# Online Scheduling of Machine Learning Jobs in Edge-Cloud Networks

1st Jingping She
*School of Cyber Science and Engineering*
*Wuhan University*
Wuhan, China
email:jingpingshe@qq.com

2nd Ne Wang
*School of Computer Science*
*Wuhan University*
Wuhan, China
email:ne.wang@whu.edu.cn

3rd Ruiting Zhou*
*School of Cyber Science and Engineering*
*Wuhan University*
Wuhan, China
email:ruitingzhou@whu.edu.cn

4th Chen Tian
*Department of Computer Science and Technology*
*Nanjing University*
Nanjing, China
email: tianchen@nju.edu.cn

***Abstract*—Compared with traditional cloud computing, edge-cloud computing brings many benefits, such as low latency, low bandwidth cost, and high security. Thanks to these advantages, a large number of distributed machine learning (ML) jobs are trained on the edge-cloud network to support smart applications, adopting the parameter server (PS) architecture. The scheduling of such ML jobs needs to consider different data transmission delay and frequent communication between workers and PSs, which brings a fundamental challenge: how to deploy workers and PSs on edge-cloud networks for ML jobs to minimize the average job completion time. To solve this problem, we propose an online scheduling framework to determine the location and execution time window for each job upon its arrival. Our algorithm includes: (i) an online scheduling framework that groups unprocessed ML jobs iteratively into multiple batches; (ii) a batch scheduling algorithm that maximizes the number of scheduled jobs in the current batch; (iii) two greedy algorithms that deploy workers and PSs to minimize the deployment cost. Large-scale and trace-driven simulations show that our algorithm is superior to the most common and advanced schedulers in today's cloud systems.**

***Keywords*—online scheduling; machine learning job; parameter server architecture; edge-cloud network**

## I. Introduction

Compared with traditional cloud computing, edge-cloud computing has many advantages, such as low latency, low bandwidth cost and high security. To support smart applications (*e.g.,* autonomous driving and smart city), a large number of distributed machine learning (ML) jobs adopt the parameter server (PS) architecture to train models on large datasets by deploying workers and PSs on the edge-cloud network. The existing ML model training mainly uses data parallelism or model parallelism [1] [2]. Data parallelism maintains multiple copies of the model between servers, while model parallelism stores some copies of the dataset. Since the scale of the model is much smaller than the dataset, the data parallel method is more popular in edge training with limited resources. The PS architecture collaborating with data parallelism is one of main methods for training ML models [3]–[5].

However, the existing researches on scheduling distributed ML jobs in edge-cloud networks have a lot of limitations. *First of all*, the differences between the cloud and edge play an important role in job scheduling. Compared to the cloud, the edge has low transmission delay, which makes ML jobs prefer to run at the edge. However, the scarcity of resources in edge servers makes it impossible to deploy all workers and PSs of an ML job on the same edge server. As a result, the frequent communication between workers and PSs will slow down the training. The situation of the cloud is just the opposite. Therefore, how to make a trade-off between transmission delay and computing power is significant to model training. *Secondly*, the resources on the edge server are limited, which need to be reasonably allocated to meet the needs of jobs as much as possible. Therefore, in the case of limited resources, how to allocate resources to jobs to minimize the average job completion time is an important issue. *At last*, the training of ML models is time-consuming and resource-intensive [6]. The online scheduling does not know the arrival time and resources required for future jobs, the jobs that arrive early have a better chance of being favorably scheduled. If the input dataset of the later job is very large, there may not be enough resources to allocate to them, and they can only be assigned to the cloud. As a result, the high transmission delay may become the bottleneck of model training.

To this end, we study the online scheduling problem that minimizes the average job completion time of all ML jobs in the edge-cloud network, taking the above challenges into consideration. Our main contributions are listed as follows:

***First***, we analyze the scheduling scenario of distributed ML jobs in edge-cloud networks in details, and model the problem as a mathematical optimization problem that minimizes the total completion time. This model accurately describes the

* Ruiting Zhou is the corresponding author. This work is supported in part by the NSFC Grants (62072344 and U20A20177), Hubei Science Foundation (2020CFB195) and Compact Exponential Algorithm Project of Huawei (YBN2020035131).

characteristics of distributed ML jobs and considers all factors that affect scheduling jobs.

***Second***, we propose an online scheduling framework $A_{online}$. The time horizon is divided into small time intervals in a geometrically increasing manner, and the original problem is transformed into multiple batch scheduling problems. Then, each batch scheduling problem is transformed into the problem of processing as many jobs as possible in this time interval. Next, we formulate the dual of the problem. Finally, we design a greedy algorithm along with two subroutines to deal with the dual problem, and further solve the original one.

***Third***, we carry out large-scale and trace-driven simulations to evaluate the proposed online scheduling algorithm, and compare it with existing schedulers of ML jobs in cloud systems. The results further confirmed the superiority of our algorithm.

In the rest of the paper, we review related work in Sec. II, and introduce system model in Sec. III. The design of the online scheduling algorithm are presented in Sec. IV and Sec. V. Performance evaluation is presented in Sec. VI and Sec. VII concludes the paper.

## II. Related Work

***Job Scheduling for Distributed ML System.*** Bao *et al.* [7] design an online job scheduling algorithm with the goal of maximizing the overall jobs' utility, and determine the adjusted number of concurrent workers and PSs according to each job's completion time. Zhang *et al.* [8] consider the demand elasticity and resource allocation of ML jobs to design the scheduling algorithm. Li *et al.* [9] study an ML proxy service that aggregates geographically distributed ML jobs into a cloud data center, and dynamically places and expands workers and PSs online in a single job to realize batch discounts. Hsieh *et al.* [10] introduce a geographic distributed ML system, and propose a new ML synchronization model to dynamically eliminate unimportant communication between data centers. Different from the above researches, our goal is to schedule ML jobs in the edge-cloud network to get good performance.
***Job Scheduling in Edge-cloud.*** A considerable amount of researches [11]–[14] has focused on how to implement real-time inference or develop edge intelligence enabling technologies. Zeng *et al.* [11] design an on-demand collaborative DNN reasoning framework for edge intelligence. Li *et al.* [12] propose a framework for DNN collaborative reasoning using edge computing through device-edge collaboration. Wang *et al.* [13] propose an algorithm that effectively schedules multiple training jobs and minimizes the job completion time. Mcmahan *et al.* [14] propose a deep network federated learning method based on iterative model averaging.

In addition, computing/data offloading from mobile users to the cloud/edge servers have been widely investigated [15]–[18]. Meng *et al.* [15] propose an online algorithm that takes into account the management of network bandwidth and computing resources to meet the maximum number of deadlines. Xu *et al.* [16] propose an efficient online algorithm for the service caching problem in the MEC system, which jointly optimizes dynamic service caching and job offloading. Both Zhang *et al.* [17] and Tan *et al.* [18] consider the upload and download delay when dispatching and scheduling jobs. The above researches well studied the characteristics of edge system, but they did not focus on ML job scheduling.

Different from above literature, our work studies the online scheduling and deployment of workers and PSs on the edge-cloud network for distributed ML jobs to reduce the average completion time of jobs.

## III. System Model

### A. System Overview

***Edge-Cloud System.*** As shown in Fig. 1, we consider an edge-cloud network consisting of multiple heterogeneous edge servers and a remote cloud, denoted as a set $[S]$. The number of types of workers and PSs are denoted as $U$ and $V$, respectively. Each server $s \in [S]$ has $W_{su}$ type-$u$ workers and $P_{sv}$ type-$v$ PSs. A set of ML jobs arrive online within a large time span $[T] = \{1, 2, ..., T\}$, with large input datasets. Job $j$ has specified the type of worker and PS to be used when it arrives at time $a_j$, denoted by $u_j$ and $v_j$. Let $[X]$ denote the set of integers $\{1, 2, \ldots, X\}$.

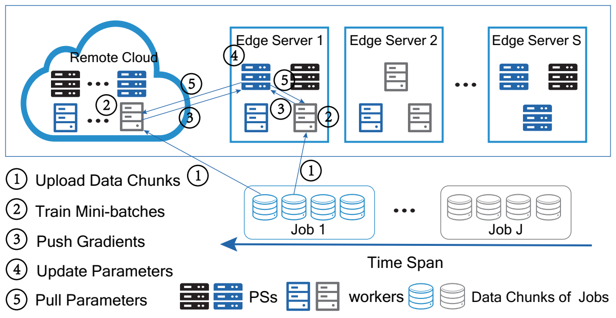


Fig. 1. Edge-cloud System.

***Training Process with PS Framework.*** We adopt the PS architecture [19] to train models. In this architecture, each worker inputs a dataset to train an ML job model. Then the gradient generated by worker is passed to all the PSs for model parameter update. Next, the workers synchronously or asynchronously pull the updated parameters from the PSs. The entire dataset is traversed once means one epoch is completed, and model training is completed after sufficient epochs. Compared to asynchronous mode, the synchronous stochastic descent gradient method (S-SGD) [20] is used widely due to the better performance on convergence.
***Information of ML Job.*** The dataset of job $j \in [J]$ is divided into $D_j$ equal-size data chunks, and each data chunk is further divided into $M_j$ equal size mini-batches. Each data chunk $d \in D_j$ is allocated to at most one worker [7] [8]. The delay of transmitting one data chunk in job $j$ to server $s$ is $\triangle^{\uparrow}_{js}$, which is the same for all workers on the same server $s$. Job $j$ requires $E_j$ epochs to complete its training.

To quantify the number of mini-batches that a worker of type $u_j$ and a PS of type $v_j$ can train for job $j$ in one time slot, denoted by $p_j$, we need to calculate the time to train a mini-batch. This time consists of the following three parts: (i) the time that a worker processes a mini-batch, denoted as $n_j$; (ii) similarly, the time that a PS updates the parameters is indicated by $H_j$; (iii) the communication time for pushing gradients and pulling parameters, denoted as $\frac{2k_j}{B_j}$, where $k_j$ is the size of gradient/parameter [6] and $B_j$ is the bandwidth. When all the workers and PSs of job $j$ are placed together, this communication time can be eliminated. We introduce and set a variable $\xi_j = 1$ to represent this case, otherwise $\xi_j = 0$. Therefore, $p_j$ can be expresses as follows:

$$p_j = \begin{cases} 1/(n_j + H_j + \frac{2k_j}{B_j}), & \text{if} \quad \xi_j = 0 \\ 1/(n_j + H_j), & \text{if} \quad \xi_j = 1 \end{cases} \tag{1}$$

***Decision Variables.*** For each job $j$, we need to decide $y_{jsu_j}(t) \in \{0, 1, 2, ...\}$ ($z_{jsv_j}(t) \in \{0, 1\}$), which represents the number of type-$u_j$ worker (type-$v_j$ PS) on server $s$ allocated to job $j$ at time $t$. We assume that there is only one PS for each job, which can represent a number of PS instances placed on the same server. Important notations are listed in Table I.

## B. Problem Formulation

***Problem Formulation.*** Let $a_j$, $h_j$ and $c_j$ be the arrival, start and completion time of job $j$, respectively. The total completion time of all jobs is $\sum_{j\in[J]}(c_j - a_j)$. The objective is equivalent to minimize the average job completion time, which is formulated as below.

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

subject to:

$$\sum_{s\in[S]} y_{jsu_j}(t) \le D_j, \forall j, \forall t \tag{2a}$$

$$\sum_{s\in[S]} z_{jsv_j}(t) = 1, \forall j, \forall t : \sum_{s\in[S]} y_{jsu_j}(t) > 0 \tag{2b}$$

$$y_{jsu_j}(t) = y_{jsu_j}(t{+}1), \forall j, \forall s, \forall t \in [h_j, c_j] \tag{2c}$$

$$z_{jsv_j}(t) = z_{jsv_j}(t{+}1), \forall j, \forall s, \forall t \in [h_j, c_j] \tag{2d}$$

$$\sum_{t\in[T]} \sum_{s\in[S]} y_{jsu_j}(t) p_j \ge E_j D_j M_j, \forall j \tag{2e}$$

$$\xi_j = 1, \forall j : s = s', \forall y_{jsu_j}(t) > 0, \forall z_{js'v_j}(t) > 0 \tag{2f}$$

$$\sum_{j\in[J]} y_{jsu_j}(t) \le W_{su}, \forall s, \forall u, \forall t \tag{2g}$$

$$\sum_{j\in[J]} z_{jsv_j}(t) \le P_{sv}, \forall s, \forall v, \forall t \tag{2h}$$

$$y_{jsu_j}(t) = z_{jsv_j}(t) = 0, \forall j, \forall s, \forall t < a_j + \triangle^{\uparrow}_{js} \tag{2i}$$

$$h_j = \arg\min_{t\in[T]} \{ \sum_{s\in[S]} y_{jsu_j}(t) > 0 \}, \forall j \tag{2j}$$

$$c_j = \arg\max_{t\in[T]} \{ \sum_{s\in[S]} y_{jsu_j}(t) > 0 \}, \forall j \tag{2k}$$

$$y_{jsu_j}(t) \in \{0, 1, 2, ...\}, z_{jsv_j}(t) \in \{0, 1\}, \\ \xi_j \in \{0, 1\}, h_j, c_j \in [T], \forall j, \forall s, \forall u, \forall v, \forall t. \tag{2l}$$

To ensure one data chunk is processed by only one worker, the number of allocated workers is limited to be at most $D_j$ in constraint (2a). Constraint (2b) indicates that each running job has only one PS. The allocated workers and PSs remain unchanged during the training process, as shown in constraint (2c) and (2d). Constraint (2e) guarantees that job $j$ can be completed by allocating sufficient workers. Constraint (2f) reveals that job $j$ can ignore the communication time when its all workers and PSs are deployed together. Constraints (2g) and (2h) are resource capacities. Constraint (2i) implies that the only after a job's arrival can it begin to train. The start time and completion time of job $j$ is calculated by constraint (2j) and constraint (2k).

***Challenges.*** The problem in (2) is a mixed integer nonlinear programming (MINLP). Even in the offline setting, MINLP (2) is NP-hard [21].

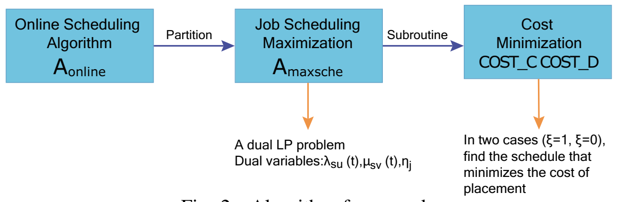
TABLE I
NOTATIONS

| Notation | Description |
|---|---|
| $[X]$ | integer set $\{1, 2, ..., X\}$ |
| $J, S, T$ | # of jobs, physical servers, time slots |
| $a_j, h_j, c_j$ | arrival, start and completion time of job $j$ |
| $E_j, D_j, M_j$ | # of epochs, data chunks and mini-batches for job $j$ |
| $U(V)$ | # of worker (PS) types |
| $W_{su}(P_{sv})$ | # of type-u workers (type-v PSs) on server $s$ |
| $u_j(v_j)$ | the type of workers (PSs) job $j$ specifies |
| $\triangle^{\uparrow}_{js}$ | delay to transmit one data chunk of job $j$ to server $s$ |
| $p_j$ | # of mini-batches trained by one worker of $j$ at a slot |
| $\xi_j$ | whether all workers and PS of $j$ are placed together |
| $y_{jsu_j}(t)$ | # of type-$u_j$ workers serving job $j$ in server $s$ at time $t$ |
| $z_{jsv_j}(t)$ | # of type-$v_j$ PSs serving job $j$ in server $s$ at time $t$ |

## IV. Online Scheduling Framework

### A. Algorithmic Framework

In order to meet the above-mentioned challenges, we design an online scheduling framework, which is divided into two parts, as shown in Fig. 2.



Online Scheduling Algorithm A_online
Partition
Job Scheduling Maximization A_maxsche
Subroutine
Cost Minimization COST_C COST_D
A dual LP problem
Dual variables: λsu (t), μsv (t), ηj
In two cases (ξ=1, ξ=0), find the schedule that minimizes the cost of placement

Fig. 2. Algorithm framework.

- In Sec. IV-B, we introduce the online scheduling framework $A_{online}$, which groups unprocessed ML jobs into multiple batches, thereby converting the total job completion time minimization problem into a series of batch scheduling problems, *i.e.,* maximize the number of scheduled jobs in a batch. Each batch scheduling problem will be solved by the maximum job scheduling algorithm $A_{maxsche}$.
- In Sec. V, we introduce the maximum job scheduling algorithm $A_{maxsche}$. $A_{maxsche}$ employs two subroutines *COST_C* and *COST_D* to select the schedule with the minimal resource placement cost for each job. The subroutines *COST_C* and *COST_D* deal with the problem of minimizing the placement cost when the workers and PS of job $j$ are deployed on the same server or not.

**Algorithm 1** Online Scheduling Framework: $A_{online}$

```
Input: $T, W_{su}, P_{sv}, u_j, v_j, \forall s \in [S], u \in [U], v \in [V]$
Output: $x_{jt}, y_{jsu_j}(t), z_{jsv_j}(t), \forall j \in [J], t \in [T]$
 1: Initialize $\tau_i = 2^{i-1}, x_{jt} = 0, f_{su}(t), g_{sv}(t), y_{jsu_j} = 0, z_{jsv_j} = 0, \forall j \in [J], t \in [T], u \in [U], v \in [V], s \in [S]$
 2: while $i = 1, 2, \dots$ do
 3:     while $t < \tau_i$ do
 4:         $J_i = J_i \cup \{j\}$
 5:     end while
 6:     if $t = \tau_i$ then
 7:         $\{\{x_{jt}\}, \{y_{jsu_j}(t)\}, \{z_{jsv_j}(t)\}, J_i^s\}_{j \in [J_i]} = A_{maxsche}(J_i, \tau_i, u_j, v_j, \{W_{su}\}, \{P_{sv}\})$
 8:         for job $j \in [J_i^s]$ do
 9:             Run job j from time $\tau_i$ to time $\tau_{i+1}$ according to $\{x_{jt}\}, \{u_j\}, \{v_j\}, \{y_{jsu_j}(t)\}, \{z_{jsv_j}(t)\}$
10:         end for
11:         $J_{i+1} = J_{i+1} \cup (J_i \setminus J_i^s)$
12:     end if
13: end while
```

## *B. Online Scheduling Algorithm*

The basic idea of our algorithm is to divide the entire time span at geometrically increasing points, and iteratively schedule ML jobs that have arrived but not processed until a certain point. Let $\tau_i$ denote the time point that divides the entire time horizon, $\tau_0 = 1, \tau_i = 2^{i-1}$. The time span between two neighboring time points is defined as a time interval, such as interval $i$'s time span is $(\tau_{i-1}, \tau_i]$ (note that interval 1 is $[\tau_0, \tau_1]$). Let $J_i$ denote the set of jobs that have arrived before time $\tau_i$ but have not been completed.

***Algorithm Details.*** In $A_{online}$, line 1 initializes the primal variables and $J_i^s$. When $\tau_0 = 1$, no job can be completed in the first time interval. Lines 3-5 group jobs that arrived before time point $\tau_i$ but have not been completed into the set $J_i$. In line 7, this algorithm calls the batch processing algorithm $A_{maxsche}$. According to the optimal scheduling result generated by $A_{maxsche}$ in lines 8-9, all jobs in the set $J_i^s$ are run in $\tau_i - \tau_{i+1}$. In the line 11, the algorithm adds these jobs that have arrived but not been scheduled in the round $i$ to set $J_{i+1}$, so that they can be processed in the round $i+1$ or subsequent rounds.

## V. Batch Scheduling Algorithm Design

### *A. The Maximum Job Scheduling Problem*

In the online scheduling algorithm, the problem of minimizing the total completion time is transformed into maximizing job scheduling in each round $i$, *i.e.,* complete as many jobs as possible between time $\tau_i$ to $\tau_{i+1}$. Let $x_j$ denote whether accept job $j$ at the current round.

$$\text{maximize} \sum_{j \in [J_i]} x_j \tag{3}$$

subject to:

$$x_j \in \{0,1\}, \forall j \in [J_i] \tag{3a}$$

$$(2a) - (2l), y_{jsu_j}(t) > 0 \; and \; z_{jsv_j}(t) > 0 \text{ iff } \; x_j > 0, \forall j \in [J_i], \forall t \in (\tau_i, \tau_{i+1}] \tag{3b}$$

This maximization problem involves integrality variables and nonlinear constraints. In order to solve these challenges, we first apply the compact-exponential technique [22] to reformulate the problem (3) into an equivalent integer linear program (ILP) with a packing structure:

$$\text{maximize} \sum_{j \in [J_i]} \sum_{l \in [L_j]} \chi_j^l \tag{4}$$

subject to:

$$\sum_{j \in [J_i]} \sum_{l \in [L_j]} \gamma_{jsu_j}^l(t) \leqslant W_{su}, \forall s, \forall u, \forall t \in (\tau_i, \tau_{i+1}] \tag{4a}$$

$$\sum_{j \in [J_i]} \sum_{l \in [L_j]} \delta_{jsv_j}^l(t) \leqslant P_{sv}, \forall s, \forall v, \forall t \in (\tau_i, \tau_{i+1}] \tag{4b}$$

$$\sum_{l \in [L_j]} \chi_j^l \leqslant 1, \forall j \in [J_i] \tag{4c}$$

$$\chi_j^l \in \{0,1\}, \forall j \in [J_i], \forall l \in [L_j] \tag{4d}$$

In ILP (4), $L_j$ represents a set of feasible schedules (*i.e.,* satisfy all constraints) for job $j$. For a feasible schedule $l \in [L_j]$, $\gamma_{jsu_j}^l(t)$ and $\delta_{jsv_j}^l(t)$ represent assigned workers and PSs at each time $t$, respectively. The binary variable $\chi_j^l$ represents whether to accept job $j$ according to the schedule $l$ or not.

Constraints (4a) and (4b) are equivalent to constraint (2g) and (2h), respectively. Constraint (4c) guarantees that each job is executed according to at most one schedule. The feasible solutions of ILP (4) is exactly that of problem (3), and their objective values are the same. However, such reformulation introduces exponential variables. To tackle this problem, we relax $\chi_j^l \in \{0,1\}$ to $\chi_j^l \geq 0$ and introduce dual variables $\lambda_{su}(t), \mu_{sv}(t)$ and $\eta_j$ to constrains (4a), (4b) and (4c), and formulate the dual problem of the ILP (4) as follows:

$$\text{minimize} \sum_{j \in [J_i]} \eta_j + \sum_{s \in [S]} \sum_{u \in [U]} \sum_{t \in (\tau_i, \tau_{i+1}]} \lambda_{su}(t) W_{su} + \sum_{s \in [S]} \sum_{v \in [V]} \sum_{t \in (\tau_i, \tau_{i+1}]} \mu_{sv}(t) P_{sv} \tag{5}$$

subject to:

$$\eta_j \geqslant 1 - \sum_{s \in [S]} \sum_{t \in (\tau_i, \tau_{i+1}]} \lambda_{su_j}(t) \gamma_{jsu_j}^l(t) - \sum_{s \in [S]} \sum_{t \in (\tau_i, \tau_{i+1}]} \mu_{sv_j}(t) \delta_{jsv_j}^l(t), \forall j \in [J_i], \forall l \in [L_j] \tag{5a}$$

$$\eta_j, \lambda_{su_j}(t), \mu_{sv_j}(t) \geqslant 0, \forall j \in [J_i], \forall s, \forall u, \forall v, \forall t \in (\tau_i, \tau_{i+1}] \tag{5b}$$

The dual variables $\lambda_{su}(t)$ and $\mu_{sv}(t)$ can be interpreted as the unit cost of type-$u$ worker and type-$v$ PS on server $s$ at time $t$, respectively. As a result, $\sum_{s \in [S]} \sum_{t \in [T]} \lambda_{su_j}(t) \gamma_{jsu_j}^l(t) + \sum_{s \in [S]} \sum_{t \in [T]} \mu_{sv_j}(t) \delta_{jsv_j}^l(t)$ represents the total resource cost of the allocated workers and PSs placed according to schedule $l$ of job $j$, the right hand side (RHS) of (5a) is the weight of the job (assuming that the weights of all jobs are the same and are set to 1) minus the resource cost of job $j$, that is, the utility

**Algorithm 2** Maximum Job Scheduling Algorithm: $A_{maxsche}$

```
Input: $J_i, \tau_i, c_j, u_j, v_j, f_{su}(t), g_{sv}(t), W_{su}, P_{sv}, U, V, \forall s \in [S], t \in [T]$
Output: $y_{jsu_j}(t), z_{jsv_j}(t), J_i^s, \forall j \in [J], t \in [T]$
1: Initialize $y_{jsu_j} = 0, z_{jsv_j} = 0, \lambda_{su}(t) = \lambda_{su}(0), \mu_{sv}(t) = \mu_{sv}(0), \forall j \in [J], t \in [T], u \in [U], v \in [V], s \in [S]$
2: for job $j \in [J_i]$ do
3:   for $c_j = \tau_i$ to $\tau_{i+1}$ do
4:     for $D_u = 1$ to $D_j$ do
5:       Set $(cost, l)$ =COST_C$(c_j, u_j, v_j, D_u, \tau_i)$
6:       Set $(cost\_d, l')$ =COST_D$(c_j, u_j, v_j, D_u, \tau_i)$
7:       if $cost\_d < cost$ then
8:         Set $(min\_cost, l) = (cost\_d, l')$
9:       end if
10:      Update $\eta_{jl} = 1 - min\_cost$
11:      if $\eta_{jl} > \eta_j$ then
12:        $l^* = l, \eta_j = \eta_{jl}$
13:      end if
14:    end for
15:    if $\eta_j > 0$ then
16:      Update $J_i^s = J_i^s \bigcup \{j\}$
17:      Set $\{y_{jsu_j}(t), z_{jsv_j}(t)\}_{\forall s, \forall u, \forall v, \forall t}$ according to $l$
18:      Update $f_{su_j}(t) = f_{su_j}(t) + y_{jsu_j}(t), g_{sv_j}(t) = g_{sv_j}(t) + z_{jsv_j}(t), \lambda_{su_j}(t) = \lambda_{su_j}(f_{su_j}(t)), \mu_{sv_j}(t) = \mu_{sv_j}(g_{sv_j}(t))$
19:    end if
20:  end for
21: end for
```

of the job. To comply with the theorem of complementary slackness, we set the dual variable $\eta_j$ to the maximum value between 0 and RHS of (5a) with the optimal schedule $l^*$:

$$\eta_j = \max\{0, \max_{l \in [L_j]} RHS\ of\ (5a)\}. \tag{6}$$

If job utility $\eta_j > 0$, we process job $j$ according to $l^*$, otherwise, we postpone it into the next time interval for scheduling.

***Algorithm Details.*** In $A_{maxsche}$(Algorithm 2), the first line initializes the primal and the dual variables. The algorithm traverses each job $j \in [J_i]$, the time to complete the job $t \in (\tau_i, \tau_{i+1}]$, and the number of deployed workers $D_u \in [1, D_j]$ (Lines 2-4). It calls the subroutines $COST\_C$ and $COST\_D$ in lines 5 and 6, and finds the schedule with the lowest placement cost in the two cases ($\xi = 1, \xi = 0$). In lines 7-13, by comparing the solutions obtained in the two cases, the optimal scheduling of job $j$ is obtained, and its utility $\eta_j$ is the highest at this time. If $\eta_j > 0$ is obtained, then all the primal and dual variables are updated on lines 16-18, and $J_i^s$ is updated on line 16, which is a set of jobs scheduled in the round $i$. In line 18, $f_{su}(t)$ represents the number of type-$u$ workers allocated on server $s$ at time $t$, and $g_{sv}(t)$ represents the number of type-$v$ PSs allocated on server $s$ at time $t$, the dual variable $\lambda_{su}(t)$ represents the unit resource price of a type-$u$ worker on the server $s$ at time $t$, which is a function of $f_{su}(t)$, and the dual

**Algorithm 3** Cost Under Centralized Placement: $COST\_C$

```
Input: $c_i, u_j, v_j, D_u, \tau_i$
Output: $cost, l$
1: Initialize $cost = \infty, l = \emptyset, y_{jsu_j}(t) = 0, z_{jsv_j}(t) = 0, \forall s$
2: Compute $d_j = \frac{E_j D_j M_j}{p_j Du}, c_j = \tau_i + d_j$
3: Sort servers in [S']=$\{s | a_j + \triangle_{js}^{\uparrow} < \tau_i \wedge min_{t \in (\tau_i, c_j]}\{W_{su_j} - f_{su_j}(t) \geq D_u \wedge min_{t \in (\tau_i, c_j]}\{P_{sv_j} - g_{sv_j}(t) \geq 1\}\}$ according to $\lambda_{su_j}(\tau_i) D_u + \mu_{sv_j}(\tau_i)$ in non-decreasing order into $s_1, s_2, \ldots, s_S$
4: if $S' == 0$ then
5:   return $\infty, l$
6: else
7:   set $y_{js_1u_j}(t) = D_u, z_{js_1v_j}(t) = 1, \forall t \in (\tau_i, c_j]$
8:   set $cost = \sum_{t \in (\tau_i, c_j]}\{\lambda_{s_1u_j}(t) D_u + \mu_{s_1v_j}(t)\}, l \leftarrow \{y, z\}$
9:   return $cost, l$
10: end if
```

variable $\mu_{sv}(t)$ represents the unit resource price of the type-$v$ PS on the server $s$ at time $t$, which is a function of $g_{sv}(t)$.

***Price Function.*** To implement the above admission rule, there are two sub-problems need to be solved, *i.e.*, determine the prices and the schedule with the minimum cost. To this end, we first design the following two sets of price functions to price workers and PSs, respectively.

$$\lambda_{su}(f_{su}(t)) = \theta_1^{\frac{f_{su}(t)}{W_{su}}} - 1, \forall s \in [S], u \in [U], t \in [\tau_i], \quad where\ \theta_1 = 2(TSUF_1) + 1. \tag{7}$$

$$\mu_{sv}(g_{sv}(t)) = \theta_2^{\frac{g_{sv}(t)}{P_{sv}}} - 1, \forall s \in [S], v \in [V], t \in [\tau_i], \quad where\ \theta_2 = 2(TSVF_2) + 1. \tag{8}$$

where $f_{su}(t), g_{sv}(t)$ denote the total number of workers and PSs consumed, respectively. $F_1, F_2$ are the upper bound of the unit price of per worker and PS, respectively, which can be estimated according to the historical experiments. Next we discuss how to determine the schedule with the minimum cost in Sec. V-B.

## *B. Cost Minimization Problem*

Since the weight of job $j$ is 1, the problem of maximizing the utility of job $j$ is equivalent to the problem of minimizing its scheduling cost:

$$\min \sum_s \sum_{t \in (\tau_i, \tau_{i+1}]} \lambda_{su_j}(t) y_{jsu_j}(t) + \sum_s \sum_{t \in (\tau_i, \tau_{i+1}]} \mu_{sv_j}(t) z_{jsv_j}(t) \tag{9}$$

subject to:

$$y_{jsu_j}(t) \leq W_{su_j} - f_{su_j}(t), \forall s, \forall t \in (\tau_i, \tau_{i+1}] \tag{9a}$$

$$z_{jsv_j}(t) \leq P_{sv_j} - g_{sv_j}(t), \forall s, \forall t \in (\tau_i, \tau_{i+1}] \tag{9b}$$

$$(2a) - (2f), (2i) - (2l), \forall j \in [J_i], \forall t \in (\tau_i, \tau_{i+1}] \tag{9c}$$

For job $j$, we propose two algorithms $COST\_C$ and $COST\_D$ to jointly find the optimal schedule with minimum cost. In algorithm $COST\_C$, all the workers and PSs of job

$j$ are placed on the same server. On the contrary, the workers and PSs of job $j$ are distributively placed in $COST\_D$.

***Algorithm Details.*** The $COST\_C$ shown in Algorithm 3 returns the minimal placement cost and the optimal schedule $l^*$ at this time. Line 2 first calculates the training time according to the number of workers and the number of data chunks for job $j$. Line 3 sorts the candidate servers that can accommodate all the needed workers and PSs of job $j$ in the non-decreasing order of total cost. Lines 4-9 determine whether there is a qualified schedule. If it exists ($S' \neq 0$), the first server $[S']$ is the target one, as a result, the decision variables and placement cost are updated according to the corresponding optimal schedule.

In $COST\_D$ (Algorithm 4), line 2 is executed for the same reason of line 2 in $COST\_C$. Line 3 lists the constraints that the start time cannot be earlier than the arrival time, and all possible selected servers are sorted in non-descending order according to the price of the workers. Lines 4-7 start with the lowest price server and deploy as many workers as possible. Lines 12-38 divide the servers into two sets according to the last used server $[s_1, s_{h-1}]$ and $[s_z, S]$. Only one server is required to deploy the PS, we analyze the two situations to obtain the solution with the least cost of resources, update the decision variables and the placement cost.

## VI. Performance Evaluation

***Settings.*** We use Microsoft's Philly Tracking [23], [24] for trace-driven simulations. It contains information on 117,325 DNN jobs submitted between August 7, 2017 and December 22, 2017, as well as information on approximately 550 machines in more than 14 virtual clusters. Based on these data, we simulate an edge cloud system consisting of 100-300 edge servers and a cloud. The default number of edge servers is 200, and one time slot is one hour. Based on the trace, this paper sets the number of GPUs and CPUs as that of workers and PSs on each edge server, respectively. For other unknown information, we use the reference [1] [7] [8] [6] to construct configurations on each worker, PS, and ML jobs as follows. $U$ and $V$ are set to be within $[8, 10]$, respectively. Then we randomly select the types of each worker and PS for servers. The bandwidth of workers is set within $[100, 5*1024]$ Mbps, and the bandwidth of PSs is set within $[5, 20]$ Gbps. ML jobs are configured as below: $E_j \in [50, 100], D_j \in [20, 50], M_j \in [10, 50]$, $x_j \in [1, T/1.5], n_j \in [0.001, 0.05]$ hour, $H_j \in [10, 100]$ milliseconds,$k_j \in [30, 575]$ MB and $\triangle_{js}^{\uparrow}$ are within $[1, 4]$ and $[10, 40]$ time slots for edge servers and cloud, respectively [25].

***Comparison Algorithms and Performance Indicators.*** We use simulations to verify $A_{online}$, and compare it with the following four most advanced and representative algorithms:

- FIFO [26]: The default scheduler in Hadoop [27] and Spark [28]. Jobs are scheduled in the order of arrival, and are dispatched to the server that can schedule the job earliest.
- DRF [7]: The default scheduler in YARN [29] and Mesos [30]. The maximum-minimum fairness is achieved by counting the number of workers and PSs.

**Algorithm 4** Cost Under Distributed Placement: $COST\_D$

```
Input: $c_i, u_j, v_j, D_u, \tau_i$
Output: $cost, l$
 1: Initialize $min\_cost = \infty, l = \emptyset, L = \emptyset, y_{jsu_j}(t) = 0, z_{jsv_j}(t) = 0, cost = 0, cost_u = 0, \forall s$
 2: Compute $d_j = \frac{E_j D_j M_j}{p_j Du}, c_j = \tau_i + d_j, D'_u = D_u$
 3: Sort servers in [s]=$\{s | a_j + \triangle_{js}^{\uparrow} < \tau_i\}$ according to $\lambda_{su_j}(t)$ in non-decreasing order into $s_1, s_2, \ldots, s_S$
 4: for $i = 1$ to $S$ do
 5:     set $y_{js_iu_j}(t) = \min\{\min_{t \in [\tau_i, c_j]}\{W_{s_iu_j} - fs_iu_j(t)\}, D_u - \sum_{i'}^{i-1} yjs_{i'}u_j(t), D'_u\}, \forall t \in (\tau_i, c_j]$
 6:     Update $D'_u = D'_u - y_{js_iu_j}(\tau_i)$
 7: end for
 8: if $D'_u > 0$ then
 9:     return $\infty, l$
10: end if
11: Compute $cost_u = \sum_{t \in (\tau_i, c_j]}\{\sum_s \lambda_{su_j}(t) y_{jsu_j}(t)\}$
12: record the last used server as $s_{s_z}$
13: for $i = 1$ to $s_z - 1$ do
14:     if $\min_{t \in (\tau_i, c_j]}\{P_{s_iv_j} - g_{s_iv_j}(t)\} \geq 1$ then
15:         set $z'_{js_1v_j} = 1, \forall t \in (\tau_i, c_j], cost = cost_u + \sum_{t \in (\tau_i, c_j]}\{\sum_s \mu_{s_iv_j}(t)\}, L = L \cup \{cost, 1 \leftarrow \{y, z'\}\}$
16:     end if
17: end for
18: for $i_1 = s_z$ to $S$ do
19:     set $D_u(s_{i_1}) = \min_{t \in (\tau_i, c_j]}\{W_{s_{i_1}u_j} - f_{s_{i_1}u_j}(t) - y_{js_{i_1}u_j}(t)\}, D_v(s_{i_1}) = \min_{t \in (\tau_i, c_j]}\{P_{s_{i_1}v_j} - g_{s_{i_1}v_j}(t)\}, y' = y, z' = \emptyset$
20:     Compute $cost = cost_u + \sum_{t \in (\tau_i, c_j]} \mu_{s_{i_1}v}(t)$
21:     for $i_2 = s_z$ to $s_1$ do
22:         if $\lambda_{s_{i_1}u_j}(\tau_i) - \lambda_{s_{i_2}u_j}(\tau_i) - \mu_{s_{i_1}v_j}(\tau_i) \leqslant 0 \wedge D_u(s_{i_1}) > 0$ then
23:             set $\Delta W = \min\{y'_{js_{i_2}u_j}(\tau_i), D_u(s_{i_1})\}, D_u(s_{i_1}) = D_u(s_{i_1}) - \Delta W$
24:             Compute $cost = cost + (\lambda_{s_{i_1}u_j}(\tau_i) - \lambda_{s_{i_2}u_j}(\tau_i) - \mu_{s_{i_1}v_j}(\tau_i))\Delta W$
25:             set $y'_{js_{i_2}u_j}(t) = y'_{js_{i_2}u_j}(t) - \Delta W, y'_{js_{i_1}u_j}(t) = y'_{js_{i_1}u_j}(t) + \Delta W, z'_{js_{i_1}v_j}(t) = 1, \forall t \in [\tau_i, c_j]$
26:         else
27:             if $D_v(s_{i_1}) \geq 1$ then
28:                 $L = L \cup \{cost, l \leftarrow \{y, z\}\}$
29:             end if
30:             break
31:         end if
32:     end for
33: end for
34: if $L = \emptyset$ then
35:     return $\infty, l$
36: end if
37: Sort $L$ according to $cost$ in non-decreasing order and record the first element as $L_1$
38: return $L_1$
```

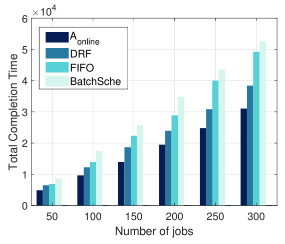


Fig. 3. Total completion time with different $J$.

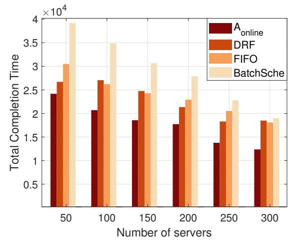


Fig. 4. Total completion time with different $S$.

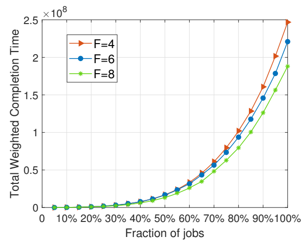


Fig. 5. Total weighted completion time under different $F$.

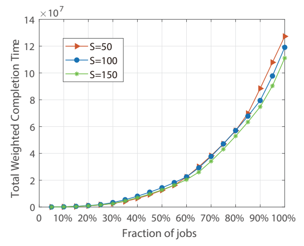


Fig. 6. Total weighted completion time under different $S$.

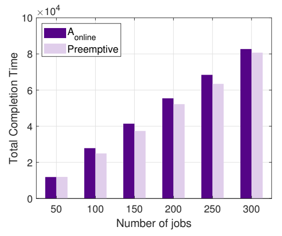


Fig. 7. Total completion time with different $J$.

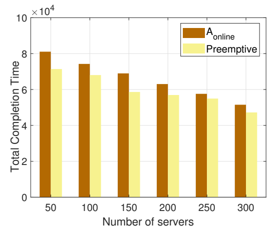


Fig. 8. Total completion time with different $S$.

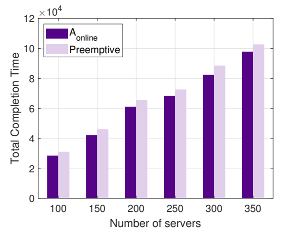


Fig. 9. Total completion time with different $J$

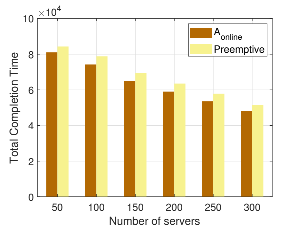


Fig. 10. Total completion time with different $S$

- BatchSche [8]: The algorithm proposed for the elastic resource requirements of ML jobs, mainly applies batch processing to minimize the total cost of resources.
- Preemptive [31]: The preemptive scheduling algorithm let jobs with a shorter completion time preempt other jobs on the server, so that they can be completed first, to reduce the total completion time. It ignores the time overhead of preemption.

***Results.*** First, we compare the total completion time of our algorithm with the three representative algorithms under the settings of different $(J)$ and $(S)$. As shown in Fig. 3, the total completion time grows with the increase in the number of jobs. This is because the increase in $J$ will bring larger workload. In Fig. 4, as the number of edge servers increases, the total completion time decreases because of the increase in $S$ bringing richer computing power. We can conclude that the performance of $A_{online}$ is obviously better than $DRF$, $FIFO$ and $BatchSche$.

Second, we study whether different resource price caps $F$ (*i.e.,* $\max\{F_1, F_2\}$) and different $S$ will affect the number of jobs received by the algorithm in each time slot. Fig. 5 shows the total weighted completion time of $A_{online}$ under different $F$. The fixed total number of jobs is 200, and the total job weight is also unchanged. It can be seen that the larger the $F$, the larger the objective value. This is because cost is related to $F$, and accepting the job will increase the unit price of related resources, which means that accepting the job needs to meet the value of $F$ when the unit resource price increases. Therefore, the larger the value of $F$, the more jobs that can be served in the same time, so that the total weighted completion time for completing the same amount of jobs will be smaller. Fig. 6 shows the total weighted completion time of $A_{online}$ under different $S$. It can be seen that the larger the value of $S$, the smaller the total weighted completion time. This is because as the number of servers increases, the total amount of resources will also increase, and more jobs can be served in the same time, making the total weighted completion time smaller.

Third, we compare our algorithm with $Preemptive$ under different settings and scenarios. Fig. 7-8 and Fig. 9-10 are plotted under different $(J)$ and $(S)$ when considering the preemption cost or not, respectively. In Fig. 7-8, $Preemptive$ performs relatively better, because $Preemptive$ does not consider the upload delay and the preemption overhead. In Fig. 9-10, the performance of $A_{online}$ is slightly better than $Preemptive$ considering preemption overhead. Fig. 11 shows the comparison of $A_{online}$ and $Preemptive$ under different parameter sizes $k_j$. As $k_j$ increases, the efficiency of $Preemptive$ is significantly better than $A_{online}$. Because the increase of $k_j$ makes $p_j$ small when $\xi = 0$, so jobs tend to be deployed on the same server with minimal preemption cost. In addition, non-preemptive scheduling may generate a lot of resource fragments. The advantage of $Preemptive$ is that the fragments can be eliminated by preempting some workers of other jobs, as a result, the total completion time is reduced. Fig. 12 and Fig. 13 demonstrate resource utilization of $A_{online}$ and $Preemptive$. We observe that with the increasing of the number of jobs, resource fragmentation are increased in $A_{online}$. As the number of edge servers increases, this situation aggravates and thus harms the resource utilization. Preemption is an effective method to eliminate fragmentation, therefore $Preemptive$ performs better in such case.

At last, Fig. 14 shows the performance ratio of $A_{online}$. The offline optimal algorithm knows all the information of jobs before the start of the scheduling. The performance ratio is the ratio of the total job completion time generated by $A_{onliine}$ to that of the offline optimal algorithm. It can be seen from the figure that when the number of jobs increases, the performance

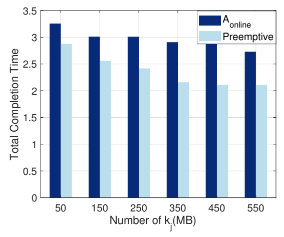


Fig. 11. Total completion time with different $k_j$.

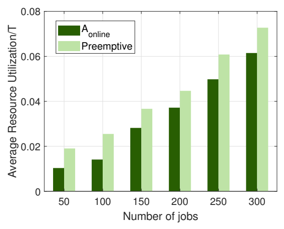


Fig. 12. Average resource utilization of $A_{online}$ with different $J$.

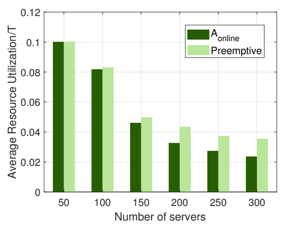


Fig. 13. Average resource utilization of $A_{online}$ with $S$.

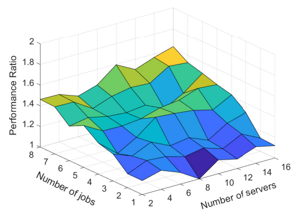


Fig. 14. Performance ratio.

ratio of $A_{online}$ shows an increasing trend. When the number of edge servers increases, the performance ratio also shows an increasing trend. The overall performance ratio of $A_{online}$ is close to 1, and its performance is relatively good.

## VII. CONCLUSION

In this paper, we focus on the design of scheduling algorithm for distributed ML jobs in edge-cloud networks. We propose an online algorithm to schedule distributed ML jobs. The online algorithm aims at minimizing the total completion time of all jobs. We first transform the online scheduling problem into a series of batch processing problems. For each batch processing problem, we adopt the compact-exponential technique and dual theory to reformulate it, and design a maximum job scheduling algorithm to solve it. At last, we design two greedy algorithms to jointly consider the job completion time and the utility of jobs in order to find the best schedule for each job. Large-scale simulations have proved that our algorithm has good performance and resource utilization compared with the benchmark algorithms.

## REFERENCES

[1] 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.

[2] 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.

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

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

[5] Y. Sun, S. Zhou, and D. Gündüz, "Energy-aware analog aggregation for federated learning with redundant data," in *Proc. of ICC*, 2020.

[6] 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.

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

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

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

[10] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: geo-distributed machine learning approaching lan speeds," in *Proc. of NSDI*, 2017.

[11] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: on-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things," *Network IEEE*, vol. 33, no. 5, pp. 96–103, 2019.

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

[13] H. Wang, Z. Liu, and H. Shen, "Job scheduling for large-scale machine learning clusters," in *Proc. of CoNEXT*, 2020.

[14] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. of AISTATS*, 2017.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

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

[25] S. Chen, L. Jiao, L. Wang, and F. Liu, "An online market mechanism for edge emergency demand response via cloudlet control," in *Proc. of IEEE INFOCOM*, 2019.

[26] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. of USENIX HotCloud*, 2010.

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

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

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

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

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