Online Scheduling of Heterogeneous Distributed Machine Learning Jobs

Qin Zhang[1], Ruiting Zhou[1,2,*], Chuan Wu[3], Lei Jiao[4], Zongpeng Li[1]
[1]Wuhan University, China [2]The Chinese University of Hong Kong, Hong Kong [*]Corresponding author
[3]The University of Hong Kong, Hong Kong [4]University of Oregon, USA

ABSTRACT

Distributed machine learning (ML) has played a key role in today's proliferation of AI services. A typical model of distributed ML is to partition training datasets over multiple worker nodes to update model parameters in parallel, adopting a *parameter server* architecture. ML training jobs are typically resource elastic, completed using various time lengths with different resource configurations. A fundamental problem in a distributed ML cluster is how to explore the demand elasticity of ML jobs and schedule them with different resource configurations, such that the utilization of resources is maximized and average job completion time is minimized. To address it, we propose an online scheduling algorithm to decide the execution time window, the number and the type of concurrent workers and parameter servers for each job upon its arrival, with a goal of minimizing the weighted average completion time. Our online algorithm consists of (i) an online scheduling framework that groups unprocessed ML training jobs into a batch iteratively, and (ii) a batch scheduling algorithm that configures each ML job to maximize the total weight of scheduled jobs in the current iteration. Our online algorithm guarantees a good parameterized competitive ratio with polynomial time complexity. Extensive evaluations using real-world data demonstrate that it outperforms state-of-the-art schedulers in today's AI cloud systems.

ACM Reference Format:
Qin Zhang[1], Ruiting Zhou[1,2,*], Chuan Wu[3], Lei Jiao[4], Zongpeng Li[1]. 2020. Online Scheduling of Heterogeneous Distributed Machine Learning Jobs. In *International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (Mobihoc '20), October 11–14, 2020, Boston, MA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397166.3409128

1 INTRODUCTION

Nowadays, most leading IT companies operate distributed machine learning (ML) clusters of GPU servers, to run ML jobs that train models over large datasets for providing AI-driven services. To train a large model, hundreds of concurrent workers (typically implemented on virtual machines or containers) are deployed in parallel. Either the training dataset or the ML model is partitioned among workers, realizing *data parallelism* or *model parallelism* [11][12][23]. In model parallelism, each worker updates part of the parameters using the entire input dataset [8]. In data parallelism, each worker has an entire copy of the ML model and computes parameter update (gradients) using a portion of input data; in each training iteration, workers exchange locally-computed gradients to obtain the global ML model update. As training data is usually enormous, data parallelism is the dominant form of parallel training in practice [11][23].

A typical approach for exchanging parameter updates among workers is through the parameter server (PS) framework [12][23]. In the PS framework, one or multiple PSs maintain model parameters as a global key-value store, and each worker uploads computed gradients to the PSs. The PSs update the corresponding parameters based on received gradients and then send updated parameters to the workers. The workers and PSs may be placed on different physical servers, when they cannot be completely accommodated on the same server, or to fully utilize expensive and fragment resources on servers [8].

ML training jobs are resource-intensive and time-consuming. Existing distributed ML systems [16][18][27] require job owners to estimate the amount of resources, including the number of workers and the resource configuration of each worker, as well as the time needed, to train the ML model using a large dataset. For example, Google uses Borg [28] and Microsoft, Tencent, and Baidu use customized versions of YARN schedulers [27] to aggressively provision each job as much resource as possible according to user demand and job priority, using strategies such as FIFO and max-min fair allocations.

However, the job owner is often uncertain of the amount of resources and time it may take to complete a job. There is *elasticity* in ML jobs' resource demand: It takes different amounts of time to train a certain model with workers of different resource configurations, especially of different numbers of GPUs. Further, the processing time of a mini-batch is typically not inversely proportional to the worker's resource allocation, which is mainly due to overhead in parallel training [14]. Next, assigning training jobs less resources than what they require in the ideal case (*i.e.,* that leads to most expedited single-job training [14] [32] [25]) may reduce average training completion time in the entire system. For example, when training CIFAR-10 CNN for 100K steps until the model achieves 87% accuracy, the single-step training time (time to train a mini-batch) can be 15 milliseconds with a single GPU and 10 milliseconds with two GPUs (suppose it is the ideal case) [14]. Thus, if there are two training jobs of this type submitted at the same time and only three GPUs are available, with adequate other resources, allocating one GPU to one job and two GPUs for the other is the best strategy for minimizing the average job completion time, which results in $(10 + 15)/2 = 12.5$ milliseconds, in contrast

Mobihoc '20, October 11–14, 2020, Boston, MA, USA

ACM ISBN 978-1-4503-8015-7/20/10…$15.00
https://doi.org/10.1145/3397166.3409128

to allocating two GPUs to each job sequentially, which results in $(10 + 20)/2 = 15$ milliseconds.

Considering demand elasticity, a fundamental problem for a ML cluster operator is: *Given limited resources, how to decide the number/type of workers and PSs and running time of each job, such that resources are maximally utilized and average weighted completion time is minimized?* Here, the weight of each job may characterize its processing priority.

To address the above problem, we first formulate the average weighted completion time minimization problem into a time-indexed mathematical program. The program formulates features of ML jobs (demand for large-volume data analysis capacity and high inter-node connection bandwidth). Different from traditional makespan minimization problems, it contains both conventional (packing-type) constraints and non-conventional (set-type and natural language described) constraints, which cannot be handled by existing approaches [24] [17]. Decision variables include the number/type of workers and PSs, and the execution window of each job. To compute schedules on the go with the shortest completion time, we divide our design into two steps:

First, we propose an online framework to convert the online optimization problem into a series of batch scheduling problems by partitioning the overall timespan into intervals with geometrically increasing length. Our online scheduling framework employs a *dual approximation algorithm* as a subroutine for performance guarantee. The dual approximation algorithm finds an infeasible solution that is super-optimal, where the performance of the algorithm is measured by the degree of infeasibility allowed. The infeasible solution will finally become feasible as job execution can span multiple intervals. The super-optimal objective value contributes to bound the average weighted completion time. This dual algorithm is realized through a batch scheduling algorithm that solves the *maximum weighted schedule problem* to schedule as many unscheduled jobs as possible before a certain time point.

Second, we observe that the maximum weighted schedule problem includes several non-conventional constraints for characterizing the configuration/placement of workers and PSs. To handle these *set-type* and *natural language described* constraints, we encode each valid schedule in a variable and reformulate the original program into an integer linear program (ILP), where only conventional packing constraints are included, at the price of introducing an exponential number of variables. Instead of solving the ILP directly, which is infeasible in practice due to time complexity, we design an approximation algorithm by applying a tailored primal-dual framework to the ILP's LP relaxation and its dual LP. We interpret dual variables as unit resource prices, and compute the best schedule for each job based on resource consumption cost and its ML framework. The algorithm schedules a job if its weight is higher than its estimated serving cost.

We carry out rigorous theoretical analysis to prove that our online algorithm runs in polynomial time, and achieves a bounded competitive ratio. We evaluate practical effectiveness of our online algorithm through trace-driven simulation studies. We implement four representative job scheduling strategies used in existing cloud platforms, and compare them with our algorithm. Simulation results confirm that our algorithm outperforms existing methods by up to 200% in average weighted completion time, especially in systems with resource shortage.

2 RELATED WORK

Resource Allocation in Distributed ML Systems. Borg [28] is a large-scale cluster manager from Google that runs jobs in a priority-based approach with preemption. Ghodsi *et al.* [16] propose a fair allocation policy of multiple resource types, similar to Mesos [18] and YARN [27]. In these systems, job owner prescribes the number and resource configuration of workers. In comparison, we design an online algorithm to guide worker deployment and resource allocation, exploiting the demand elasticity of ML jobs. Gao *et al.* [14] solve a training time minimization problem to find the best device placement of a deep neural network, using a reinforcement learning algorithm. Bao *et al.* [7] propose a deep learning-based job placement algorithm to minimize interference among co-located ML jobs. Resource allocation among multiple jobs is not considered by these work. Chen *et al.* [9] identify the demand elasticity of data analytics jobs and propose a performance-aware fair scheduler, which is designed for the offline instead of the online scenario.

Amiri *et al.* [5] propose a centralized scheduling strategy that assigns tasks to workers to minimize the average completion time with the help of one master. Zou *et al.* [37] develop a procedure to help users better choose the mini-batch size and the number of PSs. Similarly, Yan *et al.* [30] develop performance models that quantify the impact of data partitioning and system provisioning on system performance and scalability. Above papers don't consider online job scheduling and resource sharing problems. Bao *et al.* [8] design an online algorithm to guide resource allocation over time in a distributed machine learning system. Although we consider a similar problem, this work is significantly different from [8]. First, our work is the first that explores the demand elasticity. A job's scheduling and configuration are needed to be determined, while [8] focuses on adjusting the number of customized workers in each time slot, but does not address choices of different types of workers/PSs for a job, nor colocation of workers and PSs on the same physical server(s). Second, considering the demand elasticity of ML jobs, the goal of our work is to minimize the weighted completion time, while [8] aims to maximize the overall utility. Third, with the different optimization objective, our algorithmic idea to solve the weighted completion time minimization problem is also different from [8], as shown in Fig. 1.

Job Scheduling and Resource Allocation in Cloud Systems. Shi *et al.* [26] propose the first online combinatorial auction for cloud resource allocation and pricing. Chowdhury *et al.* [13] design an allocation algorithm to achieve multi-resource fairness for elastic and correlated demands. Zhang *et al.* [34] study online resource allocation in a cloud computing platform through posted-price mechanisms. Zhang *et al.* [33] design mechanisms for online cloud resource bundling and provisioning to maximize social welfare with server costs. Jiao *et al.* [21, 22] devise online prediction-free and prediction-aware algorithms to provision resources across clouds and edges for serving dynamic demands. These studies satisfy each job's demand within a fixed window, and do not consider the demand elasticity and scheduling dimensions in the solution space.

For job scheduling, Azar *et al.* [6] study online cloud job scheduling problems for deadline-sensitive jobs, assuming that one server can only execute one job in each time slot. Zheng *et al.* [35] investigate cloud brokerage service and study economic issues based on a stochastic job scheduling problem. Zhou *et al.* [36] design a mechanism for online cloud job scheduling and resource allocation, where jobs have alternative deadlines corresponding to different job valuations. Wang *et al.* [29] schedule jobs online via creating and running multiple replicas of each task in order to mitigate the straggler issue. The resource demand of each job is specified by the job owner in advance in the above literatures.

3 SYSTEM MODEL

3.1 System Overview

We consider a machine learning cluster where multiple ML training jobs run using potentially different ML frameworks (*e.g.*, TensorFlow [4], MXNet [11], CNTK [2]).

Table 1: List of Notations

J	# of jobs	R	# of resource types
T	system timespan	$[X]$	interger set $\{1, 2, \dots, X\}$
a_j	arrival time of j	D_j	# of data chunks in j
w_j	weight of job j	d_j	running duration of j
M	# of worker types	P	# of PS types
E_j	# of training epochs for job j		
K_j	# of mini-batches in one data chunk of job j		
H	# of servers to deploy workers and PSs		
C_h^r	capacity of type-r resource on server h		
$e_m^r(z_p^r)$	type-r resource of worker m (PS p)		
$b_m(B_p)$	bandwidth of worker m (PS p)		
v_{jm}	time to train a mini-batch of job j in worker m		
π_j	size of gradients generated by each worker after processing one mini-batch when serve job j		
U_j^p	time to update parameters at a type-p PS in each iteration of j		
ρ_{jm}^p	processing capacity of each worker when j employs worker m and PS p		
q_j	whether j's all workers (and PSs) are running in one server or not		
x_{jt}	whether or not training job j with starting time t		
s_{jhp}	# of type-p PSs serving job j in server h		
y_{jhm}	# of type-m workers serving job j in server h		

Especially, a set of $\mathcal{J}$ training jobs arrive with large input datasets during a large time span $[T] = 1, 2, ..., T$, to train different ML models using synchronous training, *i.e.*, synchronous stochastic gradient descent (S-SGD) method. Synchronous training can typically ensure model convergence and achieve higher model accuracy than asynchronous training [30][19], and is hence widely adopted over the latter in AI clouds of leading IT companies [1]. The large input dataset of job j ($j \in [J]$) is divided into D_j equal-sized data chunks. Each data chunk is divided into K_j equal-sized mini-batches. We consider the PS framework in this work.

Let H denote the number of physical servers for the deployment of workers and PSs. Each server $h \in [H]$ offers C_h^r units of type-r resource. R represents the number of resource types, including GPU, CPU, memory and bandwidth. Workers and PSs are implemented as virtual machines (VMs) or containers in physical servers. We refer to workers and PSs with different resource allocations as different types. Let M and P denote the number of worker and PS types, respectively. Each type-m ($m \in [M]$) worker (type-p ($p \in [P]$) PS) consumes e_m^r (z_p^r) units of type-r ($r \in [R]$) resource. Let b_m (B_p) be the bandwidth occupied by each worker m (PS p), *i.e.*, $b_m = e_m^{bandwidth}$ ($B_p = z_p^{bandwidth}$).

Upon the arrival of an ML job j at time a_j, the following decisions are made: (i) when to start the job, denoted by binary variable x_{jt}: $x_{jt} = 1$ if job j is executed with starting time t; (ii) the number of allocated type-m workers serving job j deployed on physical server h at and after a_j, indicated by integer variable y_{jhm}; (iii) the number of allocated type-p PSs serving job j deployed on physical server h at and after a_j, indicated by integer variable s_{jhp}; (iv) the amount of consecutive time slots allocated to job j, which is related to the number and processing capacity of workers serving job j, specified by d_j. We do not consider preemption in this work, because when a job is suspended, the entire image of the job needs to be stored temporarily, which increases the overhead. Table 1 summarizes important notations for easy reference.

3.2 Training Process with PS framework

The set of global parameters of each ML job is partitioned into several partitions, each maintained by one PS [23]. Each worker of job j has a complete replica of the training model. Each worker processes allocated mini-batches one by one, sends computed gradients to and receives updated parameters from all job j's PSs after processing one mini-batch (one iteration). The training process at all workers is synchronized: in each iteration, each PS updates its parameters after it has aggregated gradients from all workers, and then sends updated parameters to all workers. When the entire input dataset is trained for one round, an epoch is completed. For an ML job, the input dataset is trained for multiple epochs. Let E_j be the required training epochs of job j.

Let v_{jm} denote the time for a type-m worker to train a mini-batch of job j. Assume the computation time at a type-p PS for updating a partition of global parameters using gradients from all workers in each iteration of job j is a constant, indicated by U_j^p. The time for a type-m worker of job j, deployed on a server with no PS, to transfer gradients to all PSs in other servers is $\frac{\pi_j}{b_m}$, and vice versa, assuming the upload and download bandwidth are the same. When a worker is placed together with some PS(s) in one server, exchanging parameters/gradients with PS(s) in the same server needs no inter-server bandwidth and takes less time. With synchronous training, the time for exchanging gradients/parameters in one iteration of a job depends on the worker that spends the longest time, which is bound by $\frac{\pi_j}{b_m}$, *i.e.*, the time if any worker is not co-located with any PS.

We ignore fetching time of the input data as it can be largely hidden behind training using pipelining. Let q_j indicate whether all workers and PSs of job j are deployed in the same physical server (1) or not (0). Let ρ_{jm}^p denote the processing capacity of each worker, *i.e.*, the number of mini-batches that can be trained by each worker in one time slot, when job j employs type-m worker(s) and type-p PS(s). Thus, we have:

$$\rho_{jm}^p = \begin{cases} 1/(v_{jm} + U_j^p), & \text{if } q_j = 1 \\ 1/(v_{jm} + U_j^p + \frac{2\pi_j}{b_m}), & \text{if } q_j = 0 \end{cases} \quad (1)$$

Note that when not all workers and PSs of job j are on the same server ($q_j = 0$), ρ^p_{jm} represents the upper-bound of time for exchanging gradients/parameters in one training iteration, for model simplification.

3.3 Problem Formulation

We exploit the demand elasticity of ML jobs to minimize the sum of all jobs' weighted completion times [24], that is $\sum_{j\in J} w_j c_j$, where c_j denotes the completion time of job j and $c_j = \sum_{t\in[T]} x_{jt}(t + d_j)$, and w_j can be interpreted as the priority of job j [28]. The objective is equivalent to minimizing average weighted job completion time, given the fixed total number of jobs, J. In practice, a cluster manager can set job weights according to job arrival times, deadlines and workloads. Jobs, which have larger workload and smaller time interval between arrival time and deadline, can be assigned larger weights. The larger a job's weight is, the sooner it is scheduled. If all weights are the same, the system prefers to schedule small jobs earlier, as the total completion time is shorter. This discriminates large jobs. Assigning a larger weight to large jobs can mitigate this problem.

The offline minimization problem can be formulated as the following time-indexed program:

$$\text{minimize} \quad \sum_{j\in[J]} w_j \sum_{t\in[T]} x_{jt}(t + d_j) \tag{2}$$

subject to:

$$\sum_{t\in[T]} x_{jt} = 1, \forall j, \tag{2a}$$

$$|\{m \in [M] | \sum_{h\in[H]} y_{jhm} > 0\}| = 1, \forall j \tag{2b}$$

$$|\{p \in [P] | \sum_{h\in[H]} s_{jhp} > 0\}| = 1, \forall j \tag{2c}$$

$$q_j = 1 \text{ if and only if } h = h', \forall h, h' : y_{jhm} > 0, s_{jh'p} > 0, \forall j, \tag{2d}$$

$$\sum_{h\in[H]} \sum_{p\in[P]} s_{jhp} \geq 1, \forall j, \tag{2e}$$

$$d_j \sum_{h\in[H]} \sum_{m\in[M]} y_{jhm} \rho^p_{jm} \geq E_j D_j K_j, \forall j, \forall p : \sum_{h\in[H]} s_{jhp} > 0 \tag{2f}$$

$$\sum_{h\in[H]} \sum_{m\in[M]} y_{jhm} \leq D_j, \forall j, \tag{2g}$$

$$\sum_{j:t'\in(t-d_j,t]} x_{jt'}\Big(\sum_{m\in[M]} e^r_m y_{jhm} + \sum_{p\in[P]} z^r_p s_{jhp}\Big) \leq C^r_h, \forall t, \forall r, \forall h, \tag{2h}$$

$$\sum_{h'\in[H^{-h}]} \sum_{m\in[M]} y_{jh'm} b_m \leq \sum_{p\in[P]} s_{jhp} B_p, \forall j, \forall h : \sum_{p\in[P]} s_{jhp} > 0, \tag{2i}$$

$$x_{jt} = 0, \forall j, \forall t < a_j, \tag{2j}$$

$$y_{jhm} \in \{0, 1, \dots\}, \forall j, \forall h, \forall m, \tag{2k}$$

$$s_{jhp} \in \{0, 1, \dots\}, \forall j, \forall h, \forall p, \tag{2l}$$

$$d_j \in \{0, 1, \dots\}, \forall j, \tag{2m}$$

$$x_{jt} \in \{0, 1\}, \forall j, \forall t. \tag{2n}$$

$$q_j \in \{0, 1\}, \forall j. \tag{2o}$$

where $\forall j, t, r, h, m, p$ represents $\forall j \in [J], t \in [T], r \in [R], h \in [H], m \in [M], p \in [P]$. Constraint (2a) requires job j to be scheduled once. Constraint (2b) ensures that each job selects and employs one type of workers, as it is common to use the same type of workers to process evenly allocated input data batches for synchronous training. Though there have been recent studies that assign different workers different batch sizes [10], the relevant study is still in its infancy and not widely used in practice. If different types of workers are used in a job, the time for the workers to process equal-sized data batches varies; hence, workers requiring less training time need to wait for slower workers in each iteration, leading to lower resource efficiency. Constraint (2c) requires that each job uses one type of PSs due to the same reason.

Constraint (2d) shows the relationship among q_j, y_{jhm} and s_{jhp}, which is hard and awkward to describe by linear constraint. Constraint (2e) assures that there is at least one PS allocated to each ML job for maintaining its global parameters. Constraint (2f) guarantees that for job j, a sufficient number of workers and time slots are allocated to accomplish training of the dataset for E_j epochs. $E_j D_j K_j$ is the total count of mini-batches trained in job j. Constraint (2g) upper-bounds the number of workers by the number of data chunks D_j, to ensure that one data chunk is trained by at most one worker for E_j epochs. The resource capacity of physical servers for running workers and PSs is formulated by constraint (2h). Here, $x_{jt'} = 1$, $t' \in (t - d_j, t]$ denotes that job j is still running in time slot t. Since each of job j's workers needs to push gradients to and pull computed parameters from all its PSs, the bandwidth reservation for PSs of job j in server h should cover the total bandwidth of job j's workers placed on other servers, which can be formulated as the linear constraint (2i). Here, H^{-h} represents the set of all the servers except h. Constraint (2j) indicates that it is impossible to start job j before its arrival.

Without the non-linear constraints (2b)(2d), the weighted completion time minimization problem in (2) is still a mixed integer linear program (MILP). Even in the offline setting, with information of all jobs given, solving such MILPs is non-trivial and typically NP-hard [15].

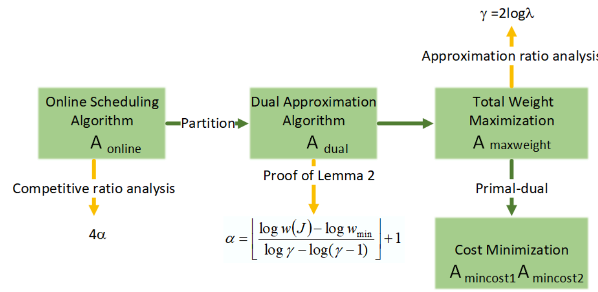

Figure 1: Main idea of our online algorithm A_{online}.

3.4 Algorithmic Idea

In order to solve the weighted completion time minimization problem, we design an efficient online algorithm with bounded competitive ratio (*i.e.,* the maximum ratio of the total weighted completion time incurred by our online algorithm over that incurred by the offline optimal approach which knows all the inputs in advance) in two steps, as shown in Fig. 1.

i. In Sec. 4, we first group unprocessed ML jobs until a certain time point into a batch, to convert the online optimization problem into a series of batch scheduling problems. Then, we invoke a dual approximation algorithm A_{dual} to schedule jobs in a batch. According to Lemma 1 [17], the schedule produced by A_{dual} is required to satisfy two properties. It is hard to yield such a schedule directly. Rather than solving the the batch scheduling problem directly, we focus on a more solvable problem instead, *i.e.*, the total weight maximization problem. Leveraging an approximation algorithm $A_{maxweight}$ for the total weight maximization problem, A_{dual} constructs a required schedule.

ii. In Sec. 5, we introduce an approximation algorithm $A_{maxweight}$ for batch processing, which solves the the total weight maximization problem. $A_{maxweight}$ applies the primal-dual framework and employs two subroutines ($A_{mincost1}$ and $A_{mincost2}$) to choose the schedule with smallest cost for each job.

Here, A_{dual} is a subroutine of A_{online} and a dual approximation algorithm to solve the maximum weighted schedule problem in Definition 1. A_{dual} invokes $A_{maxweight}$ and $A_{maxweight}$ invokes $A_{mincost1}$ and $A_{mincost2}$. $A_{mincost1}$ and $A_{mincost2}$ solve the cost minimization problem in Sec. 5.2. Performance guarantees of various proposed algorithms are shown at the end of the yellow arrows in Fig.1.

4 ONLINE SCHEDULING FRAMEWORK

In Sec. 4.1, we introduce an online scheduling framework A_{online} that partitions the timespan to group ML jobs. It requires a *dual approximation algorithm A_{dual}* for job scheduling, which is presented in Sec. 4.2.

4.1 Online Scheduling Algorithm

Our online algorithm is partly inspired by Leslie *et al.* [17]. The basic idea is to partition the timespan of potential completion times at geometrically increasing points, and iteratively schedule unprocessed ML jobs until a certain time point. More specifically, let $\tau_0 = 1$, $\tau_i = 2^{i-1}$. In rounds $i = 1, 2, \ldots$, we wait until time τ_i. Let J_i represent the set of jobs that have arrived by time τ_i, but still not scheduled. Next, we require a *dual approximation algorithm A_{dual}* for J_i, which produces a schedule of length at most $\alpha\tau_i$ ($\alpha > 1$, which is a number to indicate the infeasibility of the schedule produced by A_{dual}) and whose total weight is at least the optimal weight of the maximum weighted schedule problem in Sec. 5. The schedule generated by A_{dual} is then assigned to run from time $\alpha\tau_i$ to time $\alpha\tau_{i+1}$. Because $\alpha\tau_{i+1} - \alpha\tau_i \geq \alpha\tau_i$, it is flexible to run job with length at most $\alpha\tau_i$ in interval $[\alpha\tau_i, \alpha\tau_{i+1}]$, and hence our online algorithm produces feasible schedules.

Definition 1. **The Maximum Weighted Schedule Problem:** *In an ML cluster, given a deadline τ_i, a set of jobs J_i at the beginning, and a weight for each job, we aim to construct a feasible schedule that maximizes the total weight of jobs completed by time τ_i.*

In A_{online} (Algorithm 1), J_i^s denotes the set of jobs scheduled during round *i*. Note that $\tau_0 = 1$ implies the assumption that no job can complete within the first time slot. Lines 3-5 group unscheduled jobs into set J_i. We invoke the dual approximation algorithm Algorithm A_{dual} for J_i in line 6. Next, we run $j \in [J_i^s]$ from time $\alpha\tau_i$ to time $\alpha\tau_{i+1}$ according to the schedule produced by A_{dual} in line 8-9. In line 11, we add job(s) in J_i which is (are) not scheduled in round i to set J_{i+1}, to process in next round $i + 1$.

Algorithm 1 An Online Algorithm A_{online}

Input: T, $C_h^r, \forall h \in [H], r \in [R]$;
Output: $x_{jt}, y_{jhm}, s_{jhp}, d_j, \forall j \in [J], t \in [T], m \in [M], p \in [P], h \in [H]$;
1: Initialize $x_{jt} = 0$, $y_{jhm} = 0$, $s_{jhp} = 0$, $d_j = 0, \forall j \in [J], t \in [T], m \in [M], p \in [P], h \in [H], J_i = \emptyset$;
2: **while** $i = 1, 2, \ldots$ **do**
3: **while** $t < \tau_i$ **do**
4: $J_i = J_i \cup \{j\}$;
5: **end while**
6: $\{\{x_{jt}\}, d_j, \{y_{jhm}\}, \{s_{jhp}\}\}_{j \in J_i, t \in [\alpha\tau_i]} = A_{dual}(J_i, \tau_i, \{C_h^r\})$;
7: **for all** $j \in [J_i^s]$ **do**
8: Run job j from time $\alpha\tau_i$ to time $\alpha\tau_{i+1}$ according to $(\{x_{jt}\}, d_j, \{y_{jhm}\}, \{s_{jhp}\})$;
9: **end for**
10: $J_{i+1} = J_{i+1} \cup (J_i \setminus J_i^s)$;
11: **end while**

Lemma 1. *Given a dual approximation algorithm for $J_i, i \in 1, 2, \ldots$, which produces a schedule satisfying two properties: (i) the length of the schedule is at most $\alpha\tau_i$; (ii) total weight of the schedule is at least the optimal weight of the corresponding maximum weighted schedule problem, A_{online} is an online 4α-approximation algorithm to minimize the total weighted completion time.*

All missing proofs are in our technical report [3].

4.2 A Dual Approximation Algorithm

The dual approximation algorithm A_{dual} (Algorithm 2) produces desired schedules based on a γ-approximation algorithm for the Maximum Weighted Schedule Problem, that schedules as many unscheduled jobs as possible before a deadline (to be detailed in Sec. 5). Lines 2-4 invoke the γ-approximation algorithm $A_{maxweight}$ for α rounds. Specifically, in the ιth ($\iota \in [\alpha]$) round, we schedule jobs in $J_i \setminus J_i^s$, *i.e.*, jobs in J_i but not served in before rounds, from time $(\iota - 1)\tau_i + 1$ to time $\iota\tau_i$.

Lemma 2. *Given a γ-approximation algorithm for the maximum weighted schedule problem which schedules as many jobs as possible before deadline τ_i, A_{dual} constructs a schedule of length at most $\alpha\tau_i$ and total weight at least the optimal objective value of the corresponding maximum weighted schedule problem.*

Proof: Let $J_{i\iota}^*$ and $J_{i\iota}^s$ be the set of jobs served optimally and completed by A_{dual} in the ιth round, respectively. Thus, the optimal objective value of the total weight maximization problem for J_i is $w(J_{i1}^*)$. And let $J_{i\iota}^{s'} = J_{i\iota}^s \cap J_{i1}^*$. In the ιth round, the input of the γ-approximation algorithm is $J_i - \cup_{\iota'=1}^{\iota-1} J_{i\iota'}^s$. When $\iota = 1$, we have

$$w(J_{i1}^s) \geq \frac{1}{\gamma} w(J_{i1}^*) \tag{3}$$

For $\iota \geq 2$, consider jobs which can be scheduled by the optimal solution but are not served by A_{dual} in the first $\iota - 1$ rounds, *i.e.*, $J_{i1}^* - \cup_{\iota'=1}^{\iota-1} w(J_{i\iota'}^{s'})$. In ιth round, since each $j \in [J_{i1}^* - \cup_{\iota'=1}^{\iota-1} w(J_{i\iota'}^{s'})]$ can be completed by the optimal solution, $w(J_{i\iota}^*) \geq w(J_{i1}^* - \cup_{\iota'=1}^{\iota-1} w(J_{i\iota'}^{s'}))$. Then we have

$$w(J_{i\iota}^s) \geq \frac{1}{\gamma}(w(J_{i1}^*) - \sum_{\iota'=1}^{\iota-1} w(J_{i\iota'}^{s'})) \geq \frac{1}{\gamma}(w(J_{i1}^*) - \sum_{\iota'=1}^{\iota-1} w(J_{i\iota'}^{s})) \tag{4}$$

For $\iota \in [\alpha]$, the following inequality holds:

$$\sum_{\iota'=1}^{\iota} w(J_{i\iota}^s) \geq [1-(1-\frac{1}{\gamma})^{\iota}]w(J_{i1}^*) \tag{5}$$

We prove (5) by induction. (5) must hold for $\iota = 1$, since (3) holds. Suppose (5) holds for ι, according to (4), we have $\sum_{\iota'=1}^{\iota+1} w(J_{i\iota}^s) \geq \frac{1}{\gamma}w(J_{i1}^*) + (1-\frac{1}{\gamma})\sum_{\iota'=1}^{\iota} w(J_{i\iota}^s) \geq [1-(1-\frac{1}{\gamma})^{\iota+1}]w(J_{i1}^*)$. Thus we prove (5). Suppose for the specific ι^*, $\sum_{\iota'=1}^{\iota^*} w(J_{i\iota'}^s) \geq w(J_{i1}^*)$ and $\sum_{\iota'=1}^{\iota^*-1} w(J_{i\iota'}^s) < w(J_{i1}^*)$. Note that $J_{i1}^* - \cup_{\iota'=1}^{\iota^*-1} w(J_{i\iota'}^s) \neq \emptyset$, then $w(J_{i\iota^*}^s) \geq \min_{j\in[J_{i1}^*]} w_j \geq w_{min}$, here $w_{min} = \min_{j\in[J]} w_j$. And since (5), $w(J_{i\iota^*}^s) \geq (1-\frac{1}{\gamma})^{\iota^*-1} w(J_{i1}^*)$. So $(1-\frac{1}{\gamma})^{\iota^*-1} w(J_{i1}^*) \geq w_{min}$, then $\iota^* \leq \frac{\log w(J_{i1}^*) - \log w_{min}}{\log\gamma - \log(\gamma-1)} + 1$. We can set $\alpha = \lfloor \frac{\log w(J) - \log w_{min}}{\log\gamma - \log(\gamma-1)} \rfloor + 1$, which satisfies

$$\alpha \geq \lfloor \frac{\log w(J_{i1}^*) - \log w_{min}}{\log\gamma - \log(\gamma-1)} \rfloor + 1 \geq \iota^*, \forall i \tag{6}$$

such that $\sum_{\iota'=1}^{\alpha} w(J_{i\iota'}^s) \geq w(J_{i1}^*), \forall i$. □

Algorithm 2 A Dual Approximation Algorithm A_{dual}

Input: $J_i, \tau_i, C_h^r, \forall h \in [H], r \in [R]$;

Output: $x_{jt}, y_{jhm}, s_{jhp}, d_j, J_i^s, \forall j \in [J_i], t \in [\tau_i], m \in [M], p \in [P], h \in [H]$;

Initialize $x_{jt} = 0, d_j = 0, y_{jhm} = 0, s_{jhp} = 0, \beta_h^r(t) = 0, J_i^s = \emptyset, \delta_h^r(t) = \Delta_h^r(0), \forall j \in [J_i], t \in [\tau_i], m \in [M], h \in [H], p \in [P], r \in [R]$;

for $\iota = 1$ to α **do**

  $\{\{x_{jt}\}, d_j, \{y_{jhm}\}, \{s_{jhp}\}\}_{j\in(J_i\setminus J_i^s), t\in[(\iota-1)\tau_i+1, \iota\tau_i]} = A_{maxweight}(J_i \setminus J_i^s, \tau_i, \{C_h^r\})$;

end for

5 APPROXIMATION ALGORITHM FOR TOTAL WEIGHT MAXIMIZATION

We next present an approximation algorithm $A_{maxweight}$ for batch processing, employing a primal-dual algorithm in Sec. 5.1. As subroutines of $A_{maxweight}$, we design two algorithms in Sec. 5.2 to compute the best schedule for each job. Theoretical analysis is presented in Sec. 5.3.

5.1 The Maximum Weighted Schedule Problem

We formulate a maximum weighted schedule problem for each round i in our online scheduling framework, that maximizes the total weight of jobs in J_i completed by time τ_i.

$$\text{maximize} \quad \sum_{j\in[J_i]} \sum_{t\in[\tau_i]} w_j x_{jt} \tag{7}$$

subject to:

$$\sum_{t\in[\tau_i]} x_{jt} \leq 1, \forall j \in [J_i], \tag{7a}$$

$$\sum_{t\in[\tau_i]} x_{jt}(t+d_j) \leq \tau_i, \forall j \in [J_i], \tag{7b}$$

$$(2b)-(2i), (2k)-(2o), \; where \; \forall t \in [\tau_i].$$

This maximization problem involves integer variables, non-linear constraint (2b) (2c) and constraints concerning multiplication of variables (2f)(2h)(7b). To address these challenges, we first apply the compact-exponential techniques [36] to reformulate problem (7) into an equivalent conventional integer linear program (ILP) with packing structure:

$$\text{maximize} \quad \sum_{j\in[J_i]} \sum_{l\in\Gamma_j} w_j x_{jl} \tag{8}$$

subject to:

$$\sum_{j\in[J_i]} \sum_{l:t\in T(l), h\in l} x_{jl} f_{jh}^r(l) \leq C_h^r, \forall t \in [\tau_i], r \in [R], h \in [H], \tag{8a}$$

$$\sum_{l\in\Gamma_j} x_{jl} \leq 1, \forall j \in [J_i], \tag{8b}$$

$$x_{jl} \in \{0,1\}, \forall j \in [J_i], l \in \Gamma_j. \tag{8c}$$

In the above ILP, Γ_j is the set of feasible schedules for job j, each corresponding to the set of decisions $(x_{jt}, d_j, y_{jhm}, s_{jhp}, q_j, \forall m \in [M], p \in [P], h \in [H], t \in [\tau_i])$ satisfying constraints (7b)(2b)(2c)(2f)(2i)(2k)(2n). Binary variable x_{jl} indicates whether job j is scheduled according to schedule $l \in \Gamma_j$ or not, $\forall j \in [J], l \in \Gamma_j$. $T(l)$ records the allocated time slots of job j in schedule $l \in \Gamma_j$. We use $h \in l$ to indicate that schedule l uses server h to deploy workers and PSs for job j. $f_{jh}^r(l)$ denotes the total type-r resource occupation of job j's schedule l on server h, *i.e.*, $f_{jh}^r(l) = \sum_{m\in l, p\in l}(e_m^r y_{jhm}^l + z_p^r s_{jhp}^l), \forall h \in l, r \in [R]$, where $m \in l$, $p \in l$ specify that schedule l trains the model using type-m workers and type-p PSs, and y_{jhm}^l (s_{jhp}^l) represents the given number of workers m (PSs p) on server h in l.

Constraint (8a) is equivalent to (2h). Constraint (8b) ensures that each job is executed according to at most one schedule. A feasible solution to ILP (8) has a corresponding feasible solution in problem (7), and vice versa, with the same objective value. Note that we introduce an exponential number of variables in ILP (8), each corresponding to a possible schedule of job j. To solve ILP (8), we formulate the dual LP of ILP (8) by relaxing $x_{jl} \in \{0,1\}$ to $x_{jl} \geq 0$ and introducing dual variables $\delta_h^r(t)$ and u_j to constraints (8a) and (8b):

$$\text{minimize} \sum_{j\in[J_i]} u_j + \sum_{t\in[\tau_i]} \sum_{h\in[H]} \sum_{r\in[R]} \delta_h^r(t) C_h^r \tag{9}$$

subject to:

$$u_j \geq w_j - \sum_{t\in T(l)} \sum_{h\in l} \sum_{r\in[R]} \delta_h^r(t) f_{jh}^r(l), \forall j \in [J_i], l \in \Gamma_j, \tag{9a}$$

$$\delta_h^r(t), u_j \geq 0, \forall j \in [J_i], t \in [\tau_i], h \in [H], r \in [R]. \tag{9b}$$

If we interpret dual variable $\delta_h^r(t)$ as the unit cost of type-r resource on server h in time t, then $\sum_{t\in T(l)} \sum_{h\in l} \sum_{r\in[R]} \delta_h^r(t) f_{jh}^r(l)$ is the total resource cost of all workers and PSs serving job j by schedule l. The RHS of (9a), *i.e.*, job weight minus overall resource cost of job j with schedule l, is the job utility. To minimize the dual objective, we assign dual variables u_j to be the maximum between 0 and the RHS of (9a) according to the best schedule l_j:

$$u_j = \max\{0, \max_{l\in\Gamma_j} \text{RHS of (9a)}\}. \tag{10}$$

If $u_j > 0$, we construct schedule of job j according to l_j ($x_{jl_j} = 1$); or otherwise, we do not schedule it ($x_{jl} = 0, \forall l \in \Gamma_j$). The rationale is that, given limited resources, we wish to schedule jobs with larger utility.

$A_{maxweight}$ in Algorithm 3 is our offline algorithm for the maximum weighted schedule problem with the input job set ϕ. Line 1 initializes primal and dual variables. For each job j in ϕ, lines 3 and 4 invoke $A_{mincost2}$ and $A_{mincost1}$ to find a schedule with the lowest cost in the two cases, *i.e.*, $q_j = 1$ and $q_j = 0$, respectively. Comparing the resulting solutions, we obtain the best schedule with the highest utility u_j for job j in lines 5-7. If $u_j > 0$, we set all primal variables according to l_j in lines 10-11 and update the dual variables using the following carefully designed price functions $\delta_h^r(\cdot)$ in line 14. Line 12 updates J_i^s, *i.e.*, the set of jobs which have been scheduled in the ith round. In line 13, $\beta_h^r(t)$ records the amount of allocated type-r resource on server h for time t.

$$\delta_h^r(\beta_h^r(t)) = \lambda^{\frac{\beta_h^r(t)}{C_h^r}} - 1, \forall h \in [H], r \in [R], t \in [\tau_i], \quad \text{where } \lambda = 2(THRF) + 1 \tag{11}$$

Algorithm 3 Total Weight Maximization $A_{maxweight}$

Input: ϕ, τ_i, C_h^r, $\forall h \in [H]$, $r \in [R]$;
Output: x_{jt}, y_{jhm}, s_{jhp}, d_j, q_j, J_i^s, $\forall j \in [J_i]$, $t \in [\tau_i]$, $m \in [M]$, $p \in [P]$, $h \in [H]$;
1: Initialize $x_{jt} = 0$, $d_j = 0$, $y_{jhm} = 0$, $s_{jhp} = 0$, $\beta_h^r(t) = 0$, $\delta_h^r(t) = \Delta_h^r(0)$, $\forall j \in [\phi]$, $t \in [\tau_i]$, $m \in [M]$, $h \in [H]$, $p \in [P]$, $r \in [R]$;
2: **for** each job $j \in [\phi]$ **do**
3: $(\text{cost}_j, l_j) = A_{mincost2}(\tau_i, \{\beta_h^r(t)\}, \{\delta_h^r(t)\}, \{C_h^r\})$;
4: $(\text{cost}, l) = A_{mincost1}(\tau_i, \{\beta_h^r(t)\}, \{\delta_h^r(t)\}, \{C_h^r\})$;
5: **if** cost < cost$_j$ **then**
6: cost$_j$ = cost, $l_j \Leftarrow l$;
7: **end if**
8: $u_j = w_j - \text{cost}_j$;
9: **if** $u_j > 0$ **then**
10: $x_{jt^-} = 1$, $d_j = L_j$;
11: Set q_j, y_{jhm}, s_{jhp} according to l_j, $\forall h \in l_j$, $m \in l_j$, $p \in l_j$;
12: $J_i^s = J_i^s \cup \{j\}$;
13: $\beta_h^r(t) = \beta_h^r(t) + f_{jh}^r(l_j)$, $\forall t \in T(l_j)$, $h \in [H]$, $r \in [R]$;
14: Update $\delta_h^r(t)$, $\forall t \in T(l_j)$, $h \in [H]$, $r \in [R]$ with (11);
15: **end if**
16: **end for**

We make two assumptions. First, the per unit resource per time slot weight is bounded: $1 \leq \frac{w_j}{\sum_{t \in T(l)} \sum_{h \in l} \sum_{r \in [R]} f_{jh}^r(l)} \leq F, \forall j, l, h, r$. Second, $\frac{f_{jh}^r(l)}{C_h^r} \leq \frac{1}{\log \lambda}$, which implies that the one type resource demand of each job on one server is small as compared to the resource capacity of each server. The price function starts at zero and increases exponentially with the increase of resource consumption. When there is little usage of type-r resource on server h, $\beta_h^r(t)$ is close to zero, which allows jobs to consume resource freely. When type-r resource on server h is exhausted, $\beta_h^r(t)$ is close to the resource capacity C_h^r, and $\delta_h^r(t)$ grows fast to a carefully designed large value λ, so that type-r resource on server h will be barely allocated to a job, unless its weight is sufficiently large.

5.2 Cost Minimization Problem

Since w_j is a constant, the utility maximization problem of job j is equivalent to the following cost minimization problem:

$$\min \sum_{t \in [t', t'+d_j)} \sum_{h \in [H]} \sum_{r \in [R]} x_{jt'} \delta_h^r(t) \Big(\sum_{m \in [M]} e_m^r y_{jhm} + \sum_{p \in [P]} z_p^r s_{jhp} \Big) \tag{12}$$

subject to:

$$\sum_{t \in [\tau_i]} x_{jt} = 1, \tag{12a}$$

$$(7b), (2b) - (2g), (2i), (2k) - (2o), \forall t \in [\tau_i], \text{ for the specific } j.$$

We next show the schedule that minimizes job j's cost can be found efficiently and optimally using Algorithm 5 and Algorithm 4. When we fix the worker type m and the PS type p serving job j, the number of acquired time slots is at most $\lceil \frac{E_j D_j K_j}{\rho_{jm}^p} \rceil$. For a fixed allocated time slot d_j, the number of workers needed is at least $\lceil \frac{E_j D_j K_j}{d_j \rho_{jm}^p} \rceil$. If we further know the starting time of job j, problem (12) is simplified as the following ILP, where $m = m'$, $p = p'$, $t' = t^-$, $t^+ = t^- + d_j$:

$$\min_{\boldsymbol{y}, \boldsymbol{s}} \quad \text{cost}(m', p', t^-, t^+) = \sum_{t \in [t^-, t^+)} \sum_{h \in [H]} \sum_{r \in [R]} \delta_h^r(t) (e_{m'}^r y_{jhm'} + z_{p'}^r s_{jhp'}) \tag{13}$$

subject to:

$$q_j = 1 \text{ if and only if } h = h', \forall h, h' : y_{jhm'} > 0, s_{jh'p'} > 0, \tag{13a}$$

$$\sum_{h \in [H]} y_{jhm'} \leq D_j, \tag{13b}$$

$$\sum_{h \in [H]} y_{jhm'} \geq \lceil \frac{E_j D_j K_j}{d_j \rho_{jm'}^{p'}} \rceil, \tag{13c}$$

$$s_{jhp'} B_{p'} \geq \sum_{h' \in [H^{-h}]} y_{jh'm'} b_{m'}, \forall h : s_{jhp'} > 0, \tag{13d}$$

$$\sum_{h \in [H]} s_{jhp'} \geq 1, \tag{13e}$$

$$y_{jhm'}, s_{jhp'} \in \{0, 1, \dots\}, \forall h \in [H], \forall p \in [P], \tag{13f}$$

$$q_j \in \{0, 1\}. \tag{13g}$$

That is, we need to find the best placement scheme for job j to minimize the overall resource cost satisfying constraints (13a)-(13g). Particularly, consider the situation where we deploy all j's workers and PSs on one server, *i.e.*, $q_j = 1$. Note that constraint (13d) is satisfied naturally, since the RHS of (13d) is zero. We come up with algorithms to find the best schedule with the smallest cost for job j as $A_{mincost2}$ and $A_{mincost1}$. $A_{mincost2}$ handles the case where all workers and PSs of job j are running on one server, *i.e.*, $q_j = 1$, $\rho_{jm}^p = 1/(v_{jm} + U_j^p)$, and $A_{mincost1}$ solves the other, *i.e.*, $q_j = 0$, $\rho_{jm}^p = 1/(v_{jm} + U_j^p + \frac{2\pi_j}{b_m})$.

In $A_{mincost1}$, we record the amount of available type-r resource on server h at time slot t using $\omega_h^r(t)$ in line 2. Next, we enumerate the worker and PS types serving job j in line 3 and 4. Then, we traverse possible execution time and compute the number of workers needed in lines 5-6. Given starting time t^- in line 7, we sort servers for worker m' deployment in non-decreasing order of total resource cost $\sum_{t \in [t^-, t^+)} \sum_{r \in [R]} \delta_h^r(t) e_{m'}^r$ recorded by Ω_h in line 8. Then lines 9-33 maximally deploy workers starting from the cheapest server, respecting capacity constraint (2h), the required number of workers N_j in (13c) and bandwidth reservation constraints (13d). Specifically, we decide the number of workers and PSs in given server n in lines 14-22 in a greedy manner, *i.e.*, the maximum number of workers and PSs are placed satisfying (13d). If there are not enough workers or PSs, completing job j is infeasible (lines 25 and 26); otherwise, we compute the overall cost $\sum_{t \in [t^-, t^+)} \sum_{h \in [H]} \sum_{r \in [R]} \delta_h^r(t)(e_{m'}^r y_{jhm'} + z_{p'}^r s_{jhp'})$ (line 28). We identify the schedule with smallest cost in lines 30-32. Finally, we

Algorithm 4 Subroutine for Job j $A_{mincost1}$

Input: $\tau_i, \beta_h^r(t), \delta_h^r(t), C_h^r, \forall h \in [H], , r \in [R], t \in [\tau_i]$;
Output: l_j, cost_m;
1: Initialize $u_j = 0, l_j = \emptyset$, cost_m $= +\infty$;
2: $q_j = 0, \omega_h^r(t) = C_h^r - \beta_h^r(t), \forall h, r, t$;
3: **for** $m' = 1$ to M **do**
4:   **for** $p' = 1$ to P **do**
5:     **for** $L_j = \lceil \frac{EjK_j}{\rho_{jm'}^{p'}} \rceil$ to $\lceil \frac{EjD_jK_j}{\rho_{jm'}^{p'}} \rceil$ **do**
6:       $N_j = \lceil \frac{EjD_jK_j}{L_j \rho_{jm'}^{p'}} \rceil, \hat{N} = N_j$;
7:       **for** $t^- = 1$ to $\tau_i - L_j$ **do**
8:         List $h \in [H]$ in nondecreasing order of Ω_h, $t^+ = t^- + L_j$;
9:         **for** $n = 1, \ldots, H$ **do**
10:           $y_{jhm} = 0, s_{jhp} = 0, \forall m, p, h$;
11:           **for** $k = 1, \ldots, H$ **do**
12:             $\hat{y} = \min\{\min_{r\in[R], t\in[t^-,t^+)} \lfloor \frac{\omega_k^r(t)}{e_{m'}^r} \rfloor, \hat{N}\}$;
13:             $y_{jkm'} = \hat{y}$;
14:             **if** $k = n$ **then**
15:               **for** $g = 0$ to $\hat{y}$ **do**
16:                 $\hat{s} = \min_{r\in[R], t\in[t^-,t^+)} \lfloor \frac{\omega_n^r(t) - g e_{m'}^r}{z_{p'}^r} \rfloor$;
17:                 **if** $\hat{s}B_{p'} \geq (N_j - g)b_{m'}$ **then**
18:                   $y_{jnm'} = g$;
19:                   $s_{jnp'} = \min\{\hat{s}, \lceil \frac{(N_j - g)b_{m'}}{B_{p'}} \rceil\}$;
20:                 **end if**
21:               **end for**
22:             **end if**
23:             $\hat{N} = \hat{N} - y_{jkm'}$;
24:           **end for**
25:           **if** $\hat{N} > 0$ **or** $s_{jnp'} < 1$ **then**
26:             cost $= +\infty$;
27:           **else**
28:             Compute cost;
29:           **end if**
30:           **if** cost < cost_m **then**
31:             cost_m = cost, $l_j \Leftarrow \{t^-, L_j, \boldsymbol{y}, \boldsymbol{s}, q_j\}$;
32:           **end if**
33:         **end for**
34:       **end for**
35:     **end for**
36:   **end for**
37: **end for**
38: **return** l_j, cost_m

return the resulting schedule l_j and the corresponding cost cost_m in line 38.

Compared to $A_{mincost1}$, $A_{mincost2}$ counts the range of acquired time slots and number of workers needed with different processing capacities. We enumerate the server to run all workers and PSs on it.

5.3 Theoretical Analysis

Theorem 1. *Algorithm 5 and Algorithm 4 yield an optimal solution of problem (13) in two scenarios, respectively.*

Algorithm 5 Subroutine for Job j $A_{mincost2}$

Input: $\tau_i, \beta_h^r(t), \delta_h^r(t), C_h^r, \forall h \in [H], , r \in [R], t \in [\tau_i]$;
Output: l_j, cost_m;
Initialize $u_j = 0, l_j = \emptyset$, cost_m $= +\infty$;
$q_j = 1, \omega_h^r(t) = C_h^r - \beta_h^r(t), \forall h, r, t$;
while traverse the value space of variables m' p' L_j t^- in order **do**
  for $h = 1, \ldots, H$ **do**
    $y_{jhm} = 0, s_{jhp} = 0, \forall m, p, h$;
    Compute $y_{jhm'}$ and $s_{jhp'}$ respecting (2h) and (13a)
    Set cost according to the feasibility of $y_{jhm'}$ and $s_{jhp'}$
    if cost < cost_m **then**
      cost_m = cost, $l_j \Leftarrow \{t^-, L_j, \boldsymbol{y}, \boldsymbol{s}, q_j\}$;
    end if
  end for
end while
return l_j, cost_m

Theorem 2. *$A_{maxweight}$ in Algorithm 3, with $A_{mincost2}$ and $A_{mincost1}$, computes a feasible solution to problems (7)(8)(9).*

Theorem 3. *A_{online} in Algorithm 1 is 4α-competitive, where $\alpha = \lfloor \frac{\log w(J) - \log w_{min}}{1 + \log\log\lambda - \log(2\log\lambda - 1)} \rfloor + 1$, where λ are defined in (11), $w(J) = \sum_{j\in J} w_j$ and $w_{min} = \min_{j\in[J]} w_j$.*

Proof: According to Lemma 1 [17] and Lemma 2, we know that the competitive ratio of A_{online} is 4α, where $\alpha = \lfloor \frac{\log w(J) - \log w_{min}}{\log\gamma - \log(\gamma - 1))} \rfloor + 1$ and γ is the approximation ratio of $A_{maxweight}$ in Algorithm 3. Then, combining Theorem 4 we finish the proof. □

We observe that the typical value of α is close to 4 in simulation studies. As shown by the proof of Lemma 2, the value of α in each round i should satisfy inequality (6). According to the definition of J_{i1}^*, we can set α to be $\lfloor \frac{\log w(J_i) - \log w_{min}}{\log\gamma - \log(\gamma-1)} \rfloor + 1$ in simulations. Further, if $J_i^s = J_i$ for the specific ι, we can terminate the ith round iteration of A_{dual} and turn to the next round.

Theorem 4. *The approximation ratio of $A_{maxweight}$ in Algorithm 3 is $2\log\lambda$.*

Theorem 5. *A_{online} in Algorithm 1 runs in polynomial time, with time complexity $O((\log w(J))JMPT^2 \log T(H\log H + H^2))$.*

6 PERFORMANCE EVALUATION

Settings. We simulate an ML cluster running for $T \in [100, 300]$ time slots (default value: 150). Each time slot is one hour long. The default number of servers is 150. The overall resource capacities, **C**, are set to be approximately $[0.2, 0.5]$ fraction of the respective overall job resource demand, which is computed by adding the ideal resource demand of all jobs. Resources configuration of each server is set according to Amazon EC2 GPU instances P3, P2 and G3. The numbers of worker and PS types are set to be 8 and 10, respectively. Following similar settings in [23][8][12], we set resource configuration for each type worker as follows: 1 to 4 GPUs, 1 to 10 vCPUs and bandwidth of 100Mbps to 5Gbps. Resource configuration for each type PS is: 1 to 10 vCPU and bandwidth of 5Gbps to 20 Gbps. For each job, w_j is in $[200, 5000]$, E_j is set within $[50, 100]$, D_j is in $[5, 50]$, K_j is in $[10, 50]$, U_j^p is in $[10, 100]$ milliseconds, v_{jm} is in $[0.001, 0.05]$ time slots, and π_j is within $[30, 575]$MB [19][8].

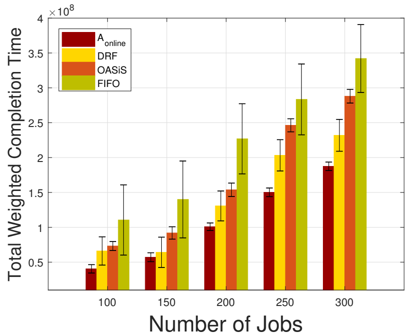

Figure 2: Total weighted completion time.

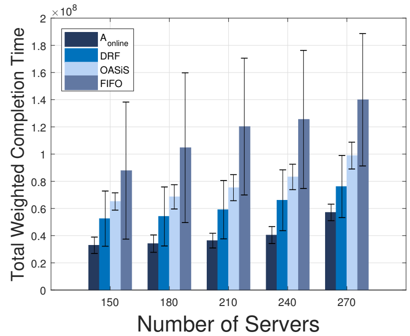

Figure 3: Total weighted completion time.

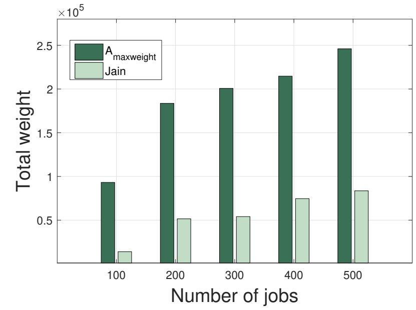

Figure 4: Total scheduled job weight of $A_{maxweight}$ and Jain *et al.*'s algorithm [20].

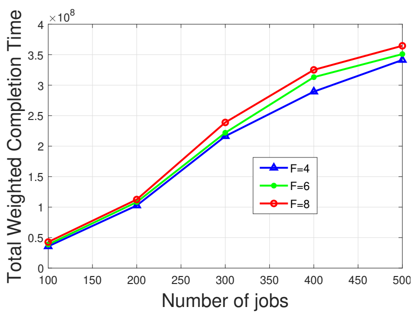

Figure 5: Total weighted completion time of A_{online} under different F.

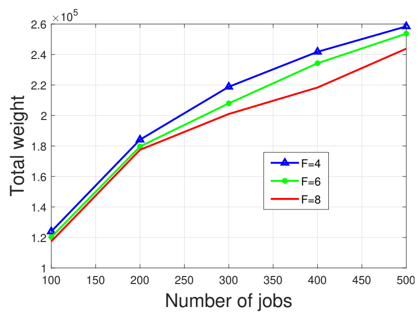

Figure 6: Total weight of $A_{maxweight}$ under different F.

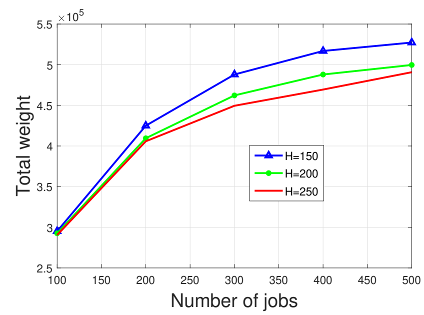

Figure 7: Total weight of $A_{maxweight}$ under different H.

Algorithms for comparison. We compare A_{online} with three job scheduling policies: (i) OASiS: given unit resource prices, jobs with larger utility, which maps to u_j in Sec. 5.1, are served first and each job selects the best placement scheme that minimizes its placement cost [8]. (ii) FIFO: default scheduler in Hadoop and Spark [31]; jobs run by order of arrival, with fixed numbers and resource configuration of workers (and PSs). The number of workers is fixed to a number within $[1, 30]$ for FIFO. (iii) Dominant Resource Fairness Scheduling (DRF): default scheduler in YARN [27] and Mesos [18]; the numbers of workers (and PSs) are computed to achieve max-min fairness in dominant resources [16]. In (i)-(iii), the resource configuration of workers (and PSs) is the same as that in the ideal case, which is derived according to recent literature [14] [32] [25] in our simulation studies. We compare $A_{maxweight}$ with an algorithm from recent literature [20] which proposes a greedy strategy to schedule jobs with deadlines in the offline scenario.

6.1 Performance of A_{online}

Fig. 2 compares the total weighted completion time produced by different algorithms under different numbers of jobs, where $T = 300$. A_{online} performs up to 200% better than the other algorithms in both cases. The objective value may grow with the increase of number of servers according to Fig. 3. Note that λ in price function (11) increases in line with the number of servers H. A_{online} prefers to schedule jobs of larger weight with larger λ when available resources are insufficient. Thus, when the overall resource capacities nearly remain the same, the total amount of fragment resources increases and effective resource capacity of the servers decreases with larger H. The objective values in Fig. 2 (Fig. 3) are the average of multiple trials.

Fig. 5 calculates the objective value obtained by A_{online} under different F, *i.e.*, the upper bound of the weight per unit resource per time slot. Recall that parameter λ in the price function and the theoretical competitive ratio are related to F. We can see that for larger values of F, the objective value is larger. Larger F represents larger weights of served jobs, *i.e.*, jobs with weight which is not large enough will be executed later. We apply the *tic* and *toc* functions in MATLAB to measure the execution time of our online algorithm. We run 10 tests on a desktop computer (Intel Core i3-6100/8GB RAM) and present the average result in Fig. 8. We can observe that, the running time of A_{online} increases with the number of jobs, but still remains at a low level (< 2 minutes). We can observe that the numbers of worker and PS allocated to jobs are in $[4, 25]$ and $[1, 4]$.

6.2 Performance of $A_{maxweight}$

Fig. 4 compares the total weight achieved by $A_{maxweight}$ with related algorithm from recent literature [20]. Our offline algorithm $A_{maxweight}$ performs much better than the other. Fig. 7 shows the total weight of $A_{maxweight}$ under different H, *i.e.*, the number of servers to deploy workers and PSs. It reflects that the total weight is smaller for larger values of H because the total amount of fragment resources increases with the increase of the number of servers. In Fig. 7, there is an upward trend in the total weight with the increment of the number of jobs. Fig. 6 represents the total weight of $A_{maxweight}$ under different F, which is related to price function in line 14 of $A_{maxweight}$. We can see that for smaller values of F,

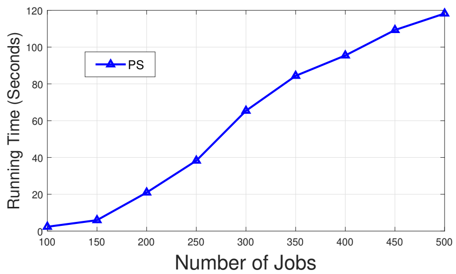

Figure 8: Running time of A_{online}.

the total weight is larger. Smaller F represents more jobs can be served with the same total number of jobs, particularly, jobs with smaller weight.

7 CONCLUSION

We proposed an online algorithm for scheduling synchronous training jobs in ML clusters. The online algorithm targets total weighted completion time minimization, consisting of (i) an online greedy-interval algorithm that converts the online scheduling problem into a series of batch processing problems; (ii) a primal-dual algorithm running for each batch, which computes the best execution window of each job, with proper number and type of workers (and parameter servers). Both theoretical analysis and trace-driven simulation studies validate our online algorithm's good performance, as compared to both offline optimum and commonly used scheduling algorithms in read-world cloud systems.

ACKNOWLEDGMENTS

This work was supported by the Fundamental Research Funds for the Central Universities (2042019kf0016) and grants from Hong Kong RGC under the contracts HKU 17204619, 17225516, C5026-18G (CRF).

REFERENCES

[1] [n.d.]. *Distributed Training in TensorFlow.* https://www.tensorflow.org/guide/distribute_strategy.

[2] [n.d.]. *Microsoft Cognitive Toolkit.* https://www.microsoft.com/en-us/cognitive-toolkit/.

[3] [n.d.]. *Technical report.* https://1drv.ms/b/s!AvAD6Lae6eSxa3HIfAJ4xM3dBaU?e=fmPk0B.

[4] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proc. of USENIX OSDI.*

[5] Mohammad Mohammadi Amiri and Deniz Gündüz. 2019. Computation scheduling for distributed machine learning with straggling workers. In *Proc. of IEEE ICASSP.*

[6] Yossi Azar, Inna Kalp-Shaltiel, Brendan Lucier, Ishai Menache, Joseph (Seffi) Naor, and Jonathan Yaniv. 2015. Truthful online scheduling with commitments. In *Proc. of ACM EC.*

[7] Yixing Bao, Yanghua Peng, and Chuan Wu. 2019. Deep Learning-based Job Placement in Distributed Machine Learning Clusters. In *Proc. of IEEE INFOCOM.*

[8] Yixing Bao, Yanghua Peng, Chuan Wu, and Zongpeng Li. 2018. Online Job Scheduling in Distributed Machine Learning Clusters. In *Proc. of IEEE INFOCOM.*

[9] Chen Chen, Wei Wang, and Bo Li. 2018. Performance-Aware Fair Scheduling: Exploiting Demand Elasticity of Data Analytics Jobs. In *Proc. of IEEE INFOCOM.*

[10] Chen Chen, Wei Wang, and Bo Li. 2019. Round-Robin Synchronization: Mitigating Communication Bottlenecks in Parameter Servers. In *Proc. of IEEE INFOCOM.*

[11] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2016. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. In *NIPS Workshop on Machine Learning Systems (LearningSys).*

[12] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *Proc. of USENIX OSDI.*

[13] Mosharaf Chowdhury, Zhenhua Liu, Ali Ghodsi, and Ion Stoica. 2016. HUG: Multi-Resource Fairness for Correlated and Elastic Demands. In *Proc. of USENIX NSDI.*

[14] Yuanxiang Gao, Li Chen, and Baochun Li. 2018. Spotlight: Optimizing Device Placement for Training Deep Neural Networks. In *Proc. of ACM ICML.*

[15] Georgii Gens and Evgenii Levner. 1980. Complexity of approximation algorithms for combinatorial problems: a survey. *ACM SIGACT News* 12, 3 (1980), 52–65.

[16] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proc. of USENIX NSDI.*

[17] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. 1997. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of operations research* 22, 3 (1997), 513–544.

[18] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *Proc. of USENIX NSDI.*

[19] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, and Kurt Keutzer. 2016. FireCaffe: Near-Linear Acceleration of Deep Neural Network Training on Compute Clusters. In *Proc. of IEEE CVPR.*

[20] Navendu Jain, Ishai Menache, Joseph (Seffi) Naor, and Jonathan Yaniv. 2015. Near-Optimal Scheduling Mechanisms for Deadline-Sensitive Jobs in Large Computing Clusters. *ACM Trans. Parallel Comput.* 2, 1 (2015), 3.

[21] Lei Jiao, Antonia Tulino, Jaime Llorca, Yue Jin, and Alessandra Sala. 2017. Smoothed online resource allocation in multi-tier distributed cloud networks. *IEEE/ACM Trans. on Netw.* 25, 4 (2017), 2556–2570.

[22] Lei Jiao, Antonia Tulino, Jaime Llorca, Yue Jin, Alessandra Sala, and Jun Li. 2018. Online control of cloud and edge resources using inaccurate predictions. In *Proc. of IEEE/ACM IWQoS.*

[23] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and BorYiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *Proc. of USENIX OSDI.*

[24] Shi Li. 2017. Scheduling to Minimize Total Weighted Completion Time via Time-Indexed Linear Programming Relaxations. In *Proc. of IEEE FOCS.*

[25] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proc. of ACM EuroSys.*

[26] Weijie Shi, Linquan Zhang, Chuan Wu, Zongpeng Li, and Francis C.M. Lau. 2014. An online auction framework for dynamic resource provisioning in cloud computing. In *Proc. of ACM SIGMETRICS.*

[27] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha amd Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache hadoop YARN: Yet another resource negotiator. In *Proc. of ACM SoCC.*

[28] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proc. of ACM EuroSys.*

[29] Tiantian Wang, Zhuzhong Qian, Lei Jiao, Xin Li, and Sanglu Lu. 2020. Geo-Clone: online task replication and scheduling for geo-distributed analytics under uncertainties. In *Proc. of IEEE/ACM IWQoS.*

[30] Feng Yan, Olatunji Ruwase, Yuxiong He, and Trishul Chilimbi. 2015. Performance Modeling and Scalability Optimization of Distributed Deep Learning Systems. In *Proc. of ACM SIGKDD.*

[31] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. In *Proc. of USENIX HotCloud.*

[32] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. 2017. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *Proc. of USENIX ATC.*

[33] Xiaoxi Zhang, Zhiyi Huang, Chuan Wu, Zongpeng Li, and Francis Lau. 2015. Online auctions in IaaS clouds: Welfare and profit maximization with server costs. In *Proc. of ACM SIGMETRICS.*

[34] Zijun Zhang, Zongpeng Li, and Chuan Wu. 2017. Optimal Posted Prices for Online Cloud Resource Allocation. In *Proc. of ACM SIGMETRICS.*

[35] Liang Zheng, Carlee Joe-Wong, Christopher G Brinton, Chee Wei Tan, Sangtae Ha, and Mung Chiang. 2016. On the viability of a cloud virtual service provider. *Proc. of ACM SIGMETRICS.*

[36] Ruiting Zhou, Zongpeng Li, Chuan Wu, and Zhiyi Huang. 2017. An efficient cloud market mechanism for computing jobs with soft deadlines. *IEEE/ACM Trans. on Netw.* 25, 2 (2017), 793–805.

[37] ShangXuan Zou, ChunYen Chen, JuiLin Wu, ChunNan Chou, ChiaChin Tsao, KuanChieh Tung, TingWei Lin, ChengLung Sung, and Edward Y. Chang. 2017. Distributed training large-scale deep architectures. In *Proc. of Springer ADMA.*