$, $i \in \mathcal{I}$, with (8) and (9), respectively. Then, following the standard procedure of dualization (relaxing the integrality constraints), we obtain the following dual problem:

$$\mathbf{D-R-DMLRS:}$$
$$\underset{\boldsymbol{\lambda},\mathbf{p}}{\text{Minimize}} \sum_{i\in\mathcal{I}}\lambda_i + \sum_{t\in\mathcal{T}}\sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}} p_h^r[t]C_h^r \tag{10}$$

$$\begin{aligned}
&\text{subject to } \lambda_i \ge u_i(\tilde{t}_{\pi_i} - a_i) - \sum_{t\in\mathcal{T}(\pi_i)}\sum_{h\in\mathcal{H}(\pi_i[t])}\sum_{r\in\mathcal{R}}(\alpha_i^r w_{ht}^{\pi_i} \\
&+ \beta_i^r s_{ht}^{\pi_i})p_h^r[t], \quad \forall i\in\mathcal{I}, \pi_i\in\Pi_i, \\
&p_h^r[t] \ge 0, \quad \forall t\in\mathcal{T}, h\in\mathcal{H}, r\in\mathcal{R}, \\
&\lambda_i \ge 0, \quad \forall i\in\mathcal{I},
\end{aligned} \tag{11}$$

where $\mathcal{T}(\pi_i)$ denotes the time-slots utilized by schedule π_i and $\mathcal{H}(\pi_i[t])$ denotes the set of machines containing workers and/or parameter servers under π_i in time-slot t. Here, $p_h^r[t]$ can be viewed as the price for type-r resource in time t. Then, the right-hand-side (RHS) of (11) can be interpreted as job utility minus overall resource cost of job i using schedule π_i. Thus $\lambda_i \ge 0$ can be viewed as the payoff of admitting job i with π_i. Next, we examine the properties of Problem D-R-DMLRS. To minimize (10), we tend to reduce λ_i and $p_h^r[t]$ until they hit zero. However, as λ_i and $p_h^r[t]$ decrease, the left-hand-side (LHS) and RHS of (11) decreases and increases, respectively (the term $u_i(\tilde{t}_{\pi_i} - a_i)$ in the RHS of (11) is a constant given π_i). Therefore, λ_i will drop to a value λ_i^*, which is equal to maximum of the RHS of (11) achieved by some schedule π_i^* and dual price $p_h^{r*}[t]$, i.e.,

$$\lambda_i^* = u_i(\tilde{t}_{\pi_i^*} - a_i) - \sum_{t\in\mathcal{T}(\pi_i^*)}\sum_{h\in\mathcal{H}(\pi_i^*[t])}\sum_{r\in\mathcal{R}}(\alpha_i^r w_{ht}^{\pi_i^*} + \beta_i^r s_{ht}^{\pi_i^*})p_h^{r*}[t].$$

This optimality structural insight implies that Problem D-R-DMLRS is equivalent to finding an optimal schedule π_i^* and dual price $p_h^{r*}[t]$ to maximize the RHS of (11). The above insights motivate the following primal-dual-based algorithm as shown in Algorithm 1.

The intuition of Algorithm 1 is as follows: By the complementary slackness condition of the Karush-Kuhn-Tucker (KKT) conditions [27], the primal constraint (9) must be tight when dual variable $\lambda_i > 0$, which implies that $x_i = 1$ (Step 3) in Problem DMLRS. Otherwise, if $\lambda_i = 0$, then the RHS of (11) is non-positive, meaning the utility is low compared to the cost of resource consumption under schedule π_i^*. Therefore, we should reject job i ($x_i = 0$ in Step 4). However, in order for the PD-ORS algorithm to work, two challenging components need to be specified: the schedule π_i^* and how to update the cost function $Q_h^r(\cdot)$[5] In what follows, we will first focus on designing $Q_h^r(\cdot)$ and defer the challenging problem of finding π_i^* to Section IV-C. For the design of $Q_h^r(\cdot)$, consider the following choice of $Q_h^r(\cdot)$:

$$Q_h^r(\rho_h^r[t]) = L(U^r/L)^{\frac{\rho_h^r[t]}{C_h^r}}, \tag{12}$$

where constants U^r, $\forall r$, and L are defined as follows:

$$U^r \triangleq \max_{i\in\mathcal{I}} \frac{u_i(\lceil \frac{E_i K_i}{F_i}(\tau_i + 2g_i\gamma_i/(b_i^{(i)}F_i))\rceil - a_i)}{\alpha_i^r + \beta_i^r}, \forall r\in\mathcal{R}, \tag{13}$$

$$L \triangleq \min_{i\in\mathcal{I}} \frac{1/(2\mu)u_i(T-a_i)}{\sum_{r\in\mathcal{R}}\lceil E_i K_i(\tau_i + 2g_i\gamma_i/(b_i^{(e)}F_i)\rceil(\alpha_i^r + \beta_i^r)}. \tag{14}$$

[5] Here, we note that the "cost function" $Q_h^r(\cdot)$ is interpreted from servers' perspective rather than jobs' perspective. Specifically, higher cost means servers allocated more resources to jobs, which implies higher utility for the jobs since jobs receive more resources from servers.

The scaling factor μ in the definition of L satisfies as follows:

$$\frac{1}{\mu} \leq \frac{\lceil E_i K_i(\tau_i + 2g_i\gamma_i/(b_i^{(e)}F_i))\rceil \sum_{r\in\mathcal{R}}(\alpha_i^r + \beta_i^r)}{T\sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}} C_h^r}, \forall i \in \mathcal{I}.$$

Here, U^r is the maximum unit-resource job utility to deploy workers and parameter servers with type-r resource. Here, $u_i(\lceil \frac{E_i K_i}{F_i}(\tau_i + 2g_i\gamma_i/(b_i^{(i)}F_i))\rceil - a_i)$ is the largest utility job i can achieve by using the maximum number of *co-located* workers and parameter servers (hence communicating rate is $b_i^{(i)}$) at all times during all E_i epochs, so that $\lceil \frac{E_i K_i}{F_i}(\tau_i + 2g_i\gamma_i/(b_i^{(i)}F_i))\rceil - a_i$ is the earliest possible job completion time. Similarly, L represents the minimum unit-time unit-resource job utility among all jobs, with $u_i(T - a_i)$ being the smallest utility for job i, and workers and parameter servers communicate at small external rate $b_i^{(e)}$. We use $\rho_h^r[t]$ to denote the allocated amount of type-r resource to machine h for (future) time slot t. The intuition behind the $Q_h^r(\cdot)$ function is as follows: i) At $t=0$, $\rho_h^r[0]=0, \forall h \in \mathcal{H}, r \in \mathcal{R}$. Hence, the price $p_h^r[0] = L$ is the lowest, $\forall h, r$, and any job can be admitted; ii) As allocated resources increases, the price increases exponentially fast to reject early coming jobs with low utility and to reserve resources for later arrived jobs with higher utility; iii) When some type-r resource is exhausted, i.e., $\rho_h^r[t] = C_h^r, \exists r \in \mathcal{R}$, $Q_h^r[C_h^r] = U^r$ and no job that requires type-r resources will be admitted since the U^r is the highest price. As will be shown later, this price function leads to a logarithmically scaling competitive ratio in online algorithm design. Note that computing the price function in Algorithm 1 requires the information of constants U^r, L, which can usually be estimated empirically based historical data.

Here, we point out a few interesting insights on the design choices of the cost function in (12). Note that U^r and L are defined in Eqns (13) and (14), respectively. Here, we intentionally choose U^r to be dependent on r and L to be independent on L due to the following reasons:

First, the rationality of choosing an upper bound U^r that varies with different resource types is to ensure that when some type-r resource is exhausted, no more jobs that require type-r resource should be allocated. In other words, when the allocated amount of type-r resource reaches the capacity of physical machine h, i.e., $\rho_h^r[t] = C_n^r, \exists r \in \mathcal{R}$, the price $p_h^r = Q_h^r(C_h^r)$ should reach the upper bound U^r, indicating that any job that requires type-r resource will not be allocated to h. However, jobs that do not require type-r resource should still be able to be scheduled on machine h. For example, jobs that do not require GPU can still be placed on a machine with no available GPUs.

Second, the reason that we choose the lower bound L to be independent of any resource type r is to yield a larger ratio of $\frac{U^r}{L}$. The larger the ratio of $\frac{U^r}{L}$ is, the greater the price will be. Intuitively, the ratio $\frac{U^r}{L}$ can be interpreted as the scheduling "uncertainty," which increases as the ratio gets larger, implying the price function reacts to the accumulative resource consumption more aggressively. Thus, choosing L to be independent of resource type r allows the price function $Q_h^r(\cdot)$ to react more aggressively to the accumulative allocated resource amount. We note that one can also choose the lower bound to be dependent on resource type r by replacing L with L^r. By doing so, the log-scaling theoretical competitive ratio in Theorem 5 still holds and the proof in Appendix XII only needs to be slightly updated with the new notation L^r. However, the empirical performance of using L^r as lower bound is worse since the price function reacts less aggressively to the accumulative allocated resources.

Algorithm 2: Determine π_i^* in Step 2 of Algorithm 1.

Initialization:

1. Let $\tilde{t}_i = a_i$. Let $\lambda_i = 0$, $\pi_i^* = \varnothing$, $w_{ih}[t] = s_{ih}[t] = 0, \forall i, t, h$.

Main Loop:

2. Compute $\Theta(\tilde{t}_i, V_i)$ in (21) using **Algorithm 3**. Denote the resulted schedule as π_i. Let $\lambda_i' = u_i(\tilde{t}_i - a_i) - \Theta(\tilde{t}_i, V_i)$. If $\lambda_i' > \lambda_i$, let $\lambda_i \leftarrow \lambda_i'$ and $\pi_i^* \leftarrow \pi_i$.
3. Let $\tilde{t}_i \leftarrow \tilde{t}_i + 1$. If $\tilde{t}_i > T$, stop; otherwise, go to Step 2.

C. Determining Schedule π_i^* in Step 2 of Algorithm 1

Now, consider the problem of finding a schedule π_i^* in Step 2 of Algorithm 1 to maximize the RHS of (11). First, we note that *any* schedule for job i has a unique completion time $\tilde{t}_i$, including the maximizer π_i^* for (11). Hence, the problem of finding the maximum RHS of (11) can be decomposed as:

$$\operatorname*{Max}_{\tilde{t}_i} \left\{ \begin{array}{c} \operatorname*{Max}_{\mathbf{w},\mathbf{s}} \; u_i(\tilde{t}_i - a_i) - \sum_{t\in\mathcal{T}}\sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}} p_h^r[t] \\ \times (\alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t]) \\ \text{s.t.} \;\; \alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t] \leq \hat{C}_h^r[t], \\ \forall t \in \mathcal{T}, r \in \mathcal{R}, h \in \mathcal{H}, \\ \text{Constraints (3)(4)(7) for } x_i = 1, \end{array} \right\}, \quad (15)$$

where $\hat{C}_h^r[t] \triangleq C_h^r - \rho_h^r[t]$. Note that in the inner problem, $u_i(\tilde{t}_i - a_i)$ is a constant for any given $\tilde{t}_i$. Thus, the inner problem can be simplified as:

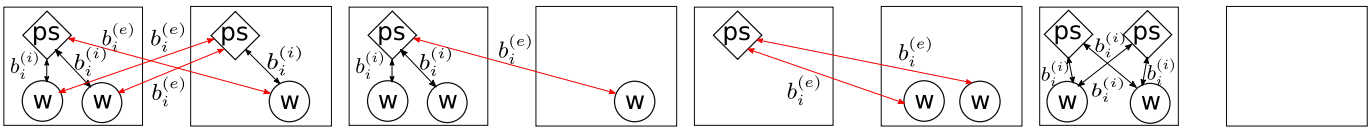
$$\operatorname*{Minimize}_{\mathbf{w},\mathbf{s}} \quad \sum_{t\in[a_i,\tilde{t}_i]}\sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}} p_h^r[t](\alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t]) \quad (16)$$

$$\text{subject to} \quad \sum_{t\in[a_i,\tilde{t}_i]}\sum_{h\in\mathcal{H}} \frac{w_{ih}[t]}{\tau_i + \frac{\gamma_i}{F_i}\frac{2g_i}{\min_{p\in\mathcal{P}_i[t],h'\in\mathcal{W}_i[t]} b_i(h,'p)}} \geq V_i, \quad (17)$$

$$\alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t] \leq \hat{C}_h^r[t], \forall r, h, \forall t \in [a_i, \tilde{t}_i],$$

$$\text{Constraint (4) for all } t \in [a_i, \tilde{t}_i], \quad (18)$$

where $V_i \triangleq E_i K_i$ represents the total training workload (total count of samples trained, where a sample is counted E_i times if trained for E_i times). Note that in Problem (16), the only coupling constraint is (17). This observation inspires a dynamic programming approach to solve Problem (16). Consider the problem below if training workload at time t is known (denoted as $V_i[t]$):



(a) $|\mathcal{P}_i[t]| \neq 1$. (b) $|\mathcal{P}_i[t]| = 1, |\mathcal{W}_i[t]| \neq 1$. (c) $|\mathcal{P}_i[t]|=|\mathcal{W}_i[t]|=1, \mathcal{P}_i[t]\neq\mathcal{W}_i[t]$. (d) $|\mathcal{P}_i[t]|=|\mathcal{W}_i[t]|=1, \mathcal{P}_i[t]=\mathcal{W}_i[t]$.

Fig. 4. Values of $\frac{2g_i}{\min_{p\in\mathcal{P}_i[t],h\in\mathcal{W}_i[t]} b_i(h,p)}$ under various settings of $\mathcal{P}_i[t]$ and $\mathcal{W}_i[t]$. Here, $\frac{2g_i}{\min_{p\in\mathcal{P}_i[t],h\in\mathcal{W}_i[t]} b_i(h,p)} = 2\frac{g_i}{b_i^{(i)}}$ if and only if in (d).

Algorithm 3: Solving $\Theta(\tilde{t}_i, V_i)$ by Dynamic Programming.

Initialization:

1. Let *cost-min* = ∞, $\pi_i = \varnothing$, and $v = 0$.

Main Loop:

2. Compute $\theta(\tilde{t}_i, v)$ using **Algorithm 4** (*to be specified*). Denote the resulted cost and schedule as *cost-v* and $\hat{\pi}_i$.

3. Compute $\Theta(\tilde{t}_i - 1, V_i - v)$ by calling **Algorithm 3** itself. Denote the resulted cost and schedule as *cost-rest* and $\tilde{\pi}_i$.

4. If cost-min $>$ cost-v + cost-rest then cost-min = cost-v + cost-rest and let $\pi_i \leftarrow \hat{\pi}_i \cup \tilde{\pi}_i$.

5. Let $v \leftarrow v + 1$. If $v > V_i$ stop; otherwise go to Step 2.

$$\underset{w_{ih}[t],s_{ih}[t],\forall h}{\text{Minimize}} \quad \sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}} p_h^r[t](\alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t]) \tag{19}$$

$$\text{subject to} \quad \sum_{h\in\mathcal{H}} \frac{w_{ih}[t]}{\tau_i + \frac{\gamma_i}{F_i}\frac{2g_i}{\min_{p\in\mathcal{P}_i[t],h'\in\mathcal{W}_i[t]} b_i(h',p)}} \geq V_i[t],$$

$$\text{Constraints (4)(18) for the given } t. \tag{20}$$

Let $\Theta(\tilde{t}_i, V_i)$ and $\theta(t, V_i[t])$ denote the optimal values of Problems (16) and (19), respectively. Then, Problem (16) is equivalent to the following dynamic program:

$$\Theta(\tilde{t}_i, V_i) = \min_{v\in[0,V_i]} \big\{\theta(\tilde{t}_i, v) + \Theta(\tilde{t}_i - 1, V_i - v)\big\}. \tag{21}$$

We find the optimal workload v to be completed in time slot $\tilde{t}_i$ by enumerating it from 0 to E_iK_i, and the remaining workload $E_iK_i - v$ will be carried out in time span $[a_i, \tilde{t}_i - 1]$. The optimal workload would be the schedule with minimum costs, i.e., the objective function of Problem (19) is minimum. Then we proceed to the next time slot $\tilde{t}_i - 1$ with the workload $E_iK_i - v$, which is the same as finding the optimal schedule and cost as the last time slot $\tilde{t}_i$ except in a smaller scale. Then, by enumerating all $\tilde{t}_i \in [a_i, T]$ and solving the dynamic program $\Theta(\tilde{t}_i, V_i)$ in (21) for every choice of $\tilde{t}_i$, we can solve Problem (15) and determine the optimal schedule π_i^*. We summarize this procedure in Algorithms 2 and 3:

In Algorithm 3, however, how to compute $\theta(t, v)$ in Step 2 (i.e., Problem (19)) is yet to be specified. A challenge in solving (19) is the *non-deterministic* constraint in (20), where $b_i(h,p)$ can be either $b_i^{(i)}$ or $b_i^{(e)}$. Therefore, we need to handle both cases. To this end, we observe the following basic fact about $\frac{2g_i}{\min_{p\in\mathcal{P}_i[t],h\in\mathcal{W}_i[t]} b_i(h,p)}$ (also illustrated in Fig. 4) as stated in Fact 1. We omit the proof of this fact due to its simplicity, which is illustrated in Fig. 4.

Fact 1: The function $(2g_i/\min_{p\in\mathcal{P}_i[t],h\in\mathcal{W}_i[t]} b_i(h,p)) = 2g_i/b_i^{(i)}$ if and only if $|\mathcal{P}_i[t]| = |\mathcal{W}_i[t]| = 1$ and $\mathcal{P}_i[t] = \mathcal{W}_i[t]$; otherwise, $(2g_i/\min_{p\in\mathcal{P}_i[t],h\in\mathcal{W}_i[t]} b_i(h,p)) = 2g_i/b_i^{(e)}$.

With Fact 1, we now consider the following two cases:

Case 1): $b_i^{(i)}$ (Internal Communication): In Case 1), Fact 1 implies that Problem (19) reduces to a *single-machine* problem (i.e., discarding $\sum_{h\in\mathcal{H}}\{\cdot\}$ in (19) and (20)). Note further that if we temporarily ignore the workload-coupling constraint (20) and use the worker-PS ratio in (2), Problem (19) can be *decoupled* across resources and simplified as:

$$\sum_{r\in\mathcal{R}} \left\{ \begin{array}{ll} [l] \text{ Min} & p_h^r[t]s_{ih}[t](\alpha_i^r\gamma_i + \beta_i^r) \\ \text{s.t.} & s_{ih}[t](\alpha_i^r\gamma_i + \beta_i^r) \leq \hat{C}_h^r[t], \\ & \text{Constraint (4) for given } r, h, \end{array} \right. \tag{22}$$

where each summand in (22) is an integer linear program (ILP) having a trivial solution $w_{ih}[t] = s_{ih}[t] = 0$, $\forall h \in \mathcal{H}$. However, $w_{ih}[t] = 0$, $\forall h \in \mathcal{H}$, clearly violates the workload constraint (20). Thus, when (22) is optimal, there should be *exactly one* machine $h' \in \mathcal{H}$ with $w_{ih'}[t] \geq 1$ and *exactly one* machine $h'' \in \mathcal{H}$ with $s_{ih''}[t] \geq 1$. This observation shows that the optimal solution of (19) *tends to favor* $|\mathcal{P}_i[t]| = |\mathcal{W}_i[t]| = 1$ if the workload constraint (20) is not binding.

Notice that the workload constraint (20) in Case 1) becomes $\gamma_i s_{ih}[t] \geq V_i[t](\tau_i + \frac{2g_i\gamma_i}{b_i^{(i)}F_i})$. This implies the following simple solution: We can first sort each physical machine h according to $\sum_{r\in\mathcal{R}} p_h^r[t](\alpha_i^r\gamma_i + \beta_i^r)$ and calculate the minimum number of $s_{ih}[t] = V_i[t](\tau_i + \frac{2g_i\gamma_i}{b_i^{(i)}F_i})/\gamma_i$ from the workload constraint. The last step is to check if the machine satisfy the resource capacity constraint (18) and constraint (4). If so, we return the schedule $(w_{ih}[t], s_{ih}[t])$ and the corresponding cost value.

Case 2): $b_i^{(e)}$ (External Communication): For those settings that do not satisfy $|\mathcal{P}_i[t]| = |\mathcal{W}_i[t]| = 1$ and $\mathcal{P}_i[t] = \mathcal{W}_i[t]$, Fact 1 indicates that parameter servers and workers are communicating at external rate $b_i^{(e)}$. In this case, the workload constraint (20) simply becomes: $\sum_{h\in\mathcal{H}} w_{ih}[t] \geq V_i[t](\tau_i + \frac{2g_i\gamma_i}{b_i^{(e)}F_i})$. Then, we can rewrite Problem (19) as:

$$\underset{w_{ih}[t],s_{ih}[t],\forall h}{\text{Minimize}} \quad \sum_{h\in\mathcal{H}} p_h^w[t]w_{ih}[t] + p_h^s[t]s_{ih}[t] \tag{23}$$

$$\text{subject to} \quad \alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t] \leq \hat{C}_h^r[t], \ \forall h, r, \tag{24}$$

$$\sum\nolimits_{h\in\mathcal{H}} w_{ih}[t] \leq F_i, \tag{25}$$

$$\sum\nolimits_{h\in\mathcal{H}} w_{ih}[t] \geq V_i[t]\left(\tau_i + \frac{2g_i\gamma_i}{b_i^{(e)}F_i}\right), \tag{26}$$

Algorithm 4: Solving $\theta(t, v)$ (i.e., Problem (19)).

Initialization:

1. Let $w_{ih}[t]=s_{ih}[t]=0, \forall h$. Let $h=1$. Pick some $\delta \in (0,1]$. Let G_δ be defined as in (29) or Eqn (30). Let $D=\lceil v(\tau_i + 2g_i\gamma_i/(b_i^{(i)}F_i))\rceil$. Let $h^*=\varnothing$. Let *cost-min*$=\infty$. Choose some integer $S \geq 1$. Let $iter \leftarrow 1$.

Handling Internal Communication:

2. Sort machines in $\mathcal{H}$ according to $\sum_{r\in\mathcal{R}} p_h^r[t](\alpha_i^r\gamma_i+\beta_i^r)$ in non-decreasing order into $h_1, h_2, \ldots, h_H$.
3. Calculate the minimum number of $s_{ih}[t]=V_i[t](\tau_i+\frac{2g_i\gamma_i}{b_i^{(i)}F_i})/\gamma_i$.
4. If Constraint (4) is not satisfied, go to Step 7.
5. If Constraint (24) is not satisfied, go to Step 7.
6. Return *cost-min*$\sum_{r\in\mathcal{R}} p_h^r[t]s_{ih}[t](\alpha_i^r\gamma_i+\beta_i^r)$ and $h^*=h$.
7. Let $h \leftarrow h+1$. If $h > H$, stop; otherwise, go to Step 2.

Handling External Communication:

8. Solve the linear programming relaxation of Problem (23). Let $\{\bar{w}_{ih}[t], \bar{s}_{ih}[t], \forall h, t\}$ be the fractional optimal solution.
9. Let $w'_{ih}[t]=G_\delta\bar{w}_{ih}[t], s'_{ih}[t]=G_\delta\bar{s}_{ih}[t], \forall h, t$.
10. Generate an integer solution $\{w_{ih}[t], s_{ih}[t], \forall h, t\}$ following the randomized rounding scheme in (27)–(28).
11. If $\{w_{ih}[t], s_{ih}[t], \forall h, t\}$ is infeasible or $iter < S$, then $iter \leftarrow iter+1$, go to Step 10.

Final Step:

12. Compare the solutions between internal and external cases. Pick the one with the lowest cost among them and return the cost and the corresponding schedule $\{w_{ih}[t], s_{ih}[t], \forall h, t\}$.

where $p_h^w[t] \triangleq \sum_{r\in\mathcal{R}} p_h^r[t]\alpha_i^r$ and $p_h^s[t] \triangleq \sum_{r\in\mathcal{R}} p_h^r[t]\beta_i^r$ denote the aggregated prices of all resources of allocating workers and PSs on machine h in time t, respectively.

Unfortunately, Problem (23) is a highly challenging integer programming problem with generalized packing and cover type constraints (i.e., integer variables rather than 0-1 variables) in (24)–(26), respectively, which is clearly NP-Hard. Also, it is well-known that there are *no* polynomial time approximation schemes (PTAS) even for the basic set-cover and bin-packing problems unless P = NP [31]. In what follows, we will pursue an instance-dependent constant ratio approximation scheme to solve Problem (23) in this paper. To this end, we propose a randomized rounding scheme: First, we solve the linear programming relaxation of Problem (23). Let $\{\bar{w}_{ih}[t], \bar{s}_{ih}[t], \forall h, t\}$ be the fractional optimal solution. We let $\delta \in (0,1]$ be a parameter. Let G_δ be a constant (the notation G_δ signifies that G_δ is dependent on δ) to be defined later, and let $w'_{ih}[t]=G_\delta\bar{w}_{ih}[t], s'_{ih}[t]=G_\delta\bar{s}_{ih}[t], \forall h, t$. Then, we randomly round $\{w'_{ih}[t], s'_{ih}[t], \forall h, t\}$ to obtain an integer solution as follows:

$$w_{ih}[t]=\begin{cases}\lceil w'_{ih}[t]\rceil, \text{ with probability } w'_{ih}[t]-\lfloor w'_{ih}[t]\rfloor,\\ \lfloor w'_{ih}[t]\rfloor, \text{ with probability } \lceil w'_{ih}[t]\rceil - w'_{ih}[t],\end{cases} \tag{27}$$

$$s_{ih}[t]=\begin{cases}\lceil s'_{ih}[t]\rceil, \text{ with probability } s'_{ih}[t]-\lfloor s'_{ih}[t]\rfloor,\\ \lfloor s'_{ih}[t]\rfloor, \text{ with probability } \lceil s'_{ih}[t]\rceil - s'_{ih}[t].\end{cases} \tag{28}$$

We will later prove in Theorem 3 (when $0 < G_\delta \leq 1$) and Theorem 4 (when $G_\delta > 1$) that the approximation ratio of this randomized rounding scheme in (27)-(28) enjoys a ratio that is independent on the problem size.

Lastly, summarizing results in Cases 1) – 2) yields the following approximation algorithm for solving Problem (19):
In the internal communication part of Algorithm 4, we first sort the machines and then check each machine one by one (Step 2). We calculate the minimum number of $s_{ih}[t]$ needed to satisfy the learning workload demand D (Step 3). If Constraint (4) is satisfied (Step 4), we further check the resource capacity constraint (18) (Step 5). If we detect a machine with all above constraints satisfied, we return the cost and schedule accordingly (Step 6). After exploring one machine, we move on to the next one as long as it is not the last machine (Step 7). The external communication part is based on LP relaxation (Step 8) and randomized rounding (Step 9-12). Note that the randomized rounding will find at most S integer feasible solutions (Step 12). Finally, we choose the lowest cost among the solutions from the internal and external communication parts (Step 13).

D. Theoretical Performance Analysis

We now examine the competitive ratio of our PD-ORS algorithm. Note that the key component in PD-ORS is our proposed randomized rounding scheme in (27)–(28), which is in turn the foundation of Algorithm 1. Thus, we first prove the following results regarding the randomized rounding algorithm. Consider an integer program with generalized cover/packing constraints: $\min\{\mathbf{c}^\top\mathbf{x} : \mathbf{Ax} \geq \mathbf{a}, \mathbf{Bx} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}_+^n\}$, where $\mathbf{A} \in \mathbb{R}_+^{m\times n}$, $\mathbf{B} \in \mathbb{R}_+^{r\times n}$, $\mathbf{a} \in \mathbb{R}_+^m$, $\mathbf{b} \in \mathbb{R}_+^r$, and $\mathbf{c} \in \mathbb{R}_+^n$. Let $\bar{\mathbf{x}}$ be a fractional optimal solution. Consider the randomized rounding scheme: Let $\mathbf{x}'=G_\delta\bar{\mathbf{x}}$ for some G_δ (to be specified). Randomly round $\mathbf{x}'$ to $\hat{\mathbf{x}} \in \mathbb{Z}_+^n$ as: $\hat{x}_j=\lceil x'_j\rceil$ w.p. $x'_j-\lfloor x'_j\rfloor$ and $\hat{x}_j=\lfloor x'_j\rfloor$ o.w. Note that in the rounding process, if $G_\delta > 1$ ($G_\delta \in (0,1]$), the packing (cover) constraint is prone to be violated and the cover (packing) constraint is easier to be satisfied. Hence, depending on which constraint is more preferred to be feasible, we consider two cases with respect to G_δ.

1) $0 < G_\delta \leq 1$: We have the following approximation result (see proof in Appendix VIII):

Lemma 1 (Rounding): Let $W_a \triangleq \min\{a_i/[\mathbf{A}]_{ij} : [\mathbf{A}]_{ij} > 0\}$ and $W_b \triangleq \min\{b_i/[\mathbf{B}]_{ij} : [\mathbf{B}]_{ij} > 0\}$. Let $\delta \in (0,1]$ be a given constant and define G_δ as:

$$G_\delta \triangleq 1+\frac{3\ln(3r/\delta)}{2W_b}-\sqrt{\left(\frac{3\ln(3r/\delta)}{2W_b}\right)^2+\frac{3\ln(3r/\delta)}{W_b}}.$$

Then, with probability greater than $1-\delta$, $\hat{\mathbf{x}}$ achieves a cost at most $\frac{3G_\delta}{\delta}$ times the cost of $\bar{\mathbf{x}}$. Meanwhile, $\hat{\mathbf{x}}$ satisfies $\Pr\{(\mathbf{A}\hat{\mathbf{x}})_i \leq a_i(1-(\frac{2}{G_\delta W_a}\ln(\frac{3}{\delta}m))^{\frac{1}{2}})G_\delta, \exists i\} \leq \frac{\delta}{3}m$.

Remark 1 (Discussions on Feasibility): An insightful remark of Lemma 1 is in order. Note that, theoretically, the expression $1-(\frac{2}{G_\delta W_a}\ln(\frac{3}{\delta}m))^{\frac{1}{2}}$ *in Lemma 1 could become negative. In this case, the last probabilistic inequality in Lemma 1 trivially holds and is not meaningful in characterizing the feasibility, even though the inequality remains valid. In order for the probabilistic*

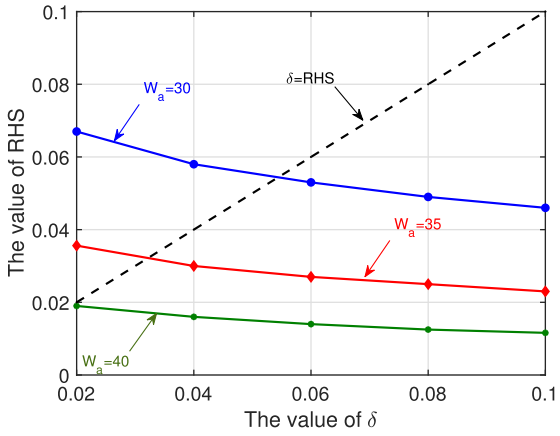


Fig. 5. *The feasibility study.*

statement to be meaningful in characterizing the feasibility of the integer linear program, we solve for δ by enforcing $1-(\frac{2}{G_\delta W_a}\ln(\frac{3\,m}{\delta}))^{\frac{1}{2}} > 0$, which yields $\delta \geq 3m/e^{\frac{G_\delta W_a}{2}}$. On the other hand, we prefer δ to be small since it bounds the feasibility violation probability and approximation ratio achievability in Lemma 1. Hence, the smaller the value of $3\ m/e^{\frac{G_\delta W_a}{2}}$, the less restrictive the condition $\delta \geq 3m/e^{\frac{G_\delta W_a}{2}}$ is.

To gain a deeper understanding on how restrictive the condition $\delta \geq 3\ m/e^{\frac{G_\delta W_a}{2}}$ is, we conduct a case study and the results are illustrated in Fig. 5. Here, we let RHS $\triangleq 3m/e^{\frac{G_\delta W_a}{2}}$ for convenience. We vary δ from 0.02 to 0.1. Clearly, the left-hand-side (LHS) of the condition is the 45° straight line. In order for the condition to hold, the curve of RHS should fall under this straight line. Based on typical computing cluster parameters, we set W_b to 15, and set $r \triangleq RH+1$ to 401 ($R=4, H=100$). We can see from Fig. 5 that as W_a increases, the curve of RHS crosses the dashed line of LHS at a smaller δ-value. This means that, the larger the value of W_a, the easier for $1-(\frac{2}{G_\delta W_a}\ln(\frac{3\,m}{\delta}))^{\frac{1}{2}}$ to become positive. Hence, the probabilistic feasibility characterization in Lemma 1 is useful for typical system parameters in practice.

2) $G_\delta > 1$: We have the following approximation result (see proof in Appendix X):

Lemma 2 (Rounding): Let $W_a \triangleq \min\{a_i/[\mathbf{A}]_{ij} : [\mathbf{A}]_{ij} > 0\}$ and $W_b \triangleq \min\{b_i/[\mathbf{B}]_{ij} : [\mathbf{B}]_{ij} > 0\}$. Let $\delta \in (0,1]$ be a given constant and define G_δ as:

$$G_\delta \triangleq 1+\frac{\ln(3\ m/\delta)}{W_a}+\sqrt{\left(\frac{\ln(3\ m/\delta)}{W_a}\right)^2+\frac{2\ln(3\ m/\delta)}{W_a}}.$$

Then, with probability greater than $1-\delta$, $\hat{\mathbf{x}}$ achieves a cost at most $\frac{3G_\delta}{\delta}$ times the cost of $\bar{\mathbf{x}}$. Meanwhile, $\hat{\mathbf{x}}$ satisfies $\Pr\{(\mathbf{B}\hat{\mathbf{x}})_i > b_i(1+(\frac{3}{G_\delta W_b}\ln(\frac{3r}{\delta}))^{\frac{1}{2}})G_\delta, \exists i\} \leq \frac{\delta}{3r}$.

Several important remarks for Lemmas 1 and 2 are summarized as follows:

i) Note that Alg. 4 is a randomized algorithm. Therefore, its performance is also characterized probabilistically, and δ is used for such probabilistic characterization. Here, it means that with probability $1-\delta$, one achieves an approximation ratio at most $\frac{3G_\delta}{\delta}$ and a probabilistic feasibility guarantee as stated in Lemma 1 when $0 < G_\delta \leq 1$ and Lemma 2 when $G_\delta > 1$. In other words, the statements mean that the probability of getting a better approximation ratio is smaller under randomized rounding (i.e., better approximation ratio $\Rightarrow$ smaller $\frac{3G_\delta}{\delta} \Rightarrow$ larger $\delta \Rightarrow$ smaller probability $1-\delta$). That is, the trade-off between the approximation ratio value and its achieving probability is quantified by δ. A larger δ implies a smaller approximation ratio, but the probability of obtaining a feasible solution of this ratio is also smaller (i.e., more rounds of rounding needed). Interestingly, for $\delta = 1$, Lemmas 1 and 2 indicate that there is still *non-zero* probability to achieve an approximation ratio not exceeding $3G_\delta$.

ii) Note that if we pick $G_\delta \in (0,1]$, the approximation ratio $\frac{3G_\delta}{\delta}$ decreases (the smaller the approximation ratio, the better) as δ increases based on Eqn. (35) in Appendix VIII. However, the growth rate of G_δ is slower compared to that of δ due to the log operator. On the other hand, if we pick $G_\delta > 1$, $\frac{3G_\delta}{\delta}$ decreases as δ increases according to Eqn. (38) in Appendix X. Therefore, the approximation ratio is ultimately controlled by parameter δ. Also, the theoretical approximation ratio $\frac{3G_\delta}{\delta}$ is conservative. Our numerical studies show that the approximation ratio performance in reality is much smaller than $\frac{3G_\delta}{\delta}$.

iii) The probabilistic guarantee of the cover constraint ($\mathbf{Ax} \geq \mathbf{a}$) when $0 < G_\delta \leq 1$ and packing constraint ($\mathbf{Bx} \leq \mathbf{b}$) when $G_\delta > 1$, is unavoidable and due to the fundamental hardness of satisfying both cover and packing constraints, which are of *conflicting* nature: Any strategy trying to better satisfy the packing constraints (multiplying a G_δ-factor with $G_\delta \in (0,1]$) may increase the probability of violating the cover constraints, and the probability of violating the packing constraints may be increased otherwise. However, the probabilistic bound here is for worst case and may be pessimistic.

iv) The results in Lemmas 1 and 2 are in fact applicable for general ILP with mixed cover/packing constraints. Hence, the results and their insights in Lemmas 1 and 2 could be of independent theoretical interest.

By specializing Lemma 1 and Lemma 2 with parameters in Problem (23), we have the following approximation results for Algorithm 4. The first result corresponds to the case where the feasibility of the resource constraint (packing) is more favored, i.e., $0 < G_\delta \leq 1$:

Theorem 3 (Approximation Performance of Alg. 4 When Resource 100.onstraint Feasibility is Favored): Let $W_1 \triangleq V_i[t](\tau_i+\frac{2g_i\gamma_i}{b_i^{(e)}F_i})$, $W_2 \triangleq \min\{F_i, \hat{C}_h^r[t]/\alpha_i^r, \hat{C}_h^r[t]/\beta_i^r, \forall r,h\}$, and $\delta \in (0,1]$. Define G_δ as:

$$G_\delta \triangleq 1+\frac{3\ln(3(RH+1)/\delta)}{2W_2} -\sqrt{\left(\frac{3\ln(3(RH+1)/\delta)}{2W_2}\right)^2+\frac{3\ln(3(RH+1)/\delta)}{W_2}}. \quad (29)$$

Then, with probability greater than $1-\delta$, Algorithm 4 obtains a schedule $\{w_{ih}[t], s_{ih}[t], \forall t, h\}$ that has an approximation ratio at most $\frac{3G_\delta}{\delta}$ with $\Pr\{\text{LHS}(26) \leq W_1(1-(\frac{2}{G_\delta W_1}\ln(\frac{3}{\delta}))^{\frac{1}{2}})G_\delta, \exists i\} \leq \frac{\delta}{3}$.

The next result corresponds to the case where the feasibility of the workload constraint (cover) is more favored, i.e., $G_\delta > 1$:

Theorem 4 (Approximation Performance of Alg. 4 When Workload 100.onstraint Feasibility is Favored): Let $W_1 \triangleq V_i[t](\tau_i + \frac{2g_i\gamma_i}{(e)})$,
$W_2 \triangleq \min\{F_i, \hat{C}_h^r[t]F_i/\alpha_i^r, \hat{C}_h^r[t]/\beta_i^r, \forall r, h\}$, and $\delta \in (0,1]$. Define G_δ as:

$$G_\delta \triangleq 1 + \frac{\ln(3/\delta)}{W_1} + \sqrt{\left(\frac{\ln(3/\delta)}{W_1}\right)^2 + \frac{2\ln(3/\delta)}{W_1}}. \tag{30}$$

Then, with probability greater than $1-\delta$, Algorithm 4 obtains a schedule $\{w_{ih}[t], s_{ih}[t], \forall t, h\}$ that has an approximation ratio at most $\frac{3G_\delta}{\delta}$ with $\Pr\{\text{LHS}(24) > \hat{C}_h^r[t]G_\delta(1+(\frac{3}{G_\delta W_2}\ln(\frac{3(HR+1)}{\delta}))^{\frac{1}{2}})\} \leq \frac{\delta}{3(HR+1)}$.

Note that Eqn. (25) is guaranteed in practice since the number of samples is typically far more than the number of workers. The competitive ratio of our online algorithm is the worst-case upper bound of the ratio of the overall utility of admitted jobs devided by the offline optimal solution of Problem DMLRS to the total utility of admitted jobs achieved by Algorithm 1 in the overall time horizon. Theorems 3 and 4 follow directly from Lemmas 1 and 2, respectively, and we omit the proof here for brevity. Based on these results, we can establish the overall competitive ratio for Algorithm 1 as follows.

Theorem 5 (Competitive Ratio of Alg 1 when $0 < G_\delta \leq 1$): Let δ, G_δ and W_1 be as defined in Theorem 3. Let U^r and L be as defined in (13) and (14), respectively. Then, with probability greater than $(1-(\delta/3)^S)^{TK_iE_i}$, PD-ORS in Algorithm 1 returns a feasible solution that is $\frac{6G_\delta}{\delta}\max_{r\in\mathcal{R}}(1, \ln\frac{U^r}{L})$–competitive.

Theorem 6 (Competitive Ratio of Alg 1 when $G_\delta > 1$): Let δ, G_δ and W_2 be as defined in Theorem 4. Let U^r and L be as defined in (13) and (14), respectively. Then, with probability greater than $(1-(\delta/3(HR+1))^S)^{TK_iE_i}$, PD-ORS in Algorithm 1 returns a feasible solution that is $\frac{6G_\delta}{\delta}\max_{r\in\mathcal{R}}(1, \ln\frac{U^r}{L})$–competitive.

It is worth pointing out that in Theorems 5 and 6, the feasibility achieving probability values, i.e., $(1-(\delta/3)^S)^{TK_iE_i}$ and $(1-(\delta/3(HR+1))^S)^{TK_iE_i}$, can controlled by choosing appropriate values of δ and S (i.e., rounds of rounding) to offset the impact of total number of DP iterations TK_iE_i. The smaller δ and the larger S are, the higher the feasibility achieving probability. Theorems 5 and 6 can be proved by weak duality and the approximation results in Theorems 3 and 4. We provide a proof in Appendix XII.

Theorem 7 (Polynomial Running Time): By combining Algorithms 1–4, the time complexity of PD-ORS is $O(\sum_{i=1}^{|\mathcal{I}|} TK_i^2E_i^2(H^3+S))$, which is polynomial.

Proof: When solving $\theta(t,v)$ using Algorithm 4, it takes $O(H\log H)$ iterations to sort machines in internal communication case under each time slot t and looping all machines to calculate the minimum number $s_{ih}[t]$ takes $O(H)$. Thus, it takes $(H\log H)$ time for the internal communication part in Algorithm 4. For the external communication part in Algorithm 4, solving the LP relaxation of Problem (23) can be approximately bounded $O(H^3)$ if we use a polynomial time LP solver (e.g., Vaidya's algorithm [33]). According to Algorithm 4, the rounding time is proportional to S. Hence, the running time for the external communications part is upper bounded by $O(|\mathcal{H}|^3+S)$. Combining the discussions above, the running time complexity of Algorithm 4 is $O(H\log H + H^3 + S) = O(H^3+S)$. Moreover, the number of states (t,v) is $O(TK_iE_i)$ in the dynamic programming (DP) for each job i, and the time complexity of executing DP is $O(K_iE_i)$. Thus, the time complexity is $O(TK_i^2E_i^2)$ in DP. In Algorithm 1, the number of steps in the main loop is equal to the number of jobs. Therefore, the overall running time complexity can be computed as $O(\sum_{i=1}^{|\mathcal{I}|} TK_i^2E_i^2(H^3+S))$. ■

V. Numerical Results

In this section, we conduct simulation studies to evaluate the efficacy of our proposed PD-ORS algorithm. We test an ML system with jobs parameters generated uniformly at random from the following intervals: $E_i \in [50, 200]$, $K_i \in [20000, 500000]$, $g_i \in [30, 575]$ MB, $\tau_i \in [10^{-5}, 10^{-4}]$ time slots, $\gamma_i \in [1, 10]$, $F_i \in [1, 200]$. We consider four types of resources: GPU, CPU, memory, and storage. For fair comparisons, following similar settings in [7], [8], [12], we set resource demand of each worker as follows: 0–4 GPUs, 1–10 vCPUs, 2–32 GB memory, and 5–10 GB storage. We set resource demand of each parameter server as follows: 1–10 vCPUs, 2–32 GB memory and 5-10 GB storage. The resource capacity of each physical machine is set roughly 18 times of the resource demands of a worker/PS following EC2 C5n instances [34]. We set the job arrival pattern according to the Google Cluster data [35], but with normalized job arrival rates in alternating time-slots as follows: the arrival rates are $1/3$ and $2/3$ in odd and even time-slots, respectively. For fair comparisons, we adopt the Sigmoid utility function [6], [36]: $u_i(t-a_i) = \frac{\theta_1}{1+e^{\theta_2(t-a_i-\theta_3)}}$, where $\theta_1 \in [1, 100]$ indicates the priority of job i, θ_2 indicates how critical the time is for the job i, and $\theta_3 \in [1, 15]$ is the estimated target completion time. We set $\theta_2 = 0$ for time-insensitive jobs (10% of jobs), $\theta_2 \in [0.01, 1]$ for time-sensitive jobs (55% of jobs) and $\theta_2 \in [4, 6]$ for time-critical jobs (35% of jobs).

We first compare our PD-ORS algorithm with three baseline job scheduling policies: (1) *FIFO* in Hadoop and Spark [37], where the jobs are processed in the order of their arrival times. In our setting, the fixed number of workers (parameter servers) is between 1 to 30, (2) *Dominant Resource Fairness Scheduling* (DRF) in Yarn [38] and Mesos [39], where the jobs are scheduled based on their dominant resource share in the cloud to achieve its max-min fairness. The number of workers and parameter servers are computed and allocated

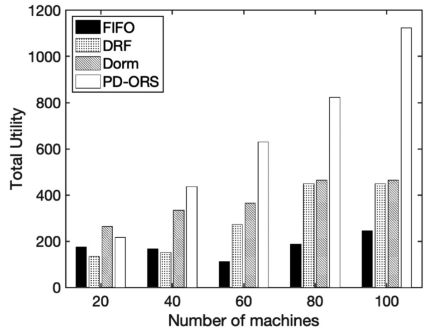


Fig. 6. Total utility with increasing number of machines (synthetic data).

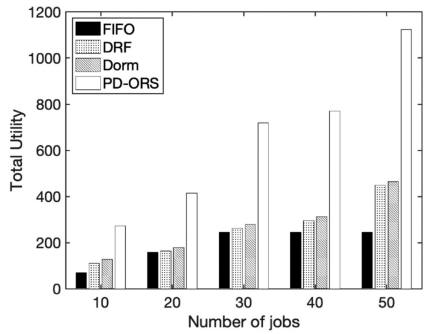


Fig. 7. Total utility with increasing number of jobs (synthetic data).

Fig. 8. Utility comparison between PD-ORS and OASiS with increasing number of jobs.

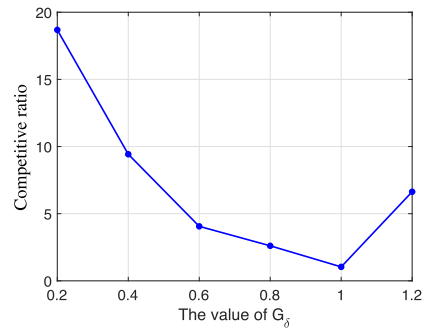


Fig. 9. Median of actual training time comparison.

dynamically, and (3) *Dorm* [12], where the numbers of workers (parameter servers) are computed and placed by an MILP resource utilization maximization problem with fairness and adjustment overhead constraints. Workers and parameter servers are placed in a round-robin fashion on available machines in Baselines (1) and (2). The comparison results are shown in Figs. 6 and 7. In Fig. 6, we set $T = 20$ and $I = 50$, while in Fig. 7, we set $T = 20$ and $H = 100$. We can see that PD-ORS significantly outperforms other policies and the gains in total utility becomes more pronounced as the numbers of jobs and machines increase.

Next, we compare our PD-ORS algorithm with the OASiS algorithm in [6], which is also a dynamic scheduling scheme. As mentioned earlier, the key difference in OASiS is that parameter servers and workers are located on two *strictly* separated sets of machines (i.e., no co-located workers and PSs). Here, we let $H = 100$ and $T = 20$. For OASiS, half of the machines host parameter servers and the other half host workers. For fair comparisons, both algorithms adopt the same Sigmoid utility function. The comparison results are shown in Fig. 8. We can see that PD-ORS outperforms OASiS by allowing co-located parameter servers and workers. We can see from Fig. 8 that the performance gap between PD-ORS and OASiS widens as the number of jobs increases, which implies that PD-ORS is more scalable. This is due to the advantage afforded by colocation of workers and parameter servers, which allows each physical machine to be fully utilized. On the other hand, the strict separation of workers and parameter servers in OASiS may lead to the inability of placing workers on server-side machines, should there be available resources or vice verse.

Next, we investigate the actual training time (completion time - arrival time) under different methods, where $T = 80$, $H = 30$ and $I = 100$. The median of the actual training time is shown in Fig. 9. Here, we simply set its training time to T (i.e., 80) if the job cannot be finished within the scheduling time span $\mathcal{T}$. As we can see from Fig. 9, PD-ORS outperforms other scheduling policies, i.e., it has the smallest median time. Also, due to the co-location advantage of PD-ORS, its median time is smaller compared to OASiS, where workers and parameter servers are placed in strictly separated sets of machines. We expect that the difference between PD-ORS and OASiS will become more noticeable as the number of machines or the capacity of each machine increases since it will allow more co-location placements.

Next, we demonstrate the competitive ratio of our algorithm PD-ORS, which is the ratio between the total job utility of the offline optimal solution and the total job utility achieved by PD-ORS. Recall that Problem DMLRS is a non-convex problem with constraints (e.g., (1)) that are not amenable to be directly solved by conventional optimization techniques. To obtain its offline optimum, all possible combinations of $w_{ih}[t], s_{ih}[t], \forall i, h, t$ need to be considered, which is time prohibitive. Thus, we limit the number of jobs I to 10 and time span T to 10, and the result is shown in Fig. 10. As we can see from the figure, the performance ratio is between 1.0 to 1.4, indicating that our proposed algorithm PD-ORS has a good competitive ratio performance.

Lastly, we examine the performance of the randomized rounding scheme in Algorithm 4, which is the key of PD-

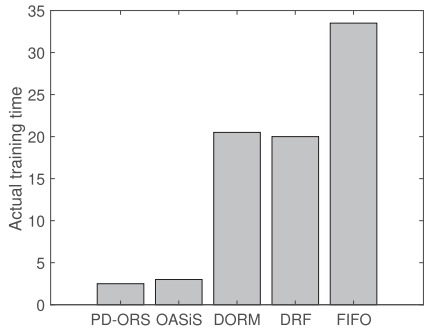


Fig. 10. Competitive ratio.

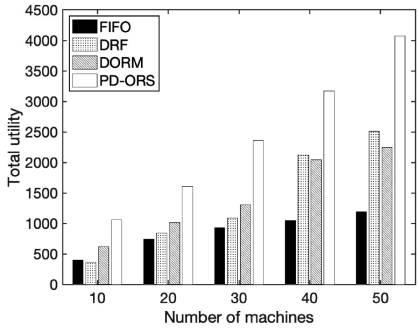


Fig. 12. Total utility with increasing number of machines (Google cluster data trace).

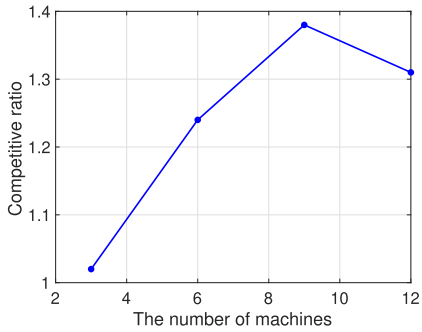


Fig. 11. Impact of pre-rounding gain factor G_δ on competitive ratio.

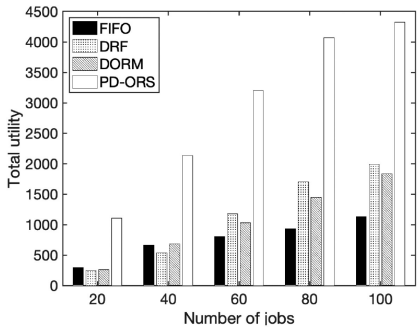


Fig. 13. Total utility with increasing number of jobs (Google cluster data trace).

ORS. We evaluate the rounding performance in terms of the ratio between the optimal total utility and the total utility obtained by our algorithm. The optimal utility is computed using the Gurobi optimization solver. We let $H = 100$, $I = 50$, $T = 20$. We vary the pre-rounding gain factor G_δ (Theorems 3 and 4) from 0.2 to 1.2. The results are shown in Fig. 11. The packing constraints are easier to satisfy with a smaller G_δ, while the cover constraints are prone to be violated as G_δ gets smaller. In our experiments, if the total rounds of randomized rounding before we find an integer feasible solution exceeds a preset threshold (e.g, 5000), we will discard the corresponding job. Theorem 3 suggests that there is a trade-off: if we set G_δ to be close to one to pursue a better total utility result, the rounding time could be large to obtain a feasible solution. As G_δ increases, the probability of violating the packing constraints increases, meaning we need to have more rounding attempts to obtain an integer feasible solution. However, according to our numerical experiences, if the machine's resource capacity is relatively large compared to the jobs' resource demands per worker/PS, the number of rounding attempts is small and not sensitive to the variation of G_δ. On the other hand, as G_δ decreases, the probability of violating the cover constraint increases. However, in practice, the model usually converges with fewer number of iterations than the pre-defined training epochs since the required number of epochs is usually overestimated [40]. In other words, the violation of the cover constraint in one iteration may be acceptable.

As we can see from Fig. 11, the best approximation ratio value is achieved when $G_\delta = 1$. This is because if G_δ approaches 0, it implies that δ decreases at a larger rate (cf. Eq (29)), resulting the increment of the performance ratio. On the other hand, if G_δ goes to infinity, it implies δ decreases (cf. Eq (30)), resulting in a much faster increment of the performance ratio. Also, we can see from Fig. 11 that the performance ratios for all choices of G_δ are much better than the theoretical bounds in Theorems 3 and 4, which shows that the approximation ratio is much tighter than the worse-case bound suggested in Theorems 5 and 6.

Next, we show further experimental results with real-world data traces. We first compare our PD-ORS algorithm with baseline scheduling algorithms, where we follow job arrivals exactly based on timestamps recorded in the Google Cluster data [35] by scaling down the original job trace (i.e., a "snippet" of the trace). Here, we set $T = 80$, $I = 100$ and $H = 30$. The comparison results are shown in Figs. 12–13. Similarly, as we can see from the figures, our algorithm PD-ORS outperforms other scheduling policies. In addition, due to the co-location advantage of PD-ORS, it achieves more total job utility compared to OASiS.

In the previous experiments, we have set the portions for time-insensitive jobs, time-sensitive jobs and time-critical jobs to 10%, 55% and 35%, respectively, which follows the default setting in [6] for fair comparison. Theoretically, the larger the portion of time-sensitive and time-critical jobs is, the better the performance is in terms of job utility compared to other scheduling policies. Based on the Google trace analysis [41], there are four categories of scheduling class of a job to indicate the latency sensitivity of the job, where we label

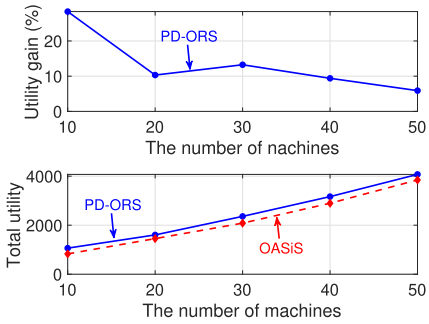


Fig. 14. Total utility with increasing number of machines [T=80, I=100, (10%, 55%, 35%)].

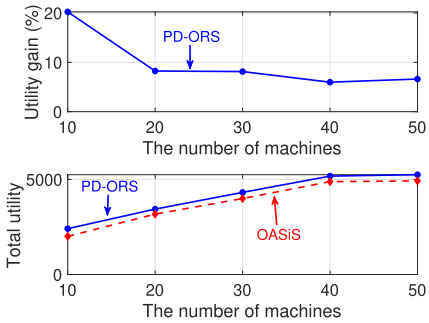


Fig. 15. Total utility with increasing number of machines [T=80, I=100, (30%, 69%, 1%)].

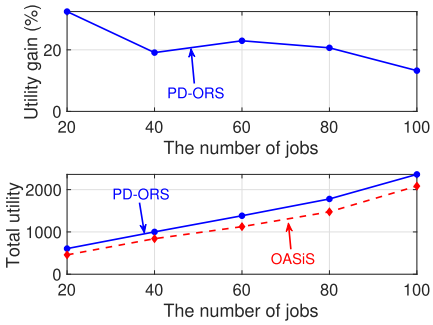


Fig. 16. Total utility with increasing number of jobs [T=80, H=30, (10%, 55%, 35%)].

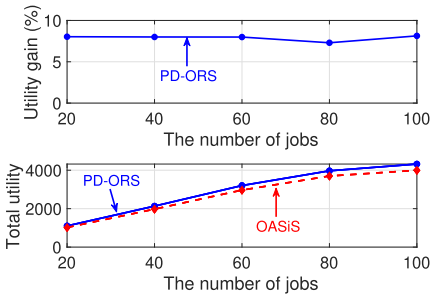


Fig. 17. Total utility with increasing number of jobs [T=80, H=30, (30%, 69%, 1%)].

class 0 as time-insensitive, Classes 1 and 2 as time-sensitive, and Class 3 as time-critical. In order to follow the practical setting in the trace, we roughly set the ratio to 30%, 69% and 1%. We set $T = 80$. The number of machines increases from 10 to 50 with the number of job fixed to 100, and the number of jobs increases from 20 to 100 with the number of machines fixed at 30. We let Figs. 14 and 16 follow the previous ratio setting (i.e., 10%, 55% and 35%), and Figs. 15 and 17 follow the revised ratio setting (i.e., 30%, 69% and 1%). We examine the utility gain compared to OASiS, where it is normalized. We present our experimental results in Figs. 14–17. We can see from the figures that as the portion of critical jobs decreases by 34%, the utility gain becomes smaller. That is, the advantage of our proposed algorithm PD-ORS becomes less prominent.

We note that although in theory we can compare our PD-ORS algorithm (a special case with utility function $u(x) = x$) with Optimus in [23] (which also takes co-location into consideration), it is not straightforward to do so in practice. Optimus requires an offline stage to estimate the θ-parameters of the speed function $f(p_j, w_j)$ (cf. θ_0–θ_3 in [[23], (3)] for asynchronous training and θ_0–θ_4 for synchronous training in [[23], (4)]). Estimating these parameters requires specific hardware and software packages that are not available in our current experimental environment. Due to the above computing resource limitations and time constraints, we are unable to conduct experiments to directly compare PD-ORS and Optimus in this work. Also, our focus in this work is on scheduling algorithmic designs for deep learning training with main contributions being on the theoretical aspects (proving rigorous scheduling performance guarantees). Implementing our PD-ORS algorithm in a similar testbed environment and having a comparison with Optimus is very interesting and will be our next step in future studies.

VI. Conclusion

In this paper, we investigated online resource scheduling for distributed machine learning jobs in a shared computing cluster. We considered the most general setting that workers and parameter servers can be co-located on the same set of physical machines. We showed that this problem can be formulated as a challenging integer nonlinear programming problem with *non-deterministic* constraints. We developed an efficient online scheduling algorithm with competitive ratio guarantee. Our main contributions are three-fold: i) We developed a new analytical model that jointly considers resource locality and allocation; ii) Through careful examinations of worker-server configuration relationships, we resolved the locality ambiguity in the model and reduce the problem to a mixed cover/packing integer program that entails low-complexity approximation algorithm design; iii) We proposed a meticulously designed randomized rounding algorithm to solve the mixed cover/packing integer program and rigorously established its approximation ratio guarantee. Collectively, our results expand the theoretical frontier of online optimization algorithm design for resource optimization in distributed machine learning systems.

VII. APPENDIX A

VIII. PROOF OF LEMMA 1

Proof: Consider the probabilities of the following "bad" events: 1) $\mathbf{c}^\top\hat{\mathbf{x}} > \frac{3G_\delta}{\delta}\mathbf{c}^\top\bar{\mathbf{x}}$; 2) $\exists i$ such that $(\mathbf{A}\hat{\mathbf{x}})_i < a_i$; and 3) $\exists i$ such that $(\mathbf{B}\hat{\mathbf{x}})_i > b_i$. Note that events 2) and 3) can be equivalently rewritten as: 2') $\exists i$ such that $\mathbb{E}\{(\mathbf{A}\hat{\mathbf{x}})_i\frac{W_a}{a_i} < W_a\}$ and 3') $\exists i$ such that $\mathbb{E}\{(\mathbf{B}\hat{\mathbf{x}})_i\frac{W_b}{b_i} > W_b\}$. Since $\mathbb{E}\{\hat{\mathbf{x}}\}=\mathbf{x}'=G_\delta\bar{\mathbf{x}}$, by linearity of expectation, we have:

$$\mathbb{E}\{\mathbf{c}^\top\hat{\mathbf{x}}\} = \mathbf{c}^\top\mathbb{E}\{\hat{\mathbf{x}}\} = \mathbf{c}^\top G_\delta\bar{\mathbf{x}} = G_\delta\mathbf{c}^\top\bar{\mathbf{x}}, \tag{31}$$

$$\mathbb{E}\left\{(\mathbf{A}\hat{\mathbf{x}})_i\frac{W_a}{a_i}\right\} = G_\delta\mathbb{E}\left\{(\mathbf{A}\bar{\mathbf{x}})_i\frac{W_a}{a_i}\right\} \geq G_\delta W_a, \tag{32}$$

$$\mathbb{E}\left\{(\mathbf{B}\hat{\mathbf{x}})_i\frac{W_b}{b_i}\right\} = G_\delta\mathbb{E}\left\{(\mathbf{B}\bar{\mathbf{x}})_i\frac{W_b}{b_i}\right\} \leq G_\delta W_b. \tag{33}$$

Then, by the Markov inequality and (31), we can obtain the probability $\Pr\{\mathbf{c}^\top\hat{\mathbf{x}} > \frac{3G_\delta}{\delta}\mathbf{c}^\top\bar{\mathbf{x}}\} \leq \frac{\delta}{3}$.

Next, we note that each $\hat{x}_j$ can be viewed as a sum of independent random variables in [0,1] as follows: The fixed part of $\lfloor x'_j\rfloor$ is a sum of $\lfloor x'_j\rfloor$ random variables with value 1 with probability 1.

Then, we have that $(\mathbf{B}\hat{\mathbf{x}})_i\frac{W_b}{b_i} = (\sum_j[\mathbf{B}]_{ij}\hat{x}_j)\frac{W_b}{b_i}$ is also a sum of independent random variables in [0,1]. Using the Chernoff bound, we have

$$\Pr\left\{(\mathbf{B}\hat{\mathbf{x}})_i\frac{W_b}{b_i} > (1+\epsilon)G_\delta W_b\right\} \leq \exp\left(-\epsilon^2\frac{G_\delta W_b}{3}\right).$$

Setting $(1+\epsilon)G_\delta=1$, i.e., $\epsilon=\frac{1}{G_\delta}-1$, we have:

$$\Pr\left\{(\mathbf{B}\hat{\mathbf{x}})_i\frac{W_b}{b_i} > W_b\right\} \leq \exp\left(-\left(\frac{1}{G_\delta}-1\right)^2 G_\delta\frac{W_b}{3}\right). \tag{34}$$

Forcing $\exp(-(\frac{1}{G_\delta}-1)^2 G_\delta\frac{W_b}{3})=\frac{\delta}{3r}$ and solving G_δ, we have:

$$G_\delta \triangleq 1+\frac{3\ln(3r/\delta)}{2W_b}-\sqrt{\left(\frac{3\ln(3r/\delta)}{2W_b}\right)^2+\frac{3\ln(3r/\delta)}{W_b}}. \tag{35}$$

Using (32), the Chernoff bound, and following similar arguments, we have:

$$\Pr\left\{(\mathbf{A}\hat{\mathbf{x}})_i\frac{W_a}{a_i} \leq (1-\epsilon)G_\delta W_a\right\} \leq \exp\left(-\epsilon^2\frac{G_\delta W_a}{2}\right).$$

Forcing $\exp(-\epsilon^2\frac{G_\delta W_a}{2}) = \frac{\delta}{3\ m}$ and solving for ϵ, we have $\epsilon = (\frac{2}{G_\delta W_a}\ln(\frac{3\ m}{\delta}))^{\frac{1}{2}}$. It follows that

$$\Pr\{(\mathbf{A}\hat{\mathbf{x}})_i\frac{W_a}{a_i} \leq (1-(\frac{2}{G_\delta W_a}\ln(\frac{3\ m}{\delta}))^{\frac{1}{2}})G_\delta W_a\} \leq \frac{\delta}{3\ m},$$

which implies that:

$$\Pr\left\{(\mathbf{A}\hat{\mathbf{x}})_i \leq a_i\left(1-\sqrt{\frac{2}{G_\delta W_a}\ln\left(\frac{3\ m}{\delta}\right)}\right)G_\delta, \exists i\right\} \leq \frac{\delta}{3\ m}. \tag{36}$$

By using union bound and (34) and (36), we have that events 1)–3) occur with probability less than $\frac{\delta}{3}+m\cdot\frac{\delta}{3\ m}+r\cdot\frac{\delta}{3r}=\delta$, and the proof is complete. ■

IX. APPENDIX B

X. PROOF OF LEMMA 2

Proof: Similar to the case when $0 < G_\delta \leq 1$, we can have the expectation equations for the bad cases as in Eqns. (31)-(33). We also can view each $\hat{x}_j$ as a sum of independent random variables in [0,1] in the same way. Then, we have that $(\mathbf{A}\hat{\mathbf{x}})_i\frac{W_a}{a_i} = (\sum_j[\mathbf{A}]_{ij}\hat{x}_j)\frac{W_a}{a_i}$ is also a sum of independent random variables in [0,1]. Using the Chernoff bound, we have that:

$$\Pr\{(\mathbf{A}\hat{\mathbf{x}})_i\frac{W_a}{a_i} \leq (1-\epsilon)G_\delta W_a\} \leq \exp(-\epsilon^2\frac{G_\delta W_a}{2}).$$

Setting $(1-\epsilon)G_\delta=1$, i.e., $\epsilon=1-\frac{1}{G_\delta}$, we have:

$$\Pr\left\{(\mathbf{A}\hat{\mathbf{x}})_i\frac{W_a}{a_i} \leq W_a\right\} \leq \exp\left(-\left(1-\frac{1}{G_\delta}\right)^2 G_\delta\frac{W_a}{2}\right). \tag{37}$$

Forcing $\exp(-(1-\frac{1}{G_\delta})^2 G_\delta\frac{W_a}{2})=\frac{\delta}{3\ m}$ and solving G_δ, we have:

$$G_\delta \triangleq 1+\frac{\ln(3\ m/\delta)}{W_a}+\sqrt{\left(\frac{\ln(3\ m/\delta)}{W_a}\right)^2+\frac{2\ln(3\ m/\delta)}{W_a}}. \tag{38}$$

Using (32), the Chernoff bound, and following similar arguments, we have:

$$\Pr\left\{(\mathbf{B}\hat{\mathbf{x}})_i\frac{W_b}{b_i} > (1+\epsilon)G_\delta W_b\right\} \leq \exp\left(-\epsilon^2\frac{G_\delta W_b}{3}\right).$$

Forcing $\exp(-\epsilon^2\frac{G_\delta W_b}{3}) = \frac{\delta}{3r}$ and solving for ϵ, we have $\epsilon = (\frac{3}{G_\delta W_b}\ln(\frac{3r}{\delta}))^{\frac{1}{2}}$. It follows that

$$\Pr\left\{(\mathbf{B}\hat{\mathbf{x}})_i\frac{W_b}{b_i} > (1+(\frac{3}{G_\delta W_b}\ln(\frac{3r}{\delta}))^{\frac{1}{2}})G_\delta W_b\right\} \leq \frac{\delta}{3r},$$

which implies that:

$$\Pr\left\{(\mathbf{B}\hat{\mathbf{x}})_i > b_i\left(1+\sqrt{\frac{3}{G_\delta W_b}\ln\left(\frac{3r}{\delta}\right)}\right)G_\delta, \exists i\right\} \leq \frac{\delta}{3r}. \tag{39}$$

By using union bound and (37) and (39), we have that events 1)–3) occur with probability less than $\frac{\delta}{3}+m\cdot\frac{\delta}{3\ m}+r\cdot\frac{\delta}{3r}=\delta$, and the proof is complete. ■

XI. APPENDIX C

XII. PROOF OF THE COMPETITIVE RATIO

We use OPT as the optimal objective value of Problem R-DMLRS, which is also the optimum to Problem D-R-DMLRS. We let $\hat{\pi}_i$ denote the approximate schedule obtained by Algorithm 2, which inexactly solves Problem D-R-DMLRS. Let P_i

and D_i be the primal and dual objective values of Problems R-DMLRS and D-RMLRS after determining the schedule $\hat{\pi}_i$ in Algorithm 1. Let P_0 and D_0 be the initial values of Problems R-DMLRS and D-RMLRS, respectively, where $P_0 = 0$ and $D_0 = \sum_{t\in\mathcal{T}}\sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}} P_h^r[0]C_h^r$. We also let P_I and D_i be the final primal and dual objective values returned by Algorithm 1. We present our main result in Lemma 8.

Lemma 8: If there exists constants $\epsilon \geq 1$, $G_\delta > 0$ and $\delta \in (0, 1]$ such that $P_i - P_{i-1} \geq \frac{\delta/3G_\delta}{\epsilon}(D_i - D_{i-1}), \forall i \in \mathcal{I}$, and if $P_0 = 0$ and $D_0 \leq \frac{1}{2}OPT$, then Algorithm 1 is $\frac{6G_\delta\epsilon}{\delta}$-competitive.

Proof of Lemma 8: Since $P_I = \sum_{i\in\mathcal{I}}(P_i - P_{i-1})$, and $D_I - D_0 = \sum_{i\in\mathcal{I}}(D_i - D_{i-1})$, we can have:

$$\begin{aligned} P_I &= \sum_{i\in\mathcal{I}}(P_i - P_{i-1}) \geq \frac{\delta/3G_\delta}{\epsilon}\sum_{i\in\mathcal{I}}(D_i - D_{i-1}) \\ &= \frac{\delta/3G_\delta}{\epsilon}(D_I - D_0). \end{aligned}$$

By weak duality theorem [42], we have

$$D_I \geq OPT \geq P_I.$$

Thus, it follows that:

$$D_I - D_0 \geq \frac{1}{2}OPT, P_I \geq \frac{\delta/3G_\delta}{\epsilon}(D_I - D_0) \geq \frac{\delta/3G_\delta}{2\epsilon}OPT,$$

and the proof is complete. ■

Next, following similar arguments in [6], [32], we introduce the relationship between the cost and resource consumption before and after processing one job. Let $p_h^{r,i}[t]$ be the unit cost of type-r resource on server h at time t after handling job i. Let $\rho_h^{r,i}[t]$ be the amount of type-r resource allocated to jobs on server h at time t after processing the job i. For ease of our subsequent analysis, we now define the following allocation-cost relation that is implied by Algorithm 1:

Definition 1: The allocation-cost relationship for Algorithm 1 with $\epsilon > 1, G_\delta > 0$ and $\delta \in (0, 1]$ is

$$p_h^{r,i-1}[t](\rho_h^{r,i}[t] - \rho_h^{r,i-1}[t]) \geq \frac{\delta/3G_\delta C_h^r}{\epsilon}(p_h^{r,i}[t] - p_h^{r,i-1}[t]), \quad \forall i \in \mathcal{I}, h \in \mathcal{H}, r \in \mathcal{R}.$$

The allocation-cost relationship shows that the cost in each time slot for scheduling a new job is bounded by the increase of term $C_h^r p_h^r[t]$ in Problem D-R-DMLRS, and the possible increment introduced by randomized rounding in Algorithm 4. This is ensured by the update of the price function and the rounding scheme, respectively.

Lemma 9: If the allocation-cost relationship holds for $\epsilon \geq 1, G_\delta > 0$ and $\delta \in (0, 1]$, then Algorithm 1 ensures $P_i - P_{i-1} \geq \frac{\delta/3G_\delta}{\epsilon}(D_i - D_{i-1}), \forall i \in \mathcal{I}$.

Proof of Lemma 9: For any job $i \in \mathcal{I}$, if job i is rejected, then we have $P_i - P_{i-1} = D_i - D_{i-1} = 0$ according to Problems R-DMLRS and D-R-DMLRS, the inequality must hold. If job i is accepted with schedule π_i, i.e., $x_{\pi_i} = 1$, then the increment value of the primal objective value P_i is

$$P_i - P_{i-1} = u_i(t_{\pi_i} - a_i).$$

Since $x_{\pi_i} = 1$, according to Algorithm 1, the constraint (11) is binding. Then, we can have

$$\begin{aligned} u_i(t_{\pi_i} - a_i) &= \lambda_i + \sum_{t\in\mathcal{T}(\pi_i)}\sum_{h\in\mathcal{H}(\pi_i[t])}\sum_{r\in\mathcal{R}}(\alpha_i^r w_{ht}^{\pi_i} + \beta_i^r s_{ht}^{\pi_i})p_h^r[t] \\ &= \lambda_i + \sum_{t\in\mathcal{T}(\pi_i)}\sum_{h\in\mathcal{H}(\pi_i[t])}\sum_{r\in\mathcal{R}} p_h^r[t] \\ &\quad \times (\rho_h^{r,i}[t] - \rho_h^{r,i-1}[t]). \end{aligned}$$

Similarly, we can have the increment value of the dual objective value D_i as follows:

$$D_i - D_{i-1} = \lambda_i + \sum_{t\in\mathcal{T}(\pi_i)}\sum_{h\in\mathcal{H}(\pi_i[t])}\sum_{r\in\mathcal{R}}(p_h^{r,i}[t] - p_h^{r,i-1}[t])C_h^r.$$

Summing up the allocation-cost relationship over all $t \in \mathcal{T}(\pi_i), h \in \mathcal{H}(\pi_i[t]), r \in \mathcal{R}$, we have

$$\begin{aligned} P_i - P_{i-1} &\geq \frac{\delta/3G_\delta}{\epsilon}(D_i - D_{i-1} - \lambda_i) + \lambda_i \\ &= \frac{\delta/3G_\delta}{\epsilon}(D_i - D_{i-1}) + (1 - \frac{\delta/3G_\delta}{\epsilon})\lambda_i. \end{aligned}$$

As $\lambda_i \geq 0, \epsilon, G_\delta > 0$ and $\delta \in (0, 1]$, we have

$$P_i - P_{i-1} \geq \frac{\delta/3G_\delta}{\epsilon}(D_i - D_{i-1}).$$

This completes the proof of Lemma 9. ■

For specific $h \in \mathcal{H}, r \in \mathcal{R}$, we define ϵ_h^r as the corresponding parameter in the allocation-cost relationship for any job $i \in \mathcal{I}$ and any time slot $t \in \mathcal{T}$. Then, it holds that $\epsilon = \max_{h,r}\{\epsilon_h^r\}$. Without loss of generality, we assume that the resource demand of each worker or parameter server is much smaller compared to the capacity of that resource on one server, i.e., $\alpha_i^r \ll C_h^r, \beta_i^r \ll C_h^r$. This is common in real-world machine learning system as large percentage of resources in the whole server. As $\rho_h^r[t]$ increases from 0 to C_h^r, then we can claim that $d\rho_h^r[t] = \rho_h^{r,i} - \rho_h^{r,i-1}$, and derive a differential version of the allocation-cost relationship, which is defined as follows:

Definition 2: The differential allocation-cost relationship for Algorithm 1 with $\epsilon_h^r \geq 1$ is

$$p_h^r[t]d\rho_h^r[t] \geq \frac{C_h^r}{\epsilon_h^r}dp_h^r[t], \forall t \in \mathcal{T}, h \in \mathcal{H}, r \in \mathcal{R}.$$

Next we show that a feasible ϵ_h^r satisfies the differential allocation-cost relationship with price function $p_h^r[t]$ defined in (12).

Lemma 10: $\epsilon_h^r = \ln\frac{U^r}{L}$, and the price function defined in (12) satisfy the differential allocation-cost relationship.

Proof of Lemma 10: The derivation of the marginal cost function is

$$dp_h^r[t] = p_h^{r'}(\rho_h^r[t])d\rho_h^r[t] = L(\frac{U^r}{L})^{\frac{\rho_h^r[t]}{C_h^r}}\ln(\frac{U^r}{L})^{\frac{1}{C_h^r}}d\rho_h^r[t].$$

The differential allocation-cost relationship is

$$L(\frac{U^r}{L})^{\frac{\rho_h^r[t]}{C_h^r}}d\rho_h^r[t] \geq \frac{C_h^r}{\epsilon_h^r}L(\frac{U^r}{L})^{\frac{\rho_h^r[t]}{C_h^r}}\ln(\frac{U^r}{L})^{\frac{1}{C_h^r}}d\rho_h^r[t],$$

which holds for $\epsilon_h^r \geq \ln(\frac{U^r}{L})$. Then, we can set $\epsilon = \max_{r\in\mathcal{R}}(1, \ln(\frac{U^r}{L}))$, which satisfies the differential allocation-cost relationship. This completes the proof. ■

With the aforementioned lemmas, we are now in a position to prove Theorems 5 and 6. Note that the Theorems provide probabilistic guarantees. We first analyze the performance ratio, which is followed by the probabilistic feasibility discussion for both cases.

Proof of Theorems 5 and 6: According to Lemma 10, the marginal cost function used in Algorithm 1 satisfies the differential allocation-cost relationship with $\epsilon = \max_r(1, \ln\frac{U^r}{L})$. Since the resource demand in a job i is much smaller than the capacity, we can derive

$$d\rho_h^r[t] = \rho_h^{r,i} - \rho_h^{i-1}[t],$$
$$dp_h^r[t] = p_h^{r'}(\rho_h^r[t])(\rho_h^{r,i}[t] - \rho_h^{r,i-1}[t]) = p_h^{r,i}[t] - p_h^{r,i-1}[t].$$

So, the differential allocation-cost relationship in Definition 2 implies the allocation-cost relationship in Definition 1 holds for $\epsilon = \max_r(1, \ln\frac{U^r}{L})$.

According to Algorithm 1, we note that

$$\frac{1}{\mu} \leq \frac{\lceil E_iK_i(\tau_i + 2g_i\gamma_i/(b_i^{(e)}F_i))\rceil \sum_{r\in\mathcal{R}}(\alpha_i^r + \beta_i^r)}{T\sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}}C_h^r}, \forall i \in \mathcal{I}.$$

Then, the minimum amount of overall resource consumption of job i can be computed as:

$$\frac{T\sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}}C_h^r}{\mu} \leq \lceil E_iK_i(\tau_i + 2g_i\gamma_i/(b_i^{(e)}F_i))\rceil \sum_{r\in\mathcal{R}}(\alpha_i^r + \beta_i^r).$$

Then, it follows that:

$$\begin{aligned}
D_0 &= \sum_{t,h,r} LC_h^r \\
&= \sum_{t,h,r} \min_{i\in\mathcal{I},\pi_i\in\Pi_i} \frac{1/(2\mu)u_i(t_{\pi_i} - a_i)C_h^r}{\sum_{r\in\mathcal{R}}\lceil E_iK_i(\tau_i + 2g_i\gamma_i/(b_i^{(e)}F_i))\rceil(\alpha_i^r + \beta_i^r)} \\
&= \frac{T\sum_{h,r}C_h^r}{2\mu}\min_{i,\pi_i}\frac{u_i(t_{\pi_i} - a_i)}{\sum_{r\in\mathcal{R}}\lceil E_iK_i(\tau_i + 2g_i\gamma_i/(b_i^{(e)}F_i)\rceil(\alpha_i^r + \beta_i^r)} \\
&\leq \frac{1}{2}\lceil E_iK_i(\tau_i + 2g_i\gamma_i/(b_i^{(e)}F_i))\rceil \sum_{r\in\mathcal{R}}(\alpha_i^r + \beta_i^r) \\
&\quad \min_{i\in\mathcal{I},\pi_i\in\Pi_i}\frac{u_i(t_{\pi_i} - a_i)}{\sum_{r\in\mathcal{R}}\lceil E_iK_i(\tau_i + 2g_i\gamma_i/(b_i^{(e)}F_i)\rceil(\alpha_i^r + \beta_i^r)}, \forall i \in \mathcal{I}. \\
&\overset{(a)}{\leq} \frac{1}{2}\lceil E_iK_i(\tau_i + 2g_i\gamma_i/(b_i^{(e)}F_i))\rceil \sum_{r\in\mathcal{R}}(\alpha_i^r + \beta_i^r)\frac{u_i(t_{\pi_i} - a_i)}{} \\
&\leq \frac{1}{2}u_i(t_{\pi_i} - a_i) \overset{(b)}{\leq} \frac{1}{2}OPT,
\end{aligned}$$

where (a) follows by selecting $(i, \pi) = \arg\min_{i\in\mathcal{I},\pi_i\in\Pi_i} u_i(t_{\pi_i} - a_i)$, and (b) follows from the assumption that the offline optimal solution accepts at least one job, which is reasonable in real-world machine learning system. Then we have $OPT \geq \min_{i,\pi} u_i(t_{\pi_i} - a_i)$. According to Lemmas 8 and 9, the competitive ratio is proved.

Recall that the randomized rounding algorithm is a key component in our algorithm. Toward this end, we show the probability of obtaining a feasible solution with the proved competitive ratio. Here, we consider the following two cases:

1): When $0 < G_\delta \leq 1$ (Theorem 5): According to Theorem 3, the probability of violating the cover constraint is no greater than $\delta/3$ at each randomized rounding iteration. Recall that our Algorithm 1 runs a predetermined number of S iterations to find a feasible integer solution. Thus the probability that no feasible integer solution returned after S iterations rounding is at most $(\delta/3)^S$. It then follows that the probability of at least one feasible integer solution found is at least $1 - (\delta/3)^S$. Moreover, the number of states (t, v) in the dynamic programming for each job i is $O(TK_iE_i)$. Therefore, with probability greater than $(1 - (\delta/3)^S)^{T_iK_iE_i}$, PD-ORS in Algorithm 1 returns a feasible integer solution with the proved competitive ratio.

2): When $G_\delta > 1$ (Theorem 6): According to Theorem 4, the probability of violating the packing constraint is no greater that $\delta/3(HR + 1)$ at each randomized rounding iteration. Following the similar arguments in *1)*, we can show that with probability greater than $(1 - (\delta/3(HR + 1))^S)^{T_iK_iE_i}$, PD-ORS in Algorithm 1 returns a feasible integer solution with the proved competitive ratio, and the proof is complete. ■

REFERENCES

[1] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 265–283.

[2] T. Chen *et al.*, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," in *Proc. NIPS Workshop Mach. Learn. Syst.*, 2016.

[3] A. Paszke *et al.*, "Pytorch: An imperative style, highperformance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[4] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

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

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

[7] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Operating Syst. Des. Implementation*, 2014, pp. 583–598.

[8] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. 11th USENIX Symp. Operating Syst. Des. Implementation*, 2014, pp. 571–582.

[9] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. Commun. ACM*, vol. 1, no. 51, 2008, pp. 107–113.

[10] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. 2nd ACM SIGOPS/EuroSys Eur. Conf. Comput. Syst.*, 2007, pp. 59–72.

[11] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2015, pp. 1355–1364.
[12] P. Sun, Y. Wen, N. B. D. Ta, and S. Yan, "Towards distributed machine learning in shared clusters: A dynamically-partitioned approach," in *Proc. IEEE Int. Conf. Smart Comput.*, 2017, pp. 1–6.
[13] Q. Zhang, R. Zhou, C. Wu, L. Jiao, and Z. Li, "Online scheduling of heterogeneous distributed machine learning jobs," in *Proc. 21st Int. Symp. Theory, Algorithmic Found., Protocol Des. Mobile Netw. Mobile Comput.*, 2020, pp. 111–120.
[14] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, 2016, pp. 50–56.
[15] W. Chen, Y. Xu, and X. Wu, "Deep reinforcement learning for multi-resource multi-cluster job scheduling," in *Proc. IEEE ICNP*, 2017.
[16] H. Mao, M. Schwarzkopf, S. Venkatakrishnan, and M. Alizadeh, "Learning graph-based cluster scheduling algorithms," in *Proc. SysML*, 2018.
[17] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM Special Int. Group Data Commun.*, 2019, pp. 270–288.
[18] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean, "A hierarchical model for device placement," in *Proc. Int. Conf. Learn. Representations*, 2018.
[19] A. Mirhoseini *et al.*, "Device placement optimization with reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2430–2439.
[20] V. Kovalevskyi, "MXNet distributed training explained in depth," [Online]. Available: https://aswesee.it/mxnet-distributed-training-explained-in-depth-part-1-b90c84bda725
[21] "Distributed training in MXNet," [Online]. Available: https://mxnet.apache.org/versions/1.8.0/api/faq/distributed_training
[22] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training imagenet in 1 hour," 2017, *arXiv:1706.02677*.
[23] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–14.
[24] "Apache Hadoop," [Online]. Available: http://hadoop.apache.org/
[25] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst. - Volume 1*. USA: Curran Associates Inc., 2012, pp. 1223–1231.
[26] T. Cheatham, A. Fahmy, D. Siefanescu, and L. Valiani, "Bulk synchronous parallel computing- A paradigm for transportable software," in *Proc. Hawaii Int. Conf. Syst. Sci.*, 2005, pp. 268–275.
[27] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, 3rd ed. New York, NY, USA: Wiley, 2006.
[28] "Multi-worker training with keras," [Online]. Available: https://www.tensorflow.org/tutorials/distribute/multi_worker_with_keras
[29] "Run deep learning with paddlepaddle on kubernetes," 2017. [Online]. Available: https://kubernetes.io/blog/2017/02/run-deep-learning-with-paddlepaddle-on-kubernetes/
[30] Z. Wang, K. Ji, Y. Zhou, Y. Liang, and V. Tarokh, "Spiderboost: A class of faster variance-reduced algorithms for nonconvex optimization," 2018, *arXiv:1810.10690*.
[31] D. S. Hochbaum, *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum, Ed. Boston, MA: PWS Publishing Company, 1997.
[32] N. Buchbinder and J. (Seffi) Naor, *The Design of Competitive Online Algorithms via a Primal-Dual Approach*, vol. 3. Norwell, MA: Now Publishers Inc., Feb. 2009.
[33] P. M. Vaidya, "An algorithm for linear programming which requires $o(((m+n)n^2+(m+n)^{1.5}n)L)$ algorithmic operations," *Mathematical Programming*, vol. 47, pp. 175–201, 1990.
[34] "Amazon ec2 instances," [Online]. Available: https://aws.amazon.com/ec2/instance-types
[35] C. Reiss *et al.*, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. 3rd ACM Symp. Cloud Comput.*, 2012, pp. 1–13.
[36] Z. Huang, B. Balasubramanian, M. Wang, T. Lan, M. Chiang, and D. H. Tsang, "Need for speed: Cora scheduler for optimizing completiontimes in the cloud," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 891–899.
[37] M. Zaharia *et al.*, "Spark: Cluster computing with working sets," in *Proc. USENIX HotCloud*, 2010, p. 10.
[38] V. K. Vavilapalli *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proc. ACM 4th Annu. Symp. Cloud Comput.*, 2013, pp. 1–16.
[39] B. Hindman *et al.*, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proc. USENIX NSDI*, 2011, pp. 22–22.
[40] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Multi-tenant GPU clusters for deep learning workloads: Analysis and implications," in *Proc. MSR-TR-2018-13*, 2018.
[41] P. Minet, Éric I. RenaultKhoufi, and S. Boumerdassi, "Analyzing traces from a google data center," in *Proc. Int. Wireless Commun. Mobile Comput. Conf.*, 2018, pp. 1167–1172.
[42] D. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA: Athena Scientific, 1999.

Menglu Yu (Graduate Student Member, IEEE) received the B.S. degree from the Department of Electrical and Information Engineering, Hunan University, China, in 2014. She is currently working toward the Ph.D. degree with the Department of Computer Science, Iowa State University. Her primary research interests include optimization for distributed machine learning systems and data centers, as well as network optimization.

Jia Liu (Senior Member, IEEE) received the Ph.D. degree from the Department of Electrical and Computer Engineering, Virginia Tech, in 2010. He is an Assistant Professor with the Department of Electrical and Computer Engineering, The Ohio State University, where he joined in August 2020. From August 2017 to August 2020, he was an Assistant Professor with the Department of Computer Science, Iowa State University. His research areas include theoretical machine learning, control and optimization for stochastic networks, and optimization for data analytics infrastructure and cyber-physical systems. Dr. Liu is a member of ACM. He was the recipient of the numerous awards at top venues, including IEEE INFOCOM'19 Best Paper Award, IEEE INFOCOM'16 Best Paper Award, IEEE INFOCOM'13 Best Paper Runner-up Award, IEEE INFOCOM'11 Best Paper Runner-up Award, and IEEE ICC'08 Best Paper Award. Dr. Liu was the recipient of the NSF CAREER Award in 2020. He was the recipient of the Google Faculty Research Award in 2020. He is also a winner of the LAS Award for Early Achievement in Research from the College of Liberal Arts and Sciences at Iowa State University in 2020, and the Bell Labs President Gold Award in 2001. His research is supported by NSF, AFOSR, AFRL, and ONR.

Chuan Wu (Senior Member, IEEE) received the B. Eng. and M.Eng. degrees from the Department of Computer Science and Technology, Tsinghua University, China, in 2000 and 2002, respectively, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Toronto, Canada, in 2008. Since September 2008, She has been with the Department of Computer Science, the University of Hong Kong, where she is currently a Professor. Her current research focuses on the areas of cloud computing, distributed machine learning systems and algorithms, and intelligent elderly care technologies. She is a member of ACM, and was the Chair of the Interest Group on Multimedia services and applications over Emerging Networks (MEN) of the IEEE Multimedia Communication Technical Committee (MMTC) from 2012 to 2014. She is an Associate Editor of IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE TRANSACTIONS ON MULTIMEDIA, *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY. She was the co-recipient of the best paper awards of HotPOST 2012 and ACM e-Energy 2016.

Bo Ji (Senior Member, IEEE) received the B.E. and M.E. degrees in information science and electronic engineering from Zhejiang University, Hangzhou, China, in 2004 and 2006, respectively, and the Ph.D. degree in electrical and computer engineering from The Ohio State University, Columbus, OH, USA, in 2012. Dr. Ji is an Associate Professor with the Department of Computer Science, Virginia Tech, Blacksburg, VA, USA. Prior to joining Virginia Tech, he was an Associate Professor with the Department of Computer and Information Sciences and a faculty member of the Center for Networked Computing at Temple University, where he was an Assistant Professor from July 2014 to June 2020. He was also a Senior Member of the Technical Staff with AT&T Labs, San Ramon, CA, from January 2013 to June 2014. His research interests include the modeling, analysis, control, optimization, and learning of computer and network systems, such as wired and wireless networks, large-scale IoT systems, high performance computing systems and data centers, and cyber-physical systems. He currently serves on the editorial boards of the IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE INTERNET OF THINGS Journal, and IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY. Dr. Ji is a member of the ACM. He is a National Science Foundation (NSF) CAREER awardee (2017) and an NSF CISE Research Initiation Initiative (CRII) awardee (2017). He is also the recipient of the IEEE INFOCOM 2019 Best Paper Award.

Elizabeth S. Bentley (Member, IEEE) received the B.S. degree in electrical engineering from Cornell University, the M.S. degree in electrical engineering from Lehigh University, and the Ph.D. degree in electrical engineering from University at Buffalo. She was a National Research Council PostDoctoral Research Associate with the Air Force Research Laboratory in Rome, NY. She is currently employed by the Air Force Research Laboratory in Rome, NY, performing in-house research and development in the Networking Technology branch. Her research interests include cross-layer optimization, wireless multiple-access communications, wireless video transmission, modeling and simulation, and directional antennas/directional networking.