Preemptive Scheduling for Distributed Machine Learning Jobs in Edge-Cloud Networks

Ne Wang, Ruiting Zhou, *Member, IEEE,* Lei Jiao, *Member, IEEE,* Renli Zhang, Bo Li, *Fellow, IEEE,* and Zongpeng Li, *Senior Member, IEEE*

***Abstract*—Recent advances in 5G and edge computing enable rapid development and deployment of edge-cloud systems, which are ideal for delay-sensitive machine learning (ML) applications such as autonomous driving and smart city. Distributed ML jobs often need to train a large model with enormous datasets, which can only be handled by deploying a distributed set of workers in an edge-cloud system. One common approach is to employ a parameter server (PS) architecture, in which training is carried out at multiple workers, while PSs are used for aggregation and model updates. In this architecture, one of the fundamental challenges is how to dispatch ML jobs to workers and PSs such that the average job completion time (JCT) can be minimized. In this work, we propose a novel online preemptive scheduling framework to decide the location and the execution time window of concurrent workers and PSs upon each job arrival. Specifically, our proposed scheduling framework consists of: i) a job dispatching and scheduling algorithm that assigns each ML job to workers and decides the schedule to train each data chunk; ii) a PS assignment algorithm that determines the placement of PS. We prove theoretically that our proposed algorithm is $D_{max}(1+1/\epsilon)$-competitive with $(1+\epsilon)$-speed augmentation, where D_{max} is the maximal number of data chunks in any job. Extensive testbed experiments and trace-driven simulations show that our algorithm can reduce the average JCT by up to 30% compared with state-of-the-art baselines.**

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

I. Introduction

With the advances in 5G and the edge computing technologies, artificial intelligence (AI) is expanding its coverage to emerging applications at the edge, *e.g.,* autonomous driving and smart city. Compared to machine learning (ML) over the central cloud, training at the edge enjoys a number of advantages such as low transmission latency, privacy protection, and less bandwidth consumption [1]. Therefore, combining the advantages of the edge and the cloud to facilitate model training seems promising. ML model training customarily exploits data parallelism or/and model parallelism [2] [3], in which data parallelism maintains multiple replicas of models among servers, and model parallelism stores copies of datasets. Data parallelism is more feasible for resource-limited edges, in which parameter server (PS) architecture [4] is commonly used to train ML models [5]–[7]. More specifically, as shown in Fig. 1, each training iteration consists of five steps. First, the job owner uploads its input dataset in data chunks to workers, which are deployed on containers or virtual machines (VMs) to host model copies. Next each worker trains the received data chunks via a local replica of the global ML model and pushes the resulting gradients to PSs, which host global model parameters and are also deployed on containers or VMs. Then the PSs update model parameters using gradients. At last, each worker pulls the latest parameters from PSs for the next training iteration.

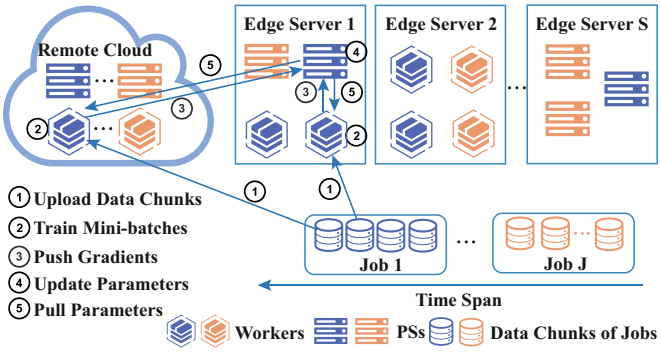

Fig. 1: An illustration of training ML jobs in the edge-cloud system.

This work focuses on the preemptive scheduling of distributed ML jobs in edge-cloud networks by exploring the benefits from both the cloud and the edge. As it turns out that it is challenging to train ML models in an edge-cloud network. First, edge servers are close to the data sources, preferred by ML jobs due to low transmission latency. But edge servers are typically resource-scarce and heterogeneous, so all the workers and PSs involved in an ML job might not be deployed on one edge server. Consequently, there could be frequent communications between workers and PSs across an edge network, which affects the job completion time. The central cloud, on the other hand, experiences longer latency but has abundant resources, where workers and PSs can be easily co-hosted on the same server. Therefore, how to assign ML jobs to workers and PSs while taking advantage of the

Manuscript received December 15, 2021; revised March 10, 2022; accepted April 15, 2022. Date of publication July, 2022; date of current version April 25, 2022. This work is supported in part by the NSFC Grants (62072344 and U20A20177), Hubei Science Foundation (2020CFB195), Compact Exponential Algorithm Project of Huawei (YBN2020035131), Huawei Project (FA2019071041), U.S. National Science Foundation under Grant CNS-2047719, RGC RIF grant R6021-20, and RGC GRF grants under the contracts 16209120 and 16200221. *(Corresponding authors: Ruiting Zhou and Zongpeng Li.)*

N. Wang is with the School of Computer Science, Wuhan University, Wuhan, China. E-mail: {ne.wang}@whu.edu.cn.

R. Zhou and R. Zhang are with the School of Computer Science and the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan, China. E-mail: {ruitingzhou, zhang_rl}@whu.edu.cn.

L. Jiao is with the Department of Computer and Information Science, University of Oregon, Eugene, OR, USA. E-mail: jiao@cs.uoregon.edu.

B. Li is with the Department of Computer Science and Engineering, Hong Kong Science and Technology, Hong Kong. E-mail: bli@cse.ust.hk

Z. Li is with the School of Computer Science, Wuhan University, Wuhan, China, and Insititue for Network Sciences and Cyberspace, Tsinghua University, Beijing, China. E-mail: zongpeng@tsinghua.edu.cn.

edge and the cloud becomes critically important. Second, the long run time of ML jobs keeps resources occupied by jobs that arrive earlier, which may make jobs that arrive later can only be dispatched to the remote cloud. Preemptive scheduling can provide equal opportunities for all jobs, thereby improving resource utilization. However, the existing preemptive schedulers of ML jobs (*e.g.* CoDDL [8] and Optimus [9]) usually need to migrate jobs' datasets among different servers, which can be time-consuming and costly. Thus we need to design a new preemptive algorithm without migrating datasets of ML jobs. Third, preemption granularity is significant to the system's overall efficiency. The training speed of any ML job is typically a concave function on the number of workers [9]. For the jobs whose speeds reach optimal levels, we can reduce part of their workers and cause relatively minor damage to their performance. Therefore the preemption granularity of an efficient scheduler should be a worker rather than a job.

Existing ML job scheduling has mainly focused on either edge or cloud systems [8]–[14], which fail to capture the salient features of edge-cloud networks. In addition, towards better system efficiency, there are proposals advocating elastic resource allocation for ML jobs according to available resources and the utility of jobs [10], [11]. While such approaches are promising but inflexible due to the fixed resource allocation. Recent work has considered scaling the number and adjusting the placement of workers and PSs over the training course [8]–[12] or introducing resource preemption at the job level [13], [14]. Such preemptive schedulers provide great flexibility in scheduling. However, these incur significant overhead to migrate jobs' datasets among different servers dynamically. Besides, the job-level preemption granularity is not efficient enough for scheduling ML jobs.

In this work, we focus on a fundamental problem of training ML jobs in an edge-cloud network: *how to preemptively dispatch ML jobs to workers and PSs such that the average job completion time (JCT) is minimized?* To the best of knowledge, this is *the first* attempt to address the problem of preemptively scheduling ML jobs in edge-cloud networks. Combining the distinct features of edge-cloud networks, we consider their effects on the design of preemptive job scheduling. Specifically, we propose a cost-efficient *preemptive* scheduler that decides the location of workers and PSs for each job only once. That is, when a job is first scheduled, its dataset is uploaded and stored on these workers allocated to it. Henceforth the jobs dispatched to the same worker preemptively occupy the worker anytime. If a job is preempted, it will be suspended, and its dataset is still stored on the worker. This effectively eliminates the need for migrating the datasets of ML jobs. Moreover, our preemptive scheduler is fine-grained, *i.e.,* we allow jobs to *partly* preempt the workers allocated to other jobs, instead of preempting all of them. This preemption is justified since the training process can still work with the remaining workers and all PSs. Finally, we design the scheduling rule by incorporating various factors, *i.e.*, the heterogeneity of edge-cloud networks, the current load of each worker, the collaborative relationship between workers and PSs, and the effects on preempted jobs. Our main contributions are summarized as follows.

- We formulate the problem of minimizing the average JCT of ML jobs as a mixed integer nonlinear program. Even the offline setting is NP-hard, where the complete information of all jobs is given a prior. The challenge further escalates when tackling the online scenario, in which we need to make scheduling decisions without knowing future information. We adopt the unrelated machine scheduling model [15] to reformulate this problem into an average fractional flow time minimization problem, which provides a lower bound of the original problem for performance analysis.
- We design A_{online} to dispatch each ML job to workers and PSs. We first design algorithm A_{greedy} for dispatching jobs to workers to train their input data chunks. For each data chunk of an ML job, A_{greedy} computes the increment of the average training time for each worker, and the minimal is selected for this data chunk. In addition, we prove that the decision of PS allocation is decoupled from worker allocation when conducting A_{greedy}. Subsequently, A_{ps} selects an available PS for each running job without affecting the objectives in both the original problem and of the reformulated problem.
- We employ the primal-dual theory to prove that A_{online} achieves a bounded competitive ratio with speed augmentation, which is an alternative version of the competitive ratio for analyzing unbound online algorithms.
- We carry out both trace-driven simulations and testbed experiments to evaluate the performance of A_{online}. The evaluation results illustrate that: (i) A_{online} reduces the average JCT by up to 35%, 40% and 50% compared to two preemptive baselines [13], [16] and one elastic sharing benchmark [11], respectively; (ii) the number of preemption in A_{online} is relatively small; (iii) A_{online} can complete the training of ML jobs with desired model accuracy; (iv) the practical competitive ratio (< 1.7) is much better than the theoretical bound.

In the rest of the paper, we review related work in Sec. II, and introduce system model in Sec. III. The online preemptive algorithm for scheduling ML jobs is designed and analyzed in Sec. IV. Sec. V presents performance evaluation. Sec. VI extends our algorithm to a capacitated setting and Sec. VII concludes the paper.

II. Related Work

Job Scheduling/Offloading in Edge-Cloud. A large number of researches on job scheduling in edge-cloud focus on efficiently offloading computation/data from mobile devices to the remote cloud or nearby edge servers [17]–[22]. Meng *et al.* [17] propose an online algorithm to jointly manage the allocation of bandwidth and computing resources to serve deadline-aware tasks. Cheng *et al.* [18] study joint resource allocation and computation-intensive applications offloading under energy and computation constraints. Both Zhang *et al.* [19] and Tan *et al.* [20] consider the upload and download delay when dispatching and scheduling jobs. Bista *et al.* [21] consider a probabilistic approach to quantify the probability of successfully offloading tasks to MEC servers. Yang *et al.* [22] investigate the job offloading problem to minimize the

overall weighted sum of energy consumption and running cost. The above schedulers are not tailored for ML jobs and fail to consider the influence on the processing capacity due to frequent communication between workers and PSs.

Distributed Machine Learning Systems. Recent proposals for ML workloads are designed with various motivations, such as fairness [23], [24] and efficiency [8]–[14], [25]–[27]. Mahajan *et al.* [23] design an auction algorithm to achieve fairness in a single-tenant cluster. Zhao *et al.* [24] extend fairness to multi-tenant setting. All studies in [8]–[12], [14] improve the efficiency of training ML jobs by elastic sharing. Our problem setting is similar to [10] and [11], but the techniques adopted to achieve efficiency are radically different. They investigate the effects of elastic resource allocation on job performance, which do not provide real-time adjustment when workload varies. The schedulers in [8], [9], [12]–[14], [26], [27] dynamically re-adjust the number of GPUs or workers according to real-time prediction on remaining workload, job priority, the length and efficiency of jobs, model accuracy, and the prediction of GPU utilization, respectively. As a result, changes in the placement of GPUs require migrating data to new GPUs for training, which incurs heavy overhead. Han *et al.* [25] propose a robust algorithm to gang-schedule placement-sensitive ML jobs while tolerating estimation errors on execution time. In this work, the job owner specifies the number of workers required, and the resource allocation is non-adjustable. While we elastically allocate resources according to the size of the job workload and dynamically adjust the allocation without data migration by enabling worker-level preemption. Xiao *et al.* [26] reduce the fragmentation of resources by packing jobs on as few machines as possible, thereby improving cluster utilization. This work achieves its goal by migrating jobs among servers, which may incur heavy transmission overhead. Qiao *et al.* [27] aim to minimize the total training time of jobs while ensuring fairness among users. They periodically adjust the number of GPUs allocated to each job to realize a fair-resource allocation among users. Such adjustment also leads to job migration among servers, resulting in a high time cost. Moreover, for a job with a given amount of resources, the scheduler in [27] facilitates its training by re-configuring a fit batch size and learning rate. Our work can employ this skill to mitigate the impact of preemption on model accuracy. All the above studies focus on model training in either cloud or edge and cannot utilize both the advantages of the edge and the cloud. Besides, they cannot provide flexible allocation without migrating data.

Our work is inspired by but significantly different from [11]. First, the scheduling in [11] is non-preemptive and may not apply to resource-intensive ML job scheduling in edge servers with the features of heterogeneity and resource scarcity. Second, our algorithmic idea to solve the JCT minimization problem is fundamentally different from [11], as shown in Fig. 2. The compact-exponential technique, first proposed in [28] and adopted in [11], cannot resolve preemptive scheduling since it reserves all resources that each job needs and requires job owners to pay in full. Besides, we adopt the unrelated parallel machine model, which has been used to schedule such jobs that cannot be split and have fixed processing time [15], [20]. As a result, it cannot be directly applied to our original model. Since for each ML job, its training needs the cooperation of multiple workers and PSs, whose number and placement make the JCT unpredictable.

III. System Model

A. System Overview

Edge-Cloud System. As shown in Fig. 1, we consider an edge-cloud system managed by a resource operator, who owns a number of heterogeneous edge servers and a remote cloud. These servers are denoted as a set $[S]$. The cloud is marked with an alias s_c and is regarded as a special server with high latency and unlimited computation capacity. Each server $s \in [S]$ is equipped with many workers and PSs in advance, denoted as $[W_s]$ and $[P_s]$, respectively. The cloud provides infinite workers and PSs, *i.e.,* $[W_{s_c}]$ and $[P_{s_c}]$ include infinite elements. A set of J ML jobs arrives at organizations online with large input data over a large time span $[T] = \{1, 2, ..., T\}$. An binary variable $v_{jsw}(o_{jsp})$ indicates whether a given worker w (PS p) on server s can serve job j or not ('0': no; '1': yes). The value of $v_{jsw}(o_{jsp})$ is determined immediately when job j is released. Let $[X]$ denote the integer set $\{1, 2, \ldots, X\}$.

ML Jobs. An ML job is defined by the following parameters. First, each job $j \in [J]$ consists of D_j equal-sized data chunks, which make up the entire datasets of the job. Furthermore, each data chunk $d \in [D_j]$ consists of B_j equal-sized mini-batches. Job j arrives at time r_j. Upon its arrival, every data chunk d is dispatched to at most one worker [10] [11] with a transmission delay $\triangle_{js}^{\uparrow}$, where $\triangle_{js}^{\uparrow}$ is set identical for all workers on the same server s. Each data chunk of job j needs to be trained for E_j epochs.

Note that job j specifies the type of its workers upon its arrival. Let n_j be the processing capacity of job j, *i.e.,* the number of mini-batches that can be trained by a qualified worker in one time slot. n_j is computed as follows. The time of one worker to process a mini-batch is job-dependent, indicated as m_j. Similarly, the time of one PS to update parameters is denoted as G_j. Specially, PSs can communicate with workers synchronously or asynchronously. We adopt the synchronous stochastic descent gradient [29] under the PS architecture to train various ML models. As a result, the communication time between workers and PSs is computed as follows. Since the size of gradients and parameters are the same [30], we use $\frac{q_j}{b_j}$ to denote the time of sending gradients or receiving updated parameters, where q_j is the size of gradients/parameters and b_j denotes the reserved bandwidth. Note that the above parameters can be obtained by pre-training a tiny partition of datasets. As a special case, when all workers and PSs are placed in one server, such as the cloud, the time to exchange parameters/gradients can be negligible. We denote this case by introducing and setting an indicator variable $\zeta_j = 1$. Different from the cloud, one edge server usually does not have enough resources to place all the workers and PSs of a job together, and hence $\zeta_j = 0$. Thus, we have:

$$n_j = \begin{cases} 1/(m_j + G_j + \dfrac{2q_j}{b_j}), & \text{if} \quad \zeta_{\mathrm{j}} = 0 \\ 1/(m_j + G_j), & \text{if} \quad \zeta_{\mathrm{j}} = 1 \end{cases} \tag{1}$$

Decision Variables. Upon arrival of an ML job j, the following decisions are made: $y_{jsw}(t) \in \{0,1\}$ ($z_{jsp}(t) \in \{0,1\}$), which indicates whether worker w (PS p) of server s is scheduling job j in time t or not. We assume that there is only one PS for each job, which can represent several PS instances placed on the same server [11], [12]. To avoid migration cost, we do not allow migrating jobs' data chunks among servers after the job dispatching decisions are made. Preemption is allowed in servers except for the cloud since an executing job might be suspended and the server can resume it later using the stored image of the job. Important notations are listed in Table I.

TABLE I. Notations

Symbol	Description
$[X]$	integer set $\{1, 2, ..., X\}$
j, s, t	indexes for job, server and time slot, respectively
J, S, T	# of jobs, servers and time slots, respectively
r_j, c_j	arrival and completion time of job j
E_j, D_j	# of training epochs and data chunks of job j
B_j	# of mini-batches in one data chunk of job j
w, p	indexes for worker and PS, respectively
$[W_s], [P_s]$	sets of workers and PSs deployed on server s
$\triangle^{\uparrow}_{js}$	delay to transmit one data chunk of job j to server s
n_j	# of mini-batches trained by one worker of j at a slot
$v_{jsw}(o_{jsp})$	whether j can be dispatched to w (p) on s
$y_{jsw}(t)$	whether worker w on server s is scheduling job j at t
$z_{jsp}(t)$	whether PS p on server s is serving job j at t
n_{jsw}	# of mini-batches trained by w on s of j at one slot
$y_{jdsw}(t)$	whether w on s is scheduling data chunk d of j at t
ζ_j	whether a job's workers and PS are placed together

B. Problem Formulation

Let c_j be the completion time of job j. The total completion time of all jobs is $\sum_{j\in[J]}(c_j - r_j)$. The objective is equivalent to minimizing average JCT, given the total number of jobs, J. We formulate the problem of minimizing total JCT as below.

$$\text{minimize} \quad \sum_{j\in[J]} (c_j - r_j) \tag{2}$$

subject to:

$$y_{jsw}(t) \le v_{jsw}, \forall j, \forall s, \forall w, \forall t \tag{2a}$$

$$\sum_{t\in[T]} \sum_{s\in[S]} \sum_{w\in[W_s]} v_{jsw} y_{jsw}(t) n_j \ge E_j D_j B_j, \forall j \tag{2b}$$

$$\sum_{s\in[S]} \sum_{w\in[W_s]} v_{jsw} y_{jsw}(t) \le D_j, \forall j, \forall t \tag{2c}$$

$$\sum_{j\in[J]} v_{jsw} y_{jsw}(t) \le 1, \forall s, \forall w, \forall t \tag{2d}$$

$$\sum_{s\in[S]} \sum_{p\in[P_s]} o_{jsp} z_{jsp}(t) = 1, \forall j, \forall t : \sum_s \sum_w v_{jsw} y_{jsw}(t) > 0 \tag{2e}$$

$$\sum_{j\in[J]} o_{jsp} z_{jsp}(t) \le 1, \forall s, \forall p, \forall t \tag{2f}$$

$$\zeta_j = 1, \forall j : s = s_c, \forall y_{jsw}(t) = 1, \forall z_{jsp}(t) = 1 \tag{2g}$$

$$\zeta_j = 0, \forall j : s \in [S] \setminus \{s_c\}, \exists y_{jsw}(t) = 1, \exists z_{jsp}(t) = 1 \tag{2h}$$

$$y_{jsw}(t) = z_{jsp}(t) = 0, \forall j, \forall s, \forall w, \forall p, \forall t < r_j + \triangle^{\uparrow}_{js} \tag{2i}$$

$$c_j = \arg\max_{t\in[T]} (\sum_s \sum_w v_{jsw} y_{jsw}(t) > 0), \forall j \tag{2j}$$

$$y_{jsw}(t), z_{jsp}(t), \zeta_j \in \{0,1\}, c_j \in [T], \forall j, \forall s, \forall w, \forall p, \forall t. \tag{2k}$$

Constraint (2a) implies that job j can be only dispatched to qualified workers ($v_{jsw} = 1$). Constraint (2b) guarantees that job j is serviced by sufficient workers. $E_j D_j B_j$ is the total number of mini-batches of job j. Constraint (2c) limits the number of allocated workers to be at most D_j, to ensure that one data chunk is trained by at most one worker for E_j epochs. Constraint (2d) ensures that each worker can serve at most one ML job at every time slot. Constraint (2e) assures that there is one PS allocated to each ML job. Constraint (2f) indicates that each PS can serve at most one ML job at every time slot. Constraints (2g) and (2h) reveal that only if the whole job is dispatched to the cloud (which is regarded as a large sever) can it ignore the communication time between workers and PS. Constraint (2i) implies that the dispatching and scheduling decision can be made only after a job's arrival. Constraint (2j) computes the JCT.

Challenges. The minimization problem of the overall JCT in (2) is a mixed-integer nonlinear programming (MINLP). Even in the offline setting, MINLP (2) is NP-hard [31]. Moreover, the coupling of decision variables $y_{jsw}(t)$ and $z_{jsp}(t)$ affects the value of n_j, and further influences the JCT c_j. Besides, even given n_j and the start training time, c_j cannot be easily determined, since we allow preemption in job scheduling.

Theorem 1 (NP-hardness) *Problem (2) is NP-hard.*

Proof: Please see Appendix A. □

IV. Algorithm Design and Analysis

A. Algorithm Idea

In order to solve MINLP (2), we first apply the unrelated parallel machine model [15] to reformulate it into ILP (4), which provides a lower bound of problem (2). Then we design an online algorithm A_{online} to solve ILP (4) and further tackle MINLP (2). ILP (4) is decomposed into two problems, namely, worker allocation and scheduling problem, and PS assignment problem. The two problems decide $y_{jdsw}(t)$ and $z_{jsp}(t)$ for each job, respectively. We develop algorithm A_{greedy} for dispatching jobs to workers, and prove that the PS allocation $z_{jsp}(t)$ is decoupled with $y_{jdsw}(t)$ in this way. Consequently, we propose a PS assignment algorithm A_{ps}.

i. In Sec. (IV-B), we reformulate MINLP (2) as a minimization problem of the average fractional flow time in ILP (4) and prove that the latter provides a lower bound of the original in Lemma 1.
ii. In Sec. (IV-C), we design a greedy algorithm A_{greedy} to solve the worker allocation problem, *i.e.*, ILP (4) without constraints (4c) and (4d).
iii. In Sec. (IV-D), we introduce algorithm A_{ps} to solve the PS assignment problem. A_{ps} and A_{greedy} constitute the solution of the original problem in (2).

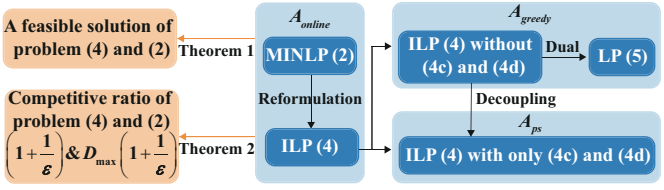

Fig. 2: Main idea of our online algorithm in Sec. IV.

B. Problem Reformulation

The total JCT $\sum_{j\in[J]}(c_j - r_j)$ equals $\sum_{j\in[J]}(\max\{c_{jd}|\forall d \in [D_j]\} - r_j)$, where c_{jd} denotes the completion time of data chunk d in job j. To circumvent the non-conventional objective and constraint (2j), we reformulate MINLP (2) to ILP (4) by employing unrelated parallel machine model. For each data chunk, we calculate its average fractional flow time using Eq. (3). Here we substitute n_{jsw} for n_j to interpret both n_j and v_{jsw}. If job j can be dispatched to worker w on server s, then $n_{jsw} = n_j$, otherwise $n_{jsw} = 0$. Let a binary variable $y_{jdsw}(t) \in \{0, 1\}$ denote whether or not to allocate data chunk d of job j to worker w on server s at time t. The average fractional flow time of one chunk d in job j is tailor-made as Eq. (3), where $\frac{E_j B_j}{n_{jsw}}$ denotes the number of time slots needed to train d in worker w on server s and $\sum_t \frac{y_{jdsw}(t)(t-r_j)}{E_j B_j / n_{jsw}}$ is the total fractional flow time [15].

$$f_{jd}(\boldsymbol{y}) = \frac{1}{D_j}\sum_t \frac{y_{jdsw}(t)(t-r_j)}{E_j B_j / n_{jsw}} = \sum_t \frac{y_{jdsw}(t) n_{jsw}}{E_j D_j B_j}(t - r_j), \tag{3}$$

where $\boldsymbol{y}$ is the collection of all $y_{jdsw}(t)$.

$$\text{minimize} \sum_{j\in[J]}\sum_{d\in[D_j]} f_{jd}(\boldsymbol{y}) \tag{4}$$

subject to:

$$\sum_{t\in[T]}\sum_{s\in[S]}\sum_{w\in[W_s]} y_{jdsw}(t) n_{jsw} \geq E_j B_j, \forall j, \forall d \in [D_j] \tag{4a}$$

$$\sum_{j\in[J]}\sum_{d\in[D_j]} y_{jdsw}(t) \leq 1, \forall s, \forall w, \forall t \geq r_j \tag{4b}$$

$$\sum_{s\in[S]}\sum_{p\in[P_s]} o_{jsp} z_{jsp}(t) = 1, \forall j, \forall t : \sum_d \sum_s \sum_w y_{jdsw}(t) > 0 \tag{4c}$$

$$\sum_{j\in[J]} o_{jsp} z_{jsp}(t) \leq 1, \forall s, \forall p, \forall t \geq r_j \tag{4d}$$

$$\sum_{s\in[S]}\sum_{w\in[W_s]} y_{jdsw}(t) \leq 1, \forall j, \forall d \in [D_j], \forall t \geq r_j \tag{4e}$$

$$\zeta_j = 1, \forall j : s = s_c, \forall y_{jdsw}(t) = 1, \forall z_{jsp}(t) = 1 \tag{4f}$$

$$\zeta_j = 0, \forall j : s \in [S] \setminus \{s_c\}, \exists y_{jdsw}(t) = 1, \exists z_{jsp}(t) = 1 \tag{4g}$$

$$y_{jdsw}(t), z_{jsp}(t), \zeta_j \in \{0, 1\}, \forall j, d, s, w, p, \forall t \geq r_j + \triangle_{js}^{\uparrow} \tag{4h}$$

Constraints (4a), (4b) and (4e) are equivalent to constraints (2b), (2d) and (2c), respectively. Constraints (4c), (4d), (4f) and (4g) are the same as (2e), (2f), (2g) and (2h), respectively. Constraint (2a) and (2i) is implied in $y_{jdsw}(t) \in \{0, 1\}$ and $t \geq r_j + \triangle_{js}^{\uparrow}$, respectively. We next prove that ILP (4) yields a lower bound for MINLP (2) in Lemma 1.

Lemma 1 *For any feasible solution, the objective value of the original MINLP (2) is at least that of ILP (4).*

Proof: Please see Appendix B. □

Algorithmic Challenge. In the simplified version of problem (4), *i.e.,* each server is only endowed with one worker (or PS) and $\forall j, D_j = 1, E_j = 1$, Garg [32] has proven that there is no online algorithm with bounded competitive ratio. We adopt speed augmentation [33] in algorithm analysis, whose formal description is shown in Definition 1, and the corresponding competitive ratio is presented in Definition 2.

Definition 1 *A server is $(1+\epsilon)$-speed if its online running speed is $(1+\epsilon)$ times the offline speed, i.e., any job j can process $(1+\epsilon)p_j$ units of workload, where p_j denotes the processing capacity in the offline setting.*

Definition 2 *An online algorithm is $(1+\epsilon)$-speed α-competitive means that the maximum ratio of the total JCT incurred by our online algorithm with $(1+\epsilon)$-speed over that calculated by the optimal offline algorithm is α.*

Online Algorithm Framework. We design algorithm A_{online} in Alg. 1 to solve ILP (4) and MINLP (2), where A_{online} consists of two subroutines A_{greedy} and A_{ps}. At each time slot, A_{online} first collects the arriving jobs at the current time, denoted as $[J_a]$ (line 2). Then it calls algorithm A_{greedy} to determine the target workers and corresponding schedule for each job in $[J_a]$ (line 3). Because the schedules of jobs may be changed due to preemption, A_{online} invokes algorithm A_{ps} for allocating PS to the running jobs when all arriving jobs have been dispatched and determined schedules (line 4). Finally, it calculates the actual objective of MINLP (2) according to the resulting schedules of all jobs (line 6).

Algorithm 1 Online ML Job Preemptive Scheduling: $\boldsymbol{A_{online}}$

Input: $T, S, [W_s], [P_s], \forall s \in [S]$
Output: $y_{jdsw}(t), z_{jsp}(t), \forall j \in [J], d \in [D_j], s \in [S], w \in [W_s], p \in [P_s], t \in [T], AVG1, AVG$

1: **for** $t \in [T]$ **do**
2: Collect jobs whose $r_j = t$ and add these jobs into set $[J_a]$
3: $\{y_{jdsw}(t), \alpha_{jd}\}_{\forall j,d,s,w,t} = A_{greedy}(T, S, \{[W_s]\}, [J_a])$
4: $\{z_{jsp}(t)\}_{\forall j,s,p,t} = A_{ps}(T, S, \{[P_s]\}, t, [J_a])$
5: **end for**
6: Compute $\sum_j (c_j - r_j)$ based on $\{y_{jdsw}(t), z_{jsp}(t)\}_{\forall j,d,s,w,p,t}$

C. Online Worker Allocation and Scheduling

Main Idea. The **core** issue is how to dispatch each data chunk and determine its schedule window for a job so that the total JCT is minimized. Intuitively, the shortest job first rule can avoid head-of-line blocking caused by long jobs. Inspired by it, our schedule rule is that each worker follows the highest average processing rate first (HAPRF) rule to **schedule** its pending data chunks, where the average processing rate for each data chunk d of job j is defined as $\gamma_{jd} = \frac{n_{jsw}}{E_j D_j B_j}$, which is a job-dependent constant. Based on the schedule rule, the time overhead incurred by dispatching one data chunk to a worker has four parts: transmission delay, waiting time, processing time, and postponed time of preempted data chunks. We thus compute the increment in total average training time of data chunk d by assuming it is dispatched to worker w on server s, and let Q_{jdsw} be the quantity:

$$\frac{1}{D_j}\triangle_{js}^{\uparrow} + \frac{1}{D_j}\sum_{d'\in\mathcal{A}(t_0):d'\to j'} \hat{p}_{j'd'}(t_0)\mathbb{1}(\gamma_{j'd'} \geq \gamma_{jd}) + \frac{1}{D_j}\frac{E_j B_j}{n_{jsw}} + \frac{E_j B_j}{n_{jsw}}\sum_{d'\in\mathcal{A}(t_0):d'\to j'} \frac{1}{D_{j'}}\mathbb{1}(\gamma_{j'd'} < \gamma_{jd}), \tag{5}$$

where $t_0 = r_j + \triangle_{js}^{\uparrow}$, $\mathcal{A}(t_0)$ denotes the pending data chunks when data chunk d arrives at worker w, and $\hat{p}_{j'd'}(t)$ denotes the remaining processing time of data chunk d' in job j' at time t. In Q_{jdsw}, the first item is the average dispatching time of data chunk d, the second item implies the average time of data chunk d waiting for training, the third item is the average processing time, and the last item indicates the average delay

time of data chunks whose average processing rate is lower than data chunk d due to the preemption of data chunk d. With the goal of minimizing total JCT, the **dispatching policy** is naturally devised as follows: it assigns data chunk d to the worker w on server s which makes Q_{jdsw} minimized.

Dual Problem. To analyze the performance of A_{online}, we formulate the dual of ILP (4). First, we relax integrality constraint (4h) to $y_{jdsw}(t) \geq 0$, which means a data chunk is allowed to be trained by multiple workers. As a result, constraint (4e) is redundant. Then we formulate the dual problem of relaxed problem (4) without constraints (4c), (4d), (4f) and (4g) by introducing dual variables α_{jd} and $\beta_{sw}(t)$ to constraints (4a) and (4b), respectively.

$$\text{maximize} \quad \sum_{j\in[J]}\sum_{d\in[D_j]}\alpha_{jd} - \sum_{s\in[S]}\sum_{w\in[W_s]}\sum_{t\in[T]}\beta_{sw}(t) \tag{6}$$

subject to:

$$\alpha_{jd}\frac{n_{jsw}}{E_jB_j} \leq \frac{n_{jsw}}{E_jD_jB_j}(t-r_j) + \beta_{sw}(t), \quad \forall j, \forall d, \forall s, \forall w, \forall t \geq r_j \tag{6a}$$

$$\alpha_{jd}, \beta_{sw}(t) \geq 0, \forall j, \forall d, \forall s, \forall w, \forall t \geq r_j \tag{6b}$$

We now construct a feasible solution to the dual LP (6). First, we set α_{jd} to $\min_{s,w} Q_{jdsw}$. Let the average weight of data chunk d of job j be $\mu_{jd} = \frac{1}{D_j}$, then $\beta_{sw}(t)$ are interpreted as the total average weight of all uncompleted data chunk on worker w on server s at time t, *i.e.,* $\beta_{sw}(t) = \sum_{d\in\mathcal{A}(t):d\to j}\frac{1}{D_j}$. Specially, $\beta_{sw}(t)$ should include the average weight of a data chunk from the time it is released.

Algorithm Design. Our online algorithm A_{greedy} is elaborated in Alg. 2. Upon new job j arrives, A_{greedy} first initiates ζ_j to 0 and defines set $\mathcal{C}_j$ that records Q_{jdsw}, the first scheduled time t^* and set $\mathcal{A}_2$ of preempted data chunks, respectively (lines 1-2). Especially, when $s = s_c$, Q_{jdsw} is computed by setting $\zeta_j = 1$. In lines 3-8, A_{greedy} calculates the initial $Q_{jdsw}, t^*, \mathcal{A}_2$ using the information of any data chunk of job j and stores them in $\mathcal{C}_j$. Next, A_{greedy} considers two cases that the first data chunk is dispatched to the cloud and edge server, respectively. In case 1, since dispatching data chunks to the cloud has no effect on $\mathcal{C}_j$, the subsequent data chunks will be allocated to the cloud (lines 12-16). In Case 2, A_{greedy} first recalculates $Q_{jds_cw_c}$ by resetting $\zeta_j = 0$ (lines 17-20), and then updates $Q_{jds_d^*w_d^*}$ for selected worker w_d^* on server s_d^* (lines 21-22). Finally, the primal and dual variables are updated according to their definitions (lines 23-24). In function CALCULATEQ(j, d, s, w), if the target server is the cloud, then we calculate Q_{jdsw} only with the first and third item in Eq. (5) since there is no preemption in the cloud (lines 2-9). Especially, if data chunk d is the first one, then $\zeta_j = 1$; otherwise, data chunk d is inserted into the queue of pending data chunks according to HAPRF rule. Correspondingly, t^*, $\mathcal{A}_1$ and $\mathcal{A}_2$ can be easily obtained. Note that t^* is at least $(r_j + \triangle_{js}^{\uparrow})$ (lines 10-11). According to $t^*, \mathcal{A}_1$ and $\mathcal{A}_2$, Q_{jdsw} is computed by Eq. (5) (line 12). In function UPDATEVARIABLES(), data chunk d is scheduling during time $[t^*, t^* + \frac{E_jB_j}{n_{jsw}})$, so the corresponding $y_{jdsw}(t)$ is set to 1 (line 2). All data chunks in $\mathcal{A}_2$ preempted by d are postponed for $\frac{E_jB_j}{n_{jsw}}$ time slots (lines 3-7). Moreover, $\beta_{sw}(t)$ increases by the average weight of job j in $[r_j, t^* + E_jB_j/n_{jsw})$ (line 8).

Algorithm 2 Online Worker Allocation and Scheduling: A_{greedy}

Input: $T, S, [W_s], \forall s \in [S], [J_a]$
Output: $y_{jdsw}(t), \alpha_{jd}, \forall j \in [J], d \in [D_j], s \in [S], w \in [W_s], t \in [T]$

1: **for** $j \in [J_a]$ **do**
2: Initiate $\zeta_j = 0$ and set $\mathcal{C}_j = \emptyset, t^* = 0, \mathcal{A}_2 = \emptyset$
3: **for** $s \in [S], w \in [W_s]$ **do**
4: **if** $n_{jsw} > 0$ **then**
5: $Q_{jd_1sw}, t^*, \mathcal{A}_2 =$ CALCULATEQ(j, d_1, s, w)
6: Add $\{Q_{jd_1sw}, t^*, \mathcal{A}_2\}$ into $\mathcal{C}_j$
7: **end if**
8: **end for**
9: **for** $d \in [D_j]$ **do**
10: Set $(s_d^*, w_d^*) = (0, 0), t^* = 0, \mathcal{A}_2 = \emptyset$
11: $(s_d^*, w_d^*), t^*, \mathcal{A}_2 = \arg\min_{s,w,t^*,\mathcal{A}_2}(\{Q_{jdsw} \in \mathcal{C}_j\})$
12: **if** $s_d^* = s_c$ **then**
13: $\forall d, \alpha_{jd} = \min_{s,w,t^*,\mathcal{A}_2}(\{Q_{jdsw} \in \mathcal{C}_j\})$
14: $\forall d, \{\{y_{jdsw}(t)\}, \{\beta_{sw}(t)\}\}_{\forall j,d,s,w,t} =$ UPDATEVARIABLES$(j, d, s_d^*, w_d^*, t^*, \mathcal{A}_2)$
15: **break**
16: **end if**
17: **if** $d = d_1$ **then**
18: $Q_{jds'w'}, t^*, \mathcal{A}_2 =$ CALCULATEQ(j, d, s_c, w_c)
19: Update $Q_{jdsw} = Q_{jds'w'}$ where $(s, w) = (s_c, w_c)$
20: **end if**
21: $Q_{jds'w'}, t^*, \mathcal{A}_2 =$ CALCULATEQ(j, d, s_d^*, w_d^*)
22: Update $Q_{jdsw} = Q_{jds'w'}$ where $(s, w) = (s', w')$
23: $\alpha_{jd} = \min_{s,w,t^*,\mathcal{A}_2}(\{Q_{jdsw} \in \mathcal{C}_j\})$
24: $\{\{y_{jdsw}(t)\}, \{\beta_{sw}(t)\}\}_{\forall j,d,s,w,t} =$ UPDATEVARIABLES$(j, d, s_d^*, w_d^*, t^*, \mathcal{A}_2)$
25: **end for**
26: **end for**

Algorithm 3 Function for Calculating Q_{jdsw}

1: **function** CALCULATEQ(j, d, s, w)
2: **if** $s = s_c$ **then**
3: **if** $d = d_1$ **then**
4: Set $\zeta_j = 1$
5: **else**
6: Set $\zeta_j = 0$
7: **end if**
8: Calculate $Q_{jdsw} = \frac{1}{D_j}\triangle_{js}^{\uparrow} + \frac{1}{D_j}\frac{E_jB_j}{n_{jsw}}, t^* = r_j + \triangle_{js}^{\uparrow}, \mathcal{A}_2 = \emptyset$
9: **else**
10: Calculate t^* and use it as dividing time point to partition the pending data chunks $\mathcal{A}(r_j + \triangle_{js}^{\uparrow})$ into two sets $\mathcal{A}_1, \mathcal{A}_2$
11: Set $t^* = \max(t^*, r_j + \triangle_{js}^{\uparrow})$
12: Calculate Q_{jdsw} by Eq.(5)
13: **end if**
14: **Return** $Q_{jdsw}, t^*, \mathcal{A}_2$
15: **end function**

Algorithm 4 Function for Updating $y_{jdsw}(t)$ and $\beta_{sw}(t)$

1: **function** UPDATEVARIABLES$(j, d, s, w, t^*, \mathcal{A}_2)$
2: Set $y_{jdsw}(t) = 1, \forall t \in [t^*, t^* + \frac{E_jB_j}{n_{jsw}})$
3: **for** $d' : d' \to j' \in \mathcal{A}_2$ **do**
4: **if** $y_{j'd's'w'}(t) = 1 \wedge t \geq t^*$ **then**
5: Set $y_{j'd's'w'}(t) = 0, y_{j'd's'w'}(t + \frac{E_{j'}B_{j'}}{n_{j's'w'}}) = 1$
6: **end if**
7: **end for**
8: Set $\beta_{sw}(t) = \beta_{sw}(t) + \frac{1}{D_j}, \forall t \in [r_j, t^* + \frac{E_jB_j}{n_{jsw}})$
9: **Return** $y_{jdsw}(t), \beta_{sw}(t), \forall j \in [J], d \in [D_j], s \in [S], w \in [W_s], t \in [T]$
10: **end function**

D. Online Parameter Server Assignment

Algorithm Design. To ensure integrity of the solution to problem (2), we next develop the PS assignment policy. According to Alg. 2, for any job, its worker allocation determines whether its PS is placed on the cloud or on edge servers. Moreover, the objective of ILP (4) is independent with $z_{jsp}(t)$. As a result, we only need to make constraints (4c) and (4d) satisfied to generate a feasible PS assignment solution. Our PS assignment rule A_{ps} is as follows: at time slot t, A_{ps} allocates one PS to job j only if it is running. Furthermore, if job j runs at continuous time slots, then A_{ps} keeps the allocated PS unchanged to save server switching overhead, otherwise randomly specifies a PS to it (let $o_{jsp} = 1$). The formal description is shown in Alg. 5.

Algorithm 5 Online Parameter Server Assignment: $\boldsymbol{A_{ps}}$

Input: $T, S, [P_s], \forall s \in [S]$, the current time t_c,the set $[J_a]$ of arrived jobs
Output: $z_{jsp}(t), \forall j \in [J], s \in [S], p \in [P_s], t \in [T]$

```
for j ∈ [J_a] do
    if Σ_{d∈[D_j]} Σ_{s∈[S]} Σ_{w∈[W_s]} y_{jdsw}(t_c) > 0 then
        if t_c > 1 ∧ z_{jsp}(t_c − 1) = 1 then
            Set z_{jsp}(t_c) = 1
        else
            Randomly select one PS p on server s from the set of PSs whose o_{jsp} = 1 and set z_{jsp}(t_c) = 1
        end if
    end if
end for
```

E. Theoretical Analysis

We first analyze the correctness of A_{greedy} and A_{ps} in Lemma 2 and 3, and then analyze the performance of A_{online} in terms of correctness, competitive ratio, and time complexity.

Lemma 2 (Correctness) *A_{greedy} in Alg. 2 computes a feasible solution to problem in (6), and generates feasible $\{y_{jdsw}(t)\}$ of problem in (4) without constraints (4c) and (4d).*

Proof: Please see Appendix C. □

Lemma 3 (Correctness) *A_{ps} generates feasible $\{z_{jsp}(t)\}$ of problem in (4), i.e., constraints (4c) and (4d) are satisfied.*

Proof: According to Alg. 5, Lemma 3 is clearly correct. □

Theorem 2 (Correctness) *A_{online} generates a feasible solution to ILP (4) and original MINLP (2).*

Proof: Combining Lemma 2 and 3, we can conclude A_{online} generates a feasible solution to MINLP (2). □

Theorem 3 (Competitive Ratio) *For ILP (4), the online algorithm A_{online} is $(1+\epsilon)$-speed $(1+1/\epsilon)$-competitive. For MINLP (2), A_{online} is $(1+\epsilon)$-speed $D_{max}(1+1/\epsilon)$-competitive, where $D_{max} = \max_{j\in[J]} D_j$.*

Proof: Please see Appendix D. □

$O(1/\epsilon)$ is the tight competitive ratio of existing unrelated parallel machine models [15]. Moreover, D_j is an input parameter, a deterministic constant independent of the problem size.

Theorem 4 (Polynomial Time) *The time complexity of A_{online} is $O(JHK_{max})$, where $H = \sum_{s\in[S]\setminus s_c} W_s + 1$ and $K_{max} = \sum_j D_j$.*

Proof: Please see Appendix E. □

V. Performance Evaluation

A. Experiment Setup

Testbed Setup. We build an edge-cloud system consisting of nine physical machines via Kubernetes 1.19 [34]. Each machine has 1 GeForce RTX 2060 GPU, 12 CPU cores, 16GB RAM, 500GB HDDs, and a dual-port 1GbE NIC, eight of which serve as edge servers, and the other acts as both the remote cloud and a central scheduler. In our testbed, the checkpoint files that record model parameters for resuming ML jobs are stored by a shared Hadoop Distributed File System (HDFS) [35]. Besides, we select two types of workers, which are equipped with 1 GPU, 3GB RAM and 3 CPU cores, 2GB RAM, respectively. There is one type of PSs, and each hosts 2 CPUs, 3GB RAM.

Workload. We generate 30 deep learning (DL) jobs as the experimental workload by scaling down the original real-world traces [36], [37]. From the traces, we map the submission time of jobs into arrival time (r_j) by setting one time slot to one hour long. The worker type that each job employs is randomly picked from the above two types, and the worker requirement of each job is computed according to the number of GPUs of the real trace. In addition, since there is no training model information in the traces, we randomly pick a model for each job from a pool of six models (Table II). Accordingly, the processing capacity n_j of job j is obtained by pre-training. Other information is set as follows: E_i is randomly picked within the range $[20, 60]$, $\triangle_{js}^{\uparrow}$ for transmitting a data chunk to edge servers and cloud are within $[1, 4]$ and $[10, 15]$ time slots, respectively. T is set to be large enough to complete all jobs.

TABLE II. Deep learning models used for experiments

Model	Dataset	# of Data chunks(D_j) / Mini-batches(B_j)
ResNet-50 [38]	CIFAR10 [39]	27 / 58
ResNet-101 [40]	CIFAR10	27 / 58
GoogLeNet [40]	Caltech101 [41]	115 / 58
LeNet [42]	Caltech101	115 / 58
AlexNet [43]	ImageNet-12 [44]	60 / 58
Inception-BN [45]	ImageNet-12	60 / 58

Simulator. We simulate a large edge-cloud system by selecting 100 servers from the traces and using a virtual remote cloud. The hardware configurations of servers in the simulator follow the distributions in the trace, *i.e.*, for each server, the number of workers, and PSs corresponds to that of GPUs and CPUs in the trace, respectively. For other information about workers, PSs and ML jobs not discussed in the trace, we set them as below according to [10] [11]. The types of workers and PSs are set to be 8 to 10, the type of each worker and PS in each server is randomly picked. The bandwidth of workers of specified type, *i.e.,* b_j, is set within $[100, 5 * 1024]$ Mbps. As for the workload in the simulator, we expand the number of DL jobs to 300 by scaling up the submission times. The other settings of ML jobs are as follows: $m_j \in [0.001, 0.05]$ hour, $G_j \in [10, 100]$ milliseconds, $q_j \in [30, 575]$ MB.

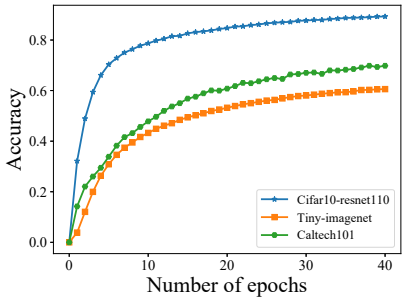

Fig. 3: Accuracy of three ML models.

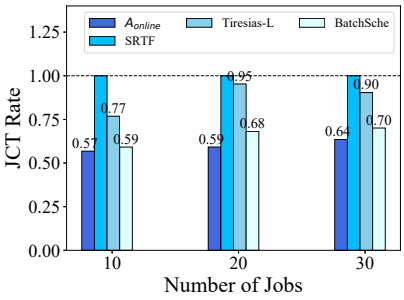

Fig. 4: JCT rate with different J in testbed.

Fig. 5: Preemption frequency with different J in testbed.

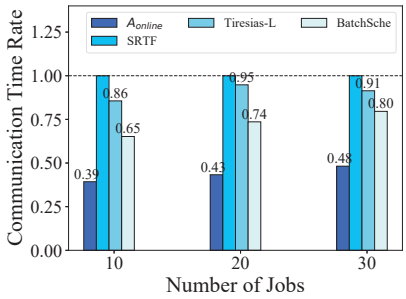

Fig. 6: Communication overhead in testbed.

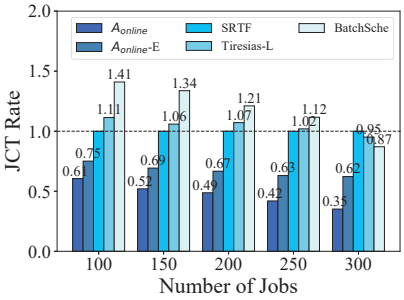

Fig. 7: JCT rate with different J in simulator.

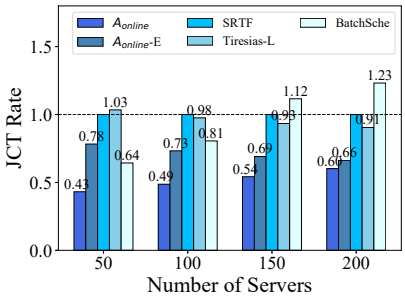

Fig. 8: JCT rate with different S in simulator.

Baselines. We compare A_{online} with the following state-of-the-art and representative ML job scheduling schemes:

- A_{online}-E: A_{online} only schedules jobs on edge servers to show how A_{online} trades off between the edge and cloud.
- Shortest-Remaining-Time-First (SRTF) [16]: jobs are scheduled in the non-decreasing order of remaining training time.
- Tiresias-L [13]: for each job, the algorithm computes its priority according to the number and the occupation time of GPUs allocated, and groups the job into a corresponding priority queue. The jobs of the queue with the highest priority are scheduled first. Besides, the scheduler adopts all-or-nothing resource allocation, which suspends or resumes an entire job in each preemption.
- BatchSche [11]: first groups the uncompleted jobs into multiple batches, then schedules batches one by one to minimize the overall resource cost.

Metrics. We evaluate the model accuracy of DL jobs, JCT rate (the ratio of the total JCT of one algorithm to that of SRTF), and preemption frequency (the total number of preemption triggered) for real experiments. Moreover, we evaluate the JCT rate, preemption frequency, cluster utilization, and actual competitive ratio in large-scale simulations.

B. Testbed Experiments

Accuracy. First, to measure the convergence of our scheduler, we plot the accuracy of three different models trained by A_{online} in Fig. 3. From this figure, we can conclude that all models using A_{online} can converge regardless of their types.

JCT Rate in Testbed. Since it takes too long to train an ML model with complete dataset and the differences on total JCT of different schedulers can also be evaluated by training a small fraction of the entire dataset, we pick 7000 images (101.8MB) from ImageNet ILSVRC2012 dataset to form a smaller scale dataset, *i.e.,* ImageNet-12. Then, we plot the JCT rate of four algorithms in Fig. 4. As observed from Fig. 4, A_{online} outperforms other schedulers. However, because the size of tested jobs is small, the improvement of A_{online} is not noticeable. To this end, we further measure the algorithm performance in our larger-scale simulations to validate the scalability in a large testbed.

Preemption Frequency in Testbed. Finally, we present the preemption frequency of A_{online}, Tiresias-L, and SRTF in Fig. 5. We observe that the preemption in all three algorithms has similar trends, *i.e.*, preemption gets more frequent with the increasing number of tested jobs. Besides, the number of preemption in A_{online} remains relatively low, which reflects the availability of A_{online} in practice. We further make this conclusion credible by conducting large-scale simulations.

Communication Overhead in Testbed. We measure the communication time rate (defined in the similar way to JCT rate) for all algorithms. From Fig. 6, we observe that the ratio of A_{online} is smaller than that of baselines. The performance improvement comes from reducing data migration time and a trade-off between edge and cloud. Compared to SRTF and Tiresias-L, A_{online} avoids migrating data among workers, thereby reducing the communication time. BatchSche rarely has communication overhead between workers and PSs, but the time to upload datasets to the cloud is relatively larger than to the edge.

C. Simulation Studies

JCT Rate in Simulator. Fig. 7 shows the JCT rate of A_{online} and four benchmarks under the different workloads. In any settings, A_{online} and A_{online}-E consistently outperform all others in their category. Besides, we observe the following two

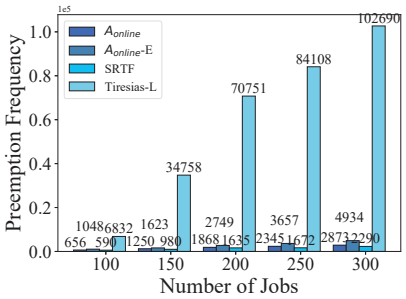

Fig. 9: Preemption frequency with different J in simulator.

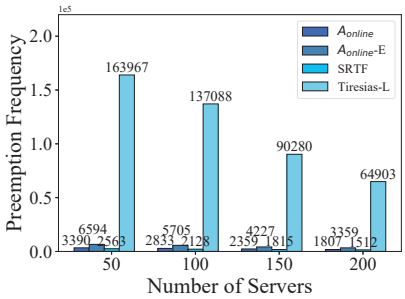

Fig. 10: Preemption frequency with different S in simulator.

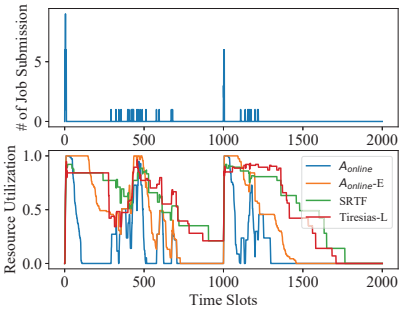

Fig. 11: Job submission and resource utilization.

facts: (i) the reduction of JCT in A_{online} and A_{online}-E over SRTF and Tiresias-L grows as the number of jobs increases. This shows that the fine-grained worker preemption in A_{online} is more effective and competitive than job-level preemptive schedulers (SRTF and Tiresias-L). The key idea of A_{online} is to preempt part of workers instead of suspending the entire job, and these preempted workers are selected by considering the transmission delay, communication time between workers and PSs, current system workload, such that the contribution to the total JCT is minimized (Eq. (5)). (ii) A_{online} performs better than A_{online}-E and BatchSche. Due to the low resource price of the cloud with unlimited capacities, BatchSche schedules jobs to the cloud with a very high probability. The fact that the JCT of A_{online} is smaller than that of BatchSche, implies dispatching most jobs to the cloud with high transmission delay is not a good choice. Moreover, compared with A_{online}-E, the reduction of JCT in A_{online} gets larger with the increasing number of jobs, which is attributed to offload part of jobs to the cloud when resource contention on edge servers becomes heavy. The two comparisons reveal that A_{online} can effectively trade off between the cloud and the edge.

Then we compare the JCT rate of A_{online} and four benchmarks under different numbers of servers in Fig. 8. First, we can observe that the performances of A_{online} and A_{online}-E are better than other baselines regardless of the setting of hardware. Besides, as the amount of edge resources increases, the JCT rates of A_{online} and BatchSche get larger. On the contrary, the JCT rates of A_{online}-E and Tiresias-L become smaller. The reason is that the larger the number of edge servers, the more jobs will be offloaded to the edge. As a result, compared to the case that resources are relatively scarce, the advantages of dispatching jobs to the cloud are not obvious.

Preemption Frequency. Fig. 9 and Fig. 10 show the preemption frequency of four preemptive algorithms under different numbers of jobs and servers, respectively. From the comparison between A_{online} and A_{online}-E, we know that A_{online} can reduce resource contention by offloading overloaded workload to the cloud when computation becomes the bottleneck of training. Although the number of preemption triggered by A_{online} is larger than SRTF, its preemption overhead may not be high. Since the preemption in A_{online} postpones the time window of scheduling on the original worker rather than reassigning the data chunks of preempted jobs to other servers.

Resource Utilization. Fig. 11 shows the job submissions and resource utilization of our cluster over time. We observe that all schedulers have similar trends for different job submissions, and their resource utilization rapidly increases at the two peaks of job submission. Furthermore, the utilization of A_{online} and A_{online}-E can reach 100% due to the elastic sharing in resources. The under-utilization of resources in SRTF and Tiresias-L results from the non-elastic job requirement. Consequently, the makespan of A_{online} is about 0.76× and 0.74× smaller than both SRTF and Tiresias-L, respectively.

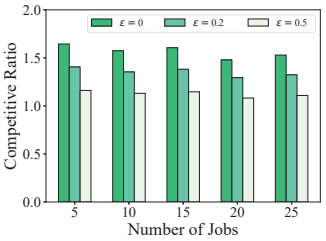

Fig. 12: Competitive ratio of A_{online} with different J, ϵ.

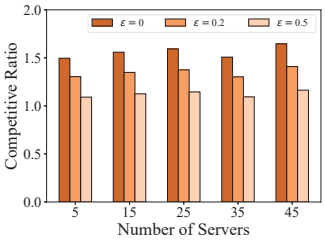

Fig. 13: Competitive ratio of A_{online} with different S, ϵ.

Actual Competitive Ratio. Fig. 12 and Fig. 13 show the competitive ratio of A_{online} with different J and S under diverse speed augmentation levels (ϵ). We simplify ILP (4) by setting $\zeta_j = 1$, whose optimal objective is smaller than that of ILP (4). As a result, we can use the simplified problem to calculate an upper bound of our actual competitive ratio of A_{online}. However, MATLAB has a capacity limit in storing the constraint coefficient matrix. Thus, we consider a reduced input with $[5, 25]$ jobs and $[5, 45]$ servers. Both Fig. 12 and Fig. 13 show that the number of jobs or servers has little influence on the competitive ratio, which implies the performance stability of the proposed algorithm. In addition, the observed competitive ratio remains at a low level (< 1.7) and is much better than the theoretical bound. As indicated by Theorem 2, the larger the ϵ, the smaller the competition ratio, and A_{online} is close to optimal when $\epsilon = 0.5$.

VI. Discussion

In this section, we discuss how to generalize our algorithm to problem (2) with storage capacity constraints.

Problem Formulation. Each server s has a limited storage capacity, denoted by U_s. The size of one data chunk is job-dependent, and we denote it by u_j. Moreover, we introduce decision variables $x_{jds}(t) \in \{0, 1\}$ to indicate whether data chunk d of job j is stored in server s at time t ($x_{jds}(t) = 1$)

or not ($x_{jds}(t) = 0$). Next, problem (2) with storage capacity constraints can be formulated as follows.

$$\text{minimize} \quad \sum_{j\in[J]} (c_j - r_j) \tag{7}$$

subject to:

$$(2a) - (2k)$$

$$y_{jsw}(t) \leq x_{jds}(t), \forall j, d, s, w, t \tag{7a}$$

$$\sum_j \sum_d x_{jds}(t) u_j \leq U_s, \forall s, t \tag{7b}$$

$$x_{jds}(t) = 1, \forall j, d, s, t \in [r_j, c_j] \tag{7c}$$

$$x_{jds}(t) = 0, \forall j, d, s, t \notin [r_j, c_j] \tag{7d}$$

$$x_{jds}(t) \in \{0, 1\}, \forall j, d, s, t \tag{7e}$$

Constraint (7a) ensures that only when a job j's data chunk d is stored at server s can it be scheduled on the server's qualified worker w at time t. Constraint (7b) guarantees that the size of all data chunks stored on a server cannot exceed the storage capacity. Constraints (7c) and (7d) imply that data migration is not allowed, and each data chunk occupies allocated storage resources until it is completed.

Problem Reformulation. Similarly, we adopt the unrelated parallel machine scheduling model to reformulate problem (7) as (8). Lemma 1 clearly holds by substituting (7) and (8) for (2) and (4), respectively.

$$\text{minimize} \quad \sum_{j\in[J]} \sum_{d\in[D_j]} f_{jd}(\boldsymbol{y}) \tag{8}$$

subject to:

$$(4a) - (4h), (7a) - (7e)$$

Algorithm Design. The workers conduct training over data chunks, which typically use one or multiple GPUs in ML clusters. Each GPU is equipped with GPU memory. If a worker is assigned a new data chunk, it first needs to delete the data of the current data chunk from GPU memory and then load the new data chunk into the GPU memory from the storage. Meanwhile, the disk storage keeps all chunks' original data until they are completed. This way, storage can be regarded as another kind of resource independent of workers. In addition, for data chunks stored on a fixed server, we cannot facilitate the training by allocating more storage resources. As a result, the storage constraint only limits the dispatching decisions of data chunks. Once the worker location of a data chunk is decided, the training speed of the data chunk only depends on the worker allocated. To this end, before dispatching one data chunk to a server, we need to first check if the server's storage capacity is enough to store the data chunk. Consequently, servers with sufficient storage and qualified workers are known. Next, we can determine the worker location and occupation time for the data chunk according to the proposed dispatching policy and HAPRF rule in Sec. IV. The remaining capacity of the selected server is thus updated. The above process is repeated until all data chunks of a job are arranged. These steps can be easily implemented by fine-tuning Alg. 2, and we omit the related details here.

Dual Problem. Next, by relaxing $x_{jds}(t) \in \{0, 1\}, y_{jdsw}(t) \in \{0, 1\}$ to $x_{jds}(t) \geq 0, y_{jdsw}(t) \geq 0$, constraints (7a), (7c),(7d) are redundant. Then except the dual variables ($\alpha_{jd}, \beta_{sw}(t)$) in Sec.IV, we introduce a new dual variable $\eta_s(t)$ to constraint (7b) to dualize problem (8) without constraints (4c),(4d),(4f) and (4g). The resulting dual problem is as follows.

$$\max \quad \sum_j \sum_d \alpha_{jd} - \sum_s \sum_w \sum_t \beta_{sw}(t) - \sum_s \sum_t \eta_s(t) U_s \tag{9}$$

subject to:

$$\alpha_{jd} \frac{n_{jsw}}{E_j B_j} \leq \frac{n_{jsw}}{E_j D_j B_j}(t - r_j) + \beta_{sw}(t) + \eta_s(t) u_j, \quad \forall j, \forall d, \forall s, \forall w, \forall t \geq r_j \tag{9a}$$

$$\alpha_{jd}, \beta_{sw}(t), \eta_s(t) \geq 0, \forall j, \forall d, \forall s, \forall w, \forall t \geq r_j \tag{9b}$$

The dual variable $\eta_s(t)$ can be interpreted as the unit price of storage resource on server s at t. Given the above analysis, we can construct a feasible solution to the dual problem, where $\eta_s(t) = 0$ and $\alpha_j, \beta_s(t)$ are set following Sec. IV.

Theoretical Analysis. Based on the constructed primal and dual solutions, we can easily prove that the lemmas and theorems still hold for the related problems (7), (8) and (9).

VII. Conclusion

In this paper, we proposed an online preemptive algorithm to efficiently schedule distributed ML jobs in edge-cloud networks. In particular, no prior work has studied the preemptive scheduling problem in the new emerging edge-cloud/5G system to support a tremendous amount of distributed ML jobs. To minimize the average JCT, we design a new preemptive scheduling framework, A_{online}, which can optimize the transmission delay and the communication time between PS and workers, based on the average processing rate and time, with a provable competitive ratio. Extensive trace-driven simulation studies verify that A_{online} can achieve a near-optimal average JCT, and decrease the average JCT by up to 30% compared with state-of-the-art algorithms. A meaningful extension might be to minimize the bandwidth consumption or to extend our framework to another widely used distributed ML training architecture, *Ring-AllReduce*.

Appendix

A. Proof of Theorem 1

Our problem is a more complex version of the unrelated parallel machine scheduling problem of maximizing the total JCT on identical machines (symbolized as $R||\sum_j c_j$), which is proven NP-hard [46]. Next, we construct a polynomial-time reduction to MINLP (2) from the problem $R||\sum_j c_j$, which is formulated as:

$$\text{minimize} \quad \sum_j c_j \tag{10}$$

subject to:

$$\sum_s \sum_t y_{js}(t) \geq p_j, \forall j \tag{10a}$$

$$\sum_j y_{js}(t) \leq 1, \forall s, \forall t \tag{10b}$$

$$y_{js}(t) \in \{0, 1\}, \forall j, \forall s, \forall t. \tag{10c}$$

p_j is the processing time of job j, and other symbols are similar to our model. First, we assume the system runs only

in edge networks. Besides, we assume the types of workers and PSs are both set to one, and the numbers of workers and PSs are set to one and unlimited, respectively. Thus, the PS allocation of any job can be satisfied, *i.e.*, constraints (2e)-(2h) hold. Moreover, we can regard a worker as a server, *i.e.*, $y_{jsw}(t)$ can be simplified as $y_{js}(t)$. Next, by letting $D_j = 1, r_j = 0, \triangle_{js}^{\uparrow} = 0, x_{jsw} = 1, \forall j, \forall s, \forall w$, constraints (2a),(2c) and (2i) are redundant. Subsequently, by letting $p_j = E_j B_j / n_j$, (10a) is equivalent to (2b). In addition, (10b) is exactly (2d), and the objectives of problem (10) and MINLP (2) are identical. Such mapping can be done in polynomial time. Consequently, problem $R||\sum_j c_j$ can be viewed as a special case of our problem, which must be NP-hard as well.

B. Proof of Lemma 1

For each data chunk $d \in D_j$ of job j, we assume that in solution $\boldsymbol{y}$, it is dispatched to worker w on server s. Note that the objective value is only related to the schedule on workers after the dispatching decisions of workers and PS are made. $f_{jd}(\boldsymbol{y})$ is maximized when data chunk d is processed from $c_{jd} - p_j$ to c_{jd}, where $p_j = \frac{E_j B_j}{n_{jsw}}$. Thus we have

$$\begin{aligned} f_{jd}(\boldsymbol{y}) &\le \sum_{t=c_{jd}-p_j}^{c_{jd}} \frac{n_{jsw}}{E_j D_j B_j}(t - r_j) \\ &= \frac{n_{jsw}}{E_j D_j B_j}(\frac{p_j}{2}(2c_{jd} - p_j) - p_j r_j) \\ &= \frac{1}{D_j}(c_{jd} - \frac{p_j}{2} - r_j) \\ &\le \frac{1}{D_j}(c_{jd} - r_j). \end{aligned}$$

Then, for all data chunks of job j, we have $\sum_j \sum_d f_{jd}(\boldsymbol{y}) \le \sum_j \sum_d \frac{1}{D_j}(c_{jd} - r_j)$. Since for any data chunk d of job j, $c_{jd} \le c_j$ is always satisfied. So,

$$\begin{aligned} &\sum_j \sum_{d\in[D_j]} f_{jd}(\boldsymbol{y}) \le \sum_{d\in[D_j]} \frac{1}{D_j}(c_{jd} - r_j) \\ &\le \sum_j \sum_{d\in[D_j]} \frac{1}{D_j}(c_j - r_j) = \sum_j (c_j - r_j) \end{aligned}$$

Therefore, any feasible solution $\boldsymbol{y}$ to MINLP (2) has a total completion time at least $\sum_j \sum_{d\in[D_j]} f_{jd}(\boldsymbol{y})$.

C. Proof of Lemma 2

Following Alg. 2, the generated $\{y_{jdsw}(t)\}$ of ILP (4) without constraints (4c) and (4d) is clearly feasible. Next we prove constraint (6a) is also satisfied.

We only discuss the case that new data chunk is dispatched to edge servers, because allocating to the cloud is obviously feasible. We fix a worker w on server s and a data chunk d of job j. Note that the arrival of data chunk d will not affect the feasibility of existing data chunks in the same worker. Because α_{jd} remains unchanged once job assigned, and $\beta_{sw}(t)$ can only increase upon the arrival of new data chunks, which helps the inequality in constraint (6a). To this end, we only analyze whether the dual variables of newly arrived data chunk d are feasible. We use $\mathcal{A}_1(\mathcal{A}_2)$ to denote the set of data chunks whose average processing rate is not lower (lower) than data chunk d. We arrange the data chunks in these two sets in ascending order of average processing rate, and let the sequence of data chunks in $\mathcal{A}_1$ and $\mathcal{A}_2$ be $d_1, ..., d_r$ and $d_{r+1}, ..., d_n$, respectively. Suppose at time $t' \ge r_j + \triangle_{js}^{\uparrow}$, worker w is scheduling data chunk d_k. There are three cases:

- Case 1($d_k \in \mathcal{A}_1$):$(t' - r_j) = \triangle_{js}^{\uparrow} + \sum_{d'\in\mathcal{A}_1:d'\to j'} \hat{p}_{j'd'}(t) - \sum_{l=k}^{l=r} \hat{p}_{j_l d_l}(t')$, where $\sum_{d'\in\mathcal{A}_1:d'\to j'} \hat{p}_{j'd'}(t) = \sum_{d'\in\mathcal{A}(t_0):d'\to j'} \hat{p}_{j'd'}(t_0)\mathbb{1}(\gamma_{j'd'} \ge \gamma_{jd})$. Recalling the definition of α_{jd}, *i.e.*, $\alpha_{jd} = \min_{s^*,w^*} Q_{jds^*w^*} \le Q_{jdsw}$, we have

$$\begin{aligned} \alpha_{jd} &\le \frac{1}{D_j} \sum_{d'\in\mathcal{A}(t_0):d'\to j'} \hat{p}_{j'd'}(t_0)\mathbb{1}(\gamma_{j'd'} > \gamma_{jd}) + \frac{1}{D_j}\frac{E_j B_j}{n_{jsw}} \\ &\quad + \frac{1}{D_j}\triangle_{js}^{\uparrow} + \frac{E_j B_j}{n_{jsw}} \sum_{d'\in\mathcal{A}(t'):d'\to j'} \frac{1}{D_{j'}}\mathbb{1}(\gamma_{j'd'} < \gamma_{jd}) \\ &= \frac{1}{D_j}(t' - r_j) + \frac{1}{D_j}\sum_{l=k}^{l=r} \hat{p}_{j_l d_l}(t') + \frac{1}{D_j}\frac{E_j B_j}{n_{jsw}} \\ &\quad + \frac{E_j B_j}{n_{jsw}} \sum_{d'\in\mathcal{A}_2} \frac{1}{D_{j'}} \\ &\le \frac{1}{D_j}(t' - r_j) + \sum_{l=k}^{l=r} \frac{1}{D_{j_l}}\frac{E_j B_j}{n_{jsw}} + \frac{1}{D_j}\frac{E_j B_j}{n_{jsw}} \\ &\quad + \frac{E_j B_j}{n_{jsw}} \sum_{d'\in\mathcal{A}_2} \frac{1}{D_{j'}} \\ &= \frac{1}{D_j}(t' - r_j) + \frac{E_j B_j}{n_{jsw}}(\frac{1}{D_j} + \sum_{l=k}^{l=n} \frac{1}{D_{j_l}}) \\ &= \frac{1}{D_j}(t' - r_j) + \frac{E_j B_j}{n_{jsw}}\beta_{sw}(t'), \end{aligned}$$

  where the third last inequality holds due to the following fact. For data chunk $d_l (l \in [k, r])$, $\gamma_{j_l d_l} \ge \gamma_{jd}$, *i.e.*, $\frac{n_{j_l sw}}{E_{j_l} D_{j_l} B_{j_l}} \ge \frac{n_{jsw}}{E_j D_j B_j} \Rightarrow \frac{1}{D_j}\frac{E_{j_l} B_{j_l}}{n_{j_l sw}} \le \frac{1}{D_{j_l}}\frac{E_j B_j}{n_{jsw}}$. In addition, recalling the definition of $p_{j_l d_l}(t)$, obviously $\hat{p}_{j_l d_l}(t) \le \frac{E_{j_l} B_{j_l}}{n_{j_l sw}}$ for any data chunk d_l of job j_l.
- Case 2($d_k \in \mathcal{A}_2$): In this case, the first $k-1$ data chunks have been completed and data chunk d_k may be partially processed. So we have $(t' - r_j) \ge \triangle_{js}^{\uparrow} + \sum_{d'\in\mathcal{A}_1:d'\to j'} \hat{p}_{j'd'}(t) + \frac{E_j B_j}{n_{jsw}} + \sum_{l=r+1}^{l=k-1} \frac{E_{j_l} B_{j_l}}{n_{j_l sw}}$. So

$$\begin{aligned} \alpha_{jd} &\le \frac{1}{D_j} \sum_{d'\in\mathcal{A}(t_0):d'\to j'} \hat{p}_{j'd'}(t_0)\mathbb{1}(\gamma_{j'd'} > \gamma_{jd}) + \frac{1}{D_j}\frac{E_j B_j}{n_{jsw}} \\ &\quad + \frac{1}{D_j}\triangle_{js}^{\uparrow} + \frac{E_j B_j}{n_{jsw}} \sum_{d'\in\mathcal{A}(t'):d'\to j'} \frac{1}{D_{j'}}\mathbb{1}(\gamma_{j'd'} < \gamma_{jd}) \\ &\le \frac{1}{D_j}(t' - r_j) - \frac{1}{D_j}\sum_{l=r+1}^{l=k-1} \frac{E_{j_l} B_{j_l}}{n_{j_l sw}} + \frac{E_j B_j}{n_{jsw}} \sum_{d'\in\mathcal{A}_2} \frac{1}{D_{j'}} \\ &\le \frac{1}{D_j}(t' - r_j) - \sum_{l=r+1}^{l=k-1} \frac{1}{D_{j_l}}\frac{E_j B_j}{n_{jsw}} + \frac{E_j B_j}{n_{jsw}} \sum_{d'\in\mathcal{A}_2} \frac{1}{D_{j'}} \\ &= \frac{1}{D_j}(t' - r_j) + \frac{E_j B_j}{n_{jsw}} \sum_{l=k}^{l=n} \frac{1}{D_{j_l}} \\ &= \frac{1}{D_j}(t' - r_j) + \frac{E_j B_j}{n_{jsw}}\beta_{sw}(t'), \end{aligned}$$

  where the third last inequality follows from the fact as followings. For data chunk $d_l (l \in [r+1, k-1])$, $\gamma_{j_l d_l} \le \gamma_{jd}$, *i.e.*, $\frac{n_{j_l sw}}{E_{j_l} D_{j_l} B_{j_l}} \le \frac{n_{jsw}}{E_j D_j B_j} \Rightarrow \frac{1}{D_j}\frac{E_{j_l} B_{j_l}}{n_{j_l sw}} \ge \frac{1}{D_{j_l}}\frac{E_j B_j}{n_{jsw}}$.

- Case 3($d_k = d$): $(t' - r_j) \geq \triangle_{js}^{\uparrow} + \sum_{d' \in \mathcal{A}_1 : d' \to j'} \hat{p}_{j'd'}(t)$. We have

$$\begin{aligned}\alpha_{jd} &\leq \frac{1}{D_j} \sum_{d' \in \mathcal{A}(t_0): d' \to j'} \hat{p}_{j'd'}(t_0) \mathbb{1}(\gamma_{j'd'} > \gamma_{jd}) + \frac{1}{D_j} \frac{E_j B_j}{n_{jsw}} \\ &\quad + \frac{1}{D_j} \triangle_{js}^{\uparrow} + \frac{E_j B_j}{n_{jsw}} \sum_{d' \in \mathcal{A}(t'): d' \to j'} \frac{1}{D_{j'}} \mathbb{1}(\gamma_{j'd'} < \gamma_{jd}) \\ &\leq \frac{1}{D_j}(t' - r_j) + \frac{1}{D_j} \frac{E_j B_j}{n_{jsw}} + \frac{E_j B_j}{n_{jsw}} \sum_{d' \in \mathcal{A}_2} \frac{1}{D_{j'}} \\ &= \frac{1}{D_j}(t' - r_j) + \frac{E_j B_j}{n_{jsw}} \beta_{sw}(t')\end{aligned}$$

Thus, $\alpha_{jd}, \beta_{sw}(t)$ is a feasible solution of problem (6).

D. Proof of Theorem 3

To prove Theorem 2, we first investigate Lemma 4.

Lemma 4 $\sum_j \sum_d \alpha_{jd} = \sum_s \sum_w \sum_t \beta_{sw}(t)$.

Proof: Observing Q_{jdsw} *i.e.,* α_{jd}, for data chunk d of a job j, we know that its completion time denoted as c_{jd} is initialized to r_j plus D_j times the sum of the first three item when it arrives. With the arrival of the future jobs, data chunk d may be delayed, and the delay time is the accumulation of processing time of all future data chunks which preempt data chunk d for scheduling, *i.e.,* $\sum_{j'} \sum_{d'} \frac{E_{j'} B_{j'}}{n_{j'sw}}$. Recall the last item of $\alpha_{j'd'}$, the average delay time of data chunk d is actually included into the last item of $\alpha_{j'd'}$ of every data chunk d'. As a result, c_{jd} is updated to add the delay time. Comparing to $\sum_j \sum_d \alpha_{jd}$, its obvious that $\sum_j \sum_d \alpha_{jd} = \sum_j \sum_d \frac{1}{D_j}(c_{jd} - r_j)$. $\sum_j \sum_d \alpha_{jd}$ can be interpreted as the average completion time of all data chunks of all jobs.

Recalling the interpretation of $\sum_s \sum_w \sum_t \beta_{sw}(t)$, it is the accumulation of the average weight of uncompleted data chunks at every time slot, which is equivalent to sum up the total average weight of each data chunk on this worker. And the total average weight of each data chunk d is $\sum_{t=r_j}^{t=c_{jd}} \frac{1}{D_j} = \frac{1}{D_j}(c_{jd} - r_j)$. $\sum_s \sum_w \sum_t \beta_{sw}(t) = \sum_j \sum_d \frac{1}{D_j}(c_{jd} - r_j)$, *i.e.,* $\sum_j \sum_d \alpha_{jd} = \sum_s \sum_w \sum_t \beta_{sw}(t)$. □

According to Definition 1, a worker with $(1+\epsilon)$-speed augmentation implies that it can process n_{jsw} mini-batches whereas only process $n_{jsw}/(1+\epsilon)$ mini-batches in offline setting. The formulation of the offline setting is equivalent to problem (4) without constraints (4c), (4d), (4f) and (4g), and only n_{jsw} is replaced with $n_{jsw}/(1+\epsilon)$. We formulate the dual for the offline setting where the speed is $1/(1+\epsilon)$.

$$\text{maximize} \quad \sum_{j \in [J]} \sum_{d \in [D_j]} \alpha'_{jd} - \sum_{s \in [S]} \sum_{w \in [W_s]} \sum_{t \in [T]} \beta'_{sw}(t) \tag{11}$$

subject to:

$$\alpha'_{jd} \frac{n_{jsw}}{(1+\epsilon) E_j B_j} \leq \frac{n_{jsw}}{(1+\epsilon) E_j D_j B_j}(t - r_j) + \beta'_{sw}(t), \quad \forall j, \forall d, \forall s, \forall w, \forall t \geq r_j \tag{11a}$$

$$\alpha'_{jd}, \beta'_{sw}(t) \geq 0, \forall j, \forall d, \forall s, \forall w, \forall t \geq r_j \tag{11b}$$

Let $\alpha'_{jd} = \alpha_{jd}, \beta'_{sw}(t) = \frac{\beta_{sw}(t)}{1+\epsilon}$. Then we have Lemma 5.

Lemma 5 *The dual variables α'_{jd} and $\beta'_{sw}(t)$ is a feasible solution of problem (11).*

Proof: We know α_{jd} and $\beta_{sw}(t)$ is a feasible solution of problem (6), then we have

$$\begin{aligned}&\alpha_{jd} \leq \frac{1}{D_j}(t - r_j) + \frac{E_j B_j}{n_{jsw}} \beta_{sw}(t) \\ \Leftrightarrow &\alpha'_{jd} \leq \frac{1}{D_j}(t - r_j) + \frac{E_j B_j}{n_{jsw}}(1+\epsilon) \beta'_{sw}(t) \\ \Leftrightarrow &\frac{\alpha'_{jd}}{1+\epsilon} \leq \frac{1}{(1+\epsilon) D_j}(t - r_j) + \frac{E_j B_j}{n_{jsw}} \beta'_{sw}(t).\end{aligned}$$

Thus, Lemma 5 is correct. □

The dual objective value of problem (11) is denoted by D_I, then

$$\begin{aligned}D_I &= \sum_{j \in [J]} \sum_{d \in [D_j]} \alpha'_{jd} - \sum_{s \in [S]} \sum_{w \in [W_s]} \sum_{t \in [T]} \beta'_{sw}(t) \\ &= \sum_j \sum_d \alpha_{jd} - \sum_{s \in [S]} \sum_{w \in [W_s]} \sum_{t \in [T]} \frac{\beta_{sw}(t)}{1+\epsilon} \\ &= \frac{\epsilon}{1+\epsilon} \sum_{j \in [J]} \sum_{d \in [D_j]} \alpha_{jd}.\end{aligned}$$

Let P_I denote the objective value of the primal problem of (11), and OPT_I is the offline optimum of this primal problem. We have $D_I \leq P_I$ by weak duality and $P_I \leq OPT_I$ via LP relaxation. In summary, $D_I \leq OPT_I$, *i.e.,* $\frac{\sum_{j \in [J]} \sum_{d \in [D_j]} \alpha_{jd}}{OPT_I} \leq (1 + 1/\epsilon)$. Recall $\sum_j \sum_{d \in [D_j]} f_{jd}(\boldsymbol{y}) \leq \sum_j \sum_{d \in [D_j]} \frac{1}{D_j}(c_{jd} - r_j)$ in Lemma 1 and $\sum_j \sum_d \alpha_{jd} = \sum_j \sum_d \frac{1}{D_j}(c_{jd} - r_j)$ in Lemma 4, we can conclude the algorithm A_{online} for ILP (4) is $(1+\epsilon)$-speed $(1 + 1/\epsilon)$-competitive.

Let OPT be the offline optimum of MINLP (2), we have $OPT_I \leq OPT$ by Lemma 1. We know $D_I \leq OPT$, then

$$\begin{aligned}D_{max} \sum_j \sum_d \alpha_{jd} &= D_{max} \sum_j \sum_d \frac{1}{D_j}(c_{jd} - r_j) \\ &\geq \sum_j \sum_d (c_{jd} - r_j) \geq \sum_j (c_j - r_j).\end{aligned}$$

We use $AVG = D_{max} \sum_j \sum_d \alpha_{jd}$ as an approximate objective value of MINLP (2), then $\frac{AVG}{OPT} \leq D_{max}(1 + 1/\epsilon)$. The proof is finished.

E. Proof of Theorem 4

For algorithm A_{greedy}, we first analyze the time complexity of functions CALCULATEQ() and UPDATEVARIABLES(). For CALCULATEQ(), both lines 3-8 and lines 11-12 can be done in constant time, whereas line 10 spends $O(K)$ steps to insert new data chunk d into the sorted set of pending data chunks by $StraightInsertionSort$ algorithm, where K is the number of pending data chunks in worker w on server s. All in all, the cost of invoking CALCULATEQ() once is $O(K)$. For UPDATEVARIABLES(), the running time is mainly produced by executing lines 3-7, and the size of $\mathcal{A}_2$ is at most K, *i. e.,* this function's time complexity is also $O(K)$. For each job j, A_{greedy} first calculates at most $(\sum_{s \in [S] \setminus s_c} W_s + 1)$ times Q_{jdsw} by calling CALCULATEQ() (lines 3-8), denotes $\sum_{s \in [S] \setminus s_c} W_s + 1$ as H where 1 is incurred by the cloud. As a result, the execution time of the *loop* in line 3 is $O(HK)$. The *loop* in line 9 can be done in two branches, both of them first execute line 11 in $O(H \log H)$ by $Quicksort$ algorithm where H also upper-bounds the number of all Q_{jdsw}. The first branch is in lines 12-16, where line 13 can be done in $O(D_j)$ time and line 14 takes $O(D_j K)$ steps by invoking $O(D_j)$ times UPDATEVARIABLES(). To sum up, the first branch takes $\max(O(H \log H), O(D_{max} K))$

steps. The second branch is other lines except lines 12-16 of $loop$ in line 9. In this case, line 11 takes $O(H \log H)$ steps only when $d = d_1$, the subsequent data chunks only need $O(H)$ steps to resort by $StraightInsertionSort$ algorithm. The running time of lines 17-20 or lines 21-22 is $O(K)$. In conclusion, the running time of the second branch is $\max(O(H \log H), O(D_{max}H), O(D_{max}K))$. In practice, the maximal K denoted by K_{max} can up to $\sum_j D_j$, which is much larger than H and $H \gg D_{max}$. In a word, the time complexity of A_{greedy} is $O([J_a]HK_{max})$.

As for algorithm A_{ps}, every step can be completed at constant time, and hence the time complexity of A_{ps} is $O([J_a])$.

For algorithm A_{online}, $\sum_{t\in[T]}[J_a] = J$. So the time complexity of A_{online} is $\max(O(JHK_{max}), O(J)) = O(JHK_{max})$.

REFERENCES

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. of ACM SoCC*, 2014.

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

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

[5] Z. Tao and Q. Li, "esgd: Communication efficient distributed deep learning on the edge," in *Proc. of HotEdge*, 2018.

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

[7] H. Hu, D. Wang, and C. Wu, "Distributed machine learning through heterogeneous edge systems," *Proc. of AAAI*, 2020.

[8] C. Hwang, T. Kim, S. Kim, J. Shin, and K. Park, "Elastic resource sharing for distributed deep learning," in *Proc. of USENIX NSDI*, 2021.

[9] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. of ACM EuroSys*, 2018.

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

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

[12] J. Song and M. Kountouris, "Wireless distributed edge learning: How many edge devices do we need?" *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2120–2134, 2021.

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

[14] G. Yeung, D. Borowiec, R. Yang, A. Friday, R. Harper, and P. Garraghan, "Horus: Interference-aware and prediction-based scheduling in deep learning systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 01, pp. 88–100, 2022.

[15] S. Anand, N. Garg, and A. Kumar, "Resource augmentation for weighted flow-time explained by dual fitting," in *Proc. of SODA*, 2012.

[16] L. E. Schrage and L. W. Miller, "The queue M/G/1 with the shortest remaining processing time discipline," *Operations Research*, vol. 14, no. 4, pp. 670–684, 1996.

[17] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: online task dispatching and scheduling with bandwidth constraint in edge computing," in *Proc. of IEEE INFOCOM*, 2019.

[18] N. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/aerial-assisted computing offloading for iot applications: A learning-based approach," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1117–1129, 2019.

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

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

[21] B. B. Bista, J. Wang, and T. Takata, "Probabilistic computation offloading for mobile edge computing in dynamic network environment," *Internet of Things*, vol. 11, p. 100225, 2020.

[22] L. Yang, C. Zhong, Q. Yang, W. Zou, and A. Fathalla, "Task offloading for directed acyclic graph applications based on edge computing in industrial internet," *Information Sciences*, vol. 540, pp. 51–68, 2020.

[23] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and efficient GPU cluster scheduling," in *Proc. of USENIX NSDI*, 2020.

[24] H. Zhao, Z. Han, Z. Yang, Q. Zhang, F. Yang, L. Zhou, M. Yang, F. C. Lau, Y. Wang, Y. Xiong, and B. Wang, "Hived: Sharing a GPU cluster for deep learning with guarantees," in *Proc. of USENIX OSDI*, 2020.

[25] Z. Han, H. Tan, S. H.-C. Jiang, W. Cao, X. Fu, L. Zhang, and F. C. M. Lau, "Spin: Bsp job scheduling with placement-sensitive execution," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2267–2280, 2021.

[26] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective cluster scheduling for deep learning," in *Proc. of USENIX OSDI*, 2018.

[27] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing, "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *Proc. of USENIX OSDI*, 2021.

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

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

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

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

[32] N. Garg and A. Kumar, "Minimizing average flow-time: Upper and lower bounds," in *Proc. of FOCS*, 2007.

[33] J. S. Chadha, N. Garg, A. Kumar, and V. N. Muralidhara, "A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation," in *Proc. of STOC*, 2009.

[34] *"Kubernetes"*, https://kubernetes.io/.

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

[36] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant gpu clusters for dnn training workloads," in *Proc. of USENIX ATC*, 2019.

[37] *Microsoft Philly Trace.*, https://github.com/msr-fiddle/philly-traces.

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

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

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

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

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

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of NIPS*, 2012.

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

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

[46] S. K. Sahni, "Algorithms for scheduling independent tasks," *Journal of the ACM*, vol. 23, no. 1, p. 116–127, 1976.

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

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

Lei Jiao received the Ph.D. degree in computer science from the University of Göttingen, Germany. He is currently an assistant professor at the Department of Computer and Information Science, University of Oregon, USA. Previously he worked as a member of technical staff at Alcatel-Lucent/Nokia Bell Labs in Dublin, Ireland and also as a researcher at IBM Research in Beijing, China. He is interested in the mathematics of optimization, control, learning, and mechanism design applied to computer and telecommunication systems, networks, and services. He publishes papers in journals such as JSAC, ToN, TPDS, TMC, and TDSC, and in conferences such as INFOCOM, MOBIHOC, ICNP, ICDCS, SECON, and IPDPS. He is a recipient of the NSF CAREER Award. He also received the Best Paper Awards of IEEE LANMAN 2013 and IEEE CNS 2019, and the 2016 Alcatel-Lucent Bell Labs UK and Ireland Recognition Award. He was on the program committees of conferences including INFOCOM, MOBIHOC, ICDCS, IWQoS, and ICC, and was also the program chair of multiple workshops with INFOCOM and ICDCS.

Renli Zhang received a B.E. degree in Information Security from Wuhan University, China, in 2020. He is currently pursuing the M.S. degree in the School of Cyber Science and Engineering at Wuhan University. His research interests include UAV-enabled wireless networks, network optimization, and online scheduling.

Bo Li is a Chair Professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He held a Cheung Kong Visiting Chair Professor in Shanghai Jiao Tong University between 2010 and 2016, and was the Chief Technical Advisor for ChinaCache Corp. (NASDAQ:CCIH), a leading CDN provider. He was an adjunct researcher at the Microsoft Research Asia (MSRA) (1999-2006) and at the Microsoft Advanced Technology Center (2007-2008). He made pioneering contributions in multimedia communications and the Internet video broadcast, in particular Coolstreaming system, which was credited as first large-scale Peer-to-Peer live video streaming system in the world. It attracted significant attention from both industry and academia and received the Test-of-Time Best Paper Award from IEEE INFOCOM (2015). He has been an editor or a guest editor for over a two dozen of IEEE and ACM journals and magazines. He was the Co-TPC Chair for IEEE INFOCOM 2004. He is a Fellow of IEEE. He received his PhD in the Electrical and Computer Engineering from University of Massachusetts at Amherst, and his B. Eng. (summa cum laude) in the Computer Science from Tsinghua University, Beijing.

Zongpeng Li received the B.E. degree in computer science from Tsinghua University, in 1999, and the PhD degree from the University of Toronto, in 2005. He has been with the University of Calgary and then Wuhan University. His research interests are in computer networks and cloud computing. He was named an Edward S.Rogers Sr. Scholar, in 2004, won the Alberta Ingenuity New Faculty Award, in 2007, and was nominated for the Alfred P. Sloan Research Fellow, in 2007. He co-authored papers that received Best Paper Awards at the following conferences: PAM 2008, HotPOST 2012, and ACM e-Energy 2016. He received the Department Excellence Award from the Department of Computer Science, University of Calgary, the Outstanding Young Computer Science Researcher Prize from the Canadian Association of Computer Science, and the Research Excellence Award from the Faculty of Science, University of Calgary. He is a senior member of the IEEE.