Sampling-Based Multi-Job Placement for Heterogeneous Deep Learning Clusters

Kaiyang Liu, *Member, IEEE*, Jingrong Wang, *Student Member, IEEE*, Zhiming Huang, *Student Member, IEEE*, and Jianping Pan, *Fellow, IEEE*

***Abstract*—Heterogeneous deep learning clusters commonly host a variety of distributed learning jobs. In such scenarios, the training efficiency of learning models is negatively affected by the slowest worker. To accelerate the training process, multiple learning jobs may compete for limited computational resources, posing significant challenges to multi-job placement among heterogeneous workers. This article presents a heterogeneity-aware scheduler to solve the multi-job placement problem while taking into account job sizing and load balancing, minimizing the average Job Completion Time (JCT) of deep learning jobs. A novel scheme based on proportional training workload assignment, feasible solution categorization, and matching markets is proposed with theoretical guarantees. To further reduce the computational complexity for low latency decision-making and improve scheduling fairness, we propose to construct the sparsification of feasible solution categories through sampling, which has negligible performance loss in JCT. We evaluate the performance of our design with real-world deep neural network benchmarks on heterogeneous computing clusters. Experimental results show that, compared to existing solutions, the proposed sampling-based scheme can achieve 1) results within 2.04% of the optimal JCT with orders-of-magnitude improvements in algorithm running time, and 2) high scheduling fairness among learning jobs.**

***Index Terms*—Distributed deep learning, job placement, job sizing, load balancing, heterogeneity-aware scheduling, fairness.**

I. Introduction

DEEP learning models, e.g., Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Transformers, have brought breakthroughs to many areas, e.g., computer vision, speech recognition, and natural language processing [1], [2], [3]. Training deep learning models requires massive computing resources. People are building fast parallel computing clusters, often equipped with specialized accelerators such as GPUs, TPUs, and FPGAs, to accelerate training jobs at scale [4]. The deep learning clusters are usually shared by multiple users to reduce operational costs and improve resource utilization.

The hardware configurations of deep learning clusters are intrinsically heterogeneous with a wide variety of accelerators [4]. Due to different computational capacities, the model training efficiency is negatively affected by stragglers, i.e., the workers that run much slower than others [5]. Learning jobs prefer to choose powerful workers to achieve higher training throughput. Learning jobs may compete for computational resources. This brings fundamental challenges to the design of multi-job placement among heterogeneous workers.

Job Placement: Over the past years, many scheduling schemes have been designed to optimize the training of learning jobs from various perspectives, e.g., training throughput, Job Completion Time (JCT), and fairness [4], [6], [7], [8], [9], [10], [11]. However, many previous schemes required the number of workers engaged in training, i.e., job sizing, as prior information [4], [6], limiting the performance to achieve the lowest JCT [12]. In addition, existing schemes also relied on prior workload assignment information to calculate the reward of job placement [13]. In [6], [7], [8], [9], the training dataset is divided into equal-sized parts to feed the workers. In heterogeneous deep learning clusters, uniform workload assignment causes a mismatch between the loads and the worker processing capabilities, incurring non-negligible performance losses.

Load Balancing: One of the most effective strategies to eliminate the negative effect of stragglers is to balance the workloads of workers according to their training throughput, which can be classified into static [14], [15] and dynamic [5], [16], [17] strategies. The previous study assumes the set of workers participating in training is known beforehand. It cannot be directly applied to the multi-tenant scenario where learning jobs are competing for the most capable workers.

The challenges of designing multi-job placement schemes are as follows. First, a critical job sizing decision shall be made when submitting learning jobs: how many workers should be requested for each job to minimize JCT? Second, considering the coupling between job placement and load balancing variables, a naive approach is to optimize the load balancing sequentially after the job placement decision has been found. However, this approach is prone to be sub-optimal in the global configuration space (see the numerical example in Section IV-A).

Manuscript received 16 March 2023; revised 10 April 2024; accepted 12 April 2024. Date of publication 17 April 2024; date of current version 30 April 2024. The authors would like to acknowledge that this work was supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC), Canada Foundation for Innovation (CFI), and British Columbia Knowledge Development Fund (BCKDF). Recommended for acceptance by J. Zola. *(Corresponding author: Jingrong Wang.)*

Kaiyang Liu is with the Department of Computer Science, Memorial University of Newfoundland, St. John's, NL A1B 3X5, Canada (e-mail: kaiyang.liu@mun.ca).

Jingrong Wang is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: jr.wang@mail.utoronto.ca).

Zhiming Huang and Jianping Pan are with the Department of Computer Science, University of Victoria, Victoria, BC V8P 5C2, Canada (e-mail: zhiminghuang@uvic.ca; pan@uvic.ca).

Digital Object Identifier 10.1109/TPDS.2024.3390109

Another approach is to use an alternating optimization technique [18], which involves optimizing one set of variables while keeping the result of the other set constant. However, the multi-round optimization introduces extra overhead and delay.

At last, finding the optimal parallelization policy for deep learning jobs can often be formulated as integer programming problems [8], [18], [19], which are NP-hard and computationally expensive to solve. The high computational complexity leads to long solution times with the increasing scale of learning systems. Furthermore, the parallelization policy often needs to be recalculated periodically to keep up with dynamic system changes, e.g., new job arrival, worker failure, etc. Therefore, production systems frequently rely on heuristics that are easy to calculate [8], [11]. However, previous research work indicates that the heuristics are hard to maintain consistently good performance as the problem scale and inputs change [20], and may incur non-negligible performance loss (see Figs. 9 and 11 in Section VI).

In this work, we propose a novel heterogeneity-aware scheduling scheme tailored for multi-tenant, heterogeneous deep learning clusters. Considering the challenges above, our purpose is to design schemes that run much faster than directly solving integer programming problems and can achieve consistently better performance than existing heuristics. First, through the design of proportional training workload assignment, the problem with two types of job placement and load balancing variables can be simplified into a problem with a single type of variable. Then, for job sizing, all feasible solutions can be classified into different categories Ψ based on the number of selected workers for each job. By applying the matching markets method [21] to all feasible solution categories in Ψ, the parallelization policy can be obtained with theoretical guarantees. The computational complexity of our design is mainly determined by the set size $|\Psi|$. To reduce the computational complexity and improve scheduling fairness, we further propose to construct the sparsification of feasible solution categories through sampling with negligible performance loss. The main contributions of this work include:

- We propose a novel scheme that leverages proportional training workload assignment, feasible solution categorization, and matching markets to achieve optimal training throughput and near-optimal JCT.
- By exploring the features of feasible solution categories in Ψ, a sampling-based scheme is proposed to reduce the computational complexity and improve scheduling fairness.
- Experimental results show that, when compared with the optimal scheme, the proposed sampling-based scheme is over 49$\times$ faster with only a 2.04% performance loss in JCT.

The rest of this paper is organized as follows. Section II summarizes the related work. Section III presents the system model and the problem statement. Sections IV and V-B provide the scheme design along with theoretical analysis. Section VI evaluates the efficiency and performance of our design through extensive experiments. Section VII concludes this paper and lists future work.

II. Related Work

A. Load Balancing

Load balancing schemes can be classified into static and dynamic strategies. The static load balancing is oblivious to running time variations. For example, existing learning frameworks, e.g., TensorFlow [22], Caffe [23], and MXNet [24], uniformly assign workloads to all the workers. Some other static strategies follow fixed assignment rules, e.g., Round-Robin [14] or with a priori knowledge of the system status [15].

In response to the time-varying systems, dynamic strategies adjust the workload distribution through continuous system monitoring and state information collection. FlexRR [16] measures the instantaneous training throughput of workers at a fine granularity. Once the straggler lags behind other workers over a given threshold, certain workloads will be offloaded to a faster worker. Similarly, Su et al. [17] proposed to increase the workloads of workers that finish one iteration of training faster while doing the opposite for those that are slower. Chen et al. [5] proposed to redistribute workloads from heavily-loaded workers to lightly-loaded ones, seeking to equalize the batch processing latencies of all workers.

All load balancing schemes above optimize the individual training of one learning job, which assume full worker participation or that the set of selected workers is given as a priori knowledge. Different from these studies, our goal is to balance the workloads of multiple learning jobs when they are competing for the most capable workers.

B. Job Placement

Deep learning training jobs bring new challenges to the schedulers in computing clusters. For example, the state-of-the-art schedulers in production systems, e.g., Apache YARN [25], only perform non-preemptive scheduling of jobs as they arrive. Consequently, users may experience long queuing delays.

For high training throughput or low JCT, a common and effective solution is to use heuristic methods to determine the job scheduling priority. Tiresias [6] designs the Least Attained Service (LAS) algorithm to prioritize jobs. The job which has received the least amount of temporal and dimensional resources will be scheduled next. E-LAS [7] prioritizes jobs based on the real-time epoch progress rate, i.e., the proportion of the current training epoch over the total number of training epochs. Compared with Tiresias, the average JCT is reduced with such an improvement. Optimus [8] assumes that the remaining time of a training job is predictable, and designs a greedy job placement scheme to maximize the cluster-wide training throughput. Saturn [12] uses a commercial solver for the joint optimization of parallelism selection, job sizing, and schedule construction. Beyond heuristics or using a solver, an alternative strategy is to use machine learning techniques for job placement [9]. The above research focuses on the homogeneous GPU cluster setting.

TABLE I
NOTATIONS

Symbol	Definition
$\mathcal{K}$	Set of heterogeneous workers, $\lvert\mathcal{K}\rvert = K$
$\mathcal{S}$	Set of learning jobs, $\lvert\mathcal{S}\rvert = S$
Π_i, W_i, D_i	Training dataset, model parameters, and model size of learning job i, $i \in \mathcal{S}$
$\pi_{i,k}$	Assigned data samples to worker k, $k \in \kappa_i$, $\pi_{i,k} \subseteq \Pi_i$
$\rho_{i,k}$	Number of processed data samples per second
$\sigma_{i,k}$	Binary variable to indicate whether worker k is chosen for training job i or not
T_i	Number of training epochs of learning job i
Λ_i	JCT of learning job i
r	Data transfer rate among workers
$l_i^{[\mathrm{p}]}, l_i^{[\mathrm{c}]}$	Computation and communication latency of an epoch
Ψ	Set of feasible solution categories
V_i	Training throughput of learning job i
p_k	Price of worker k
F	Fairness degree
$\mathcal{N}$	Set of samples selected from Ψ, $\lvert\mathcal{N}\rvert = N$
$\alpha, \beta \in [0, 1]$	Predefined constants to strike a trade-off

The training dataset is divided into equal-sized parts to feed the workers in cases of data parallelism. The impact of load balancing on training efficiency for heterogeneous clusters is overlooked, causing a non-negligible performance loss.

From another perspective, fair schedulers seek to guarantee that each job can achieve better performance via resource sharing in deep learning clusters. Themis [10] introduces a new metric named finish-time fairness, which aims to assign more computational resources to the jobs whose received service is less than the preplanned amount. Themis is designed for homogeneous computing clusters with a single type of GPU. Gavel [4] is a heterogeneity-aware scheduler for fairness. A round-based policy is designed to improve scheduling flexibility and ensure timely GPU resource reallocation. However, the scalability of Gavel is limited as the mathematical solving procedure is highly complicated and time-consuming. To achieve a higher solution scalability in Gavel, POP [11] decomposes the original complex optimization problem into multiple smaller ones and solves them in parallel. This paper also focuses on solution scalability and fairness via a sampling scheme in Section V-B.

III. System Model and Problem Statement

In this section, we introduce the model of heterogeneous computing clusters for distributed deep learning and then provide the problem formulation. To be specific, the widely used Ring-AllReduce architecture with data parallelism is adopted for distributed training. The major notations used in this paper are summarized in Table I.

A. Heterogeneous Computing Clusters

As shown in Fig. 1, we consider a set of heterogeneous computing clusters comprising multiple workers (denoted by $\mathcal{K} = \{1, \ldots, K\}$), which carry out distributed training to learn deep learning models. The well-known data-parallel training

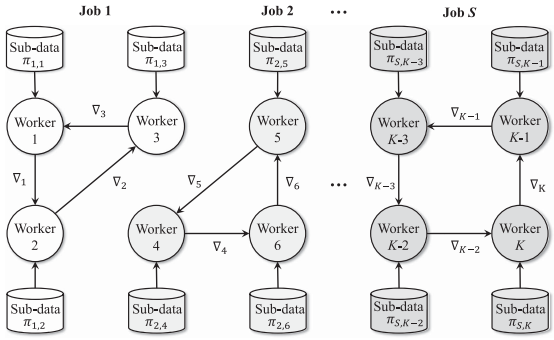


Fig. 1. Framework of distributed deep learning with the Ring-AllReduce architecture in heterogeneous computing clusters.

with the Ring-AllReduce architecture is adopted. As shown in Fig. 1, distributed workers constitute a single ring to aggregate the model gradients without using the central parameter server (PS). Compared with the PS architecture, Ring-AllReduce is more communication efficient as it eliminates the bottleneck in the PS by distributing communication evenly over all participant workers [26]. Many distributed learning frameworks adopt Ring-AllReduce, e.g., NVIDIA Collective Communications Library (NCCL) [27], Gloo [28], and Baidu-AllReduce [29].

The data collected from various sources is aggregated at the workers for deep neural network training. The computing clusters may need to handle several learning jobs simultaneously with various learning goals. Let $\mathcal{S} = \{1, \ldots, S\}$ denote the set of active learning jobs. Each learning job is represented by a tuple $\{\Pi_i, W_i, D_i\}$, where Π_i represents the entire training dataset for job i, W_i represents the global parameters of the deep learning model, D_i represents the size of the global model parameters, and $i \in \mathcal{S}$. The size of gradients is the same as the model size D_i if gradient sparsification or quantization techniques are not applied.

In the training process, each worker collects the training dataset, keeps a local copy of the deep learning model, and sends the calculated local gradient to the next worker. All workers are connected through high-speed Ethernet to ensure the stability and efficiency of communication. The hardware configurations of workers may differ greatly. The training of learning models shows heterogeneous performance behavior across various types of workers due to architectural differences. In heterogeneous computing clusters, how to select appropriate workers (or equivalently job placement) and balance the loads to accelerate the training of learning jobs is the main concern of this paper.

B. Distributed Deep Learning

Gradient descent and its variants are the most widely used methods to train deep neural networks in an iterative manner [26]. Let $\sigma_{i,k} \in \{0, 1\}$ indicate whether worker k is chosen for the distributed training job i or not, yielding 1 if true and 0 if false. Let κ_i denote the set of selected workers for job i (with size $\lvert\kappa_i\rvert = K_i$). As shown in Fig. 1, the training dataset is split among selected workers (denoted by $\pi_{i,k}$) with data parallelism. Using allocated data, each worker trains its local model parameters.

Every training epoch at each worker usually consists of three steps:

1) The loss function $f_{i,k}(W_i^t;\pi_{i,k})$ is calculated using the feed-forward method;

2) The gradients $\nabla f_{i,k}(W_i^t;\pi_{i,k})$ are calculated using the backward propagation method;

3) After all workers finish the calculation of local gradients, the model synchronization begins. Each worker divides its own local gradients into K_i sub-arrays, which are referred to as "chunks". The worker k sends its kth chunk to the next worker, while it receives the $(k-1)$th chunk from the previous worker simultaneously. Then, worker k performs the reduction operation to the received $(k-1)$th chunk and its own $(k-1)$th chunk, and sends the reduced chunk to the next worker. By repeating the receive-reduce-send steps K_i-1 times, each worker obtains a complete reduction of global gradients. The global model can be updated as follows

$$W_i^{t+1} = W_i^t - \eta \cdot \frac{1}{K_i} \cdot \sum_{k\in\kappa_i} \nabla f_{i,k}(W_i^t;\pi_{i,k}), i \in \mathcal{S}, \quad (1)$$

where η is the learning rate. The training in (1) is also known as Bulk Synchronous Parallel (BSP) [30], which synchronizes all updates after each worker goes through its shard of data. Please note that Ring-AllReduce is not suitable for asynchronous communication, e.g., Stale Synchronous Parallel (SSP) [31] and Asynchronous Parallel (ASP) [32] because of the collective communication fashion [26].

Computation Latency and Training Throughput: If learning job i is placed at worker k, the computation latency of one training epoch is

$$l_{i,k}^{[\mathrm{p}]} = \frac{|\pi_{i,k}|}{\rho_{i,k}}, \quad (2)$$

where $\rho_{i,k}$ denotes the number of processed data samples per second at worker k. Typically, the computations of training are tensor-based, which are highly structured with hundreds of thousands of short iterations. In each iteration, samples are packed into a matrix for fast processing. Fig. 2 shows the average computation latency of various learning jobs deployed on different GPU workers over 100 rounds of experiments.[1] Experimental results demonstrate that the computation latency of feed-forward computation and backward propagation is highly predictable and increases almost linearly with the number of assigned samples $|\pi_{i,k}|$. Similar observations can be found in [4]. For each learning job, the computation latency of one training epoch is determined by the straggler in the selected workers with the maximum delay

$$l_i^{[\mathrm{p}]} = \max_{k\in\mathcal{K}}\{l_{i,k}^{[\mathrm{p}]} \cdot \sigma_{i,k}\}. \quad (3)$$

The training throughput of learning job i is given by

$$V_i = \frac{|\Pi_i|}{\max_{k\in\mathcal{K}}\{l_{i,k}^{[\mathrm{p}]} \cdot \sigma_{i,k}\}}, \quad (4)$$

which represents the number of data samples that the learning system can process in a given amount of time.

[1]See Section VI-A for details of the experimental setup.

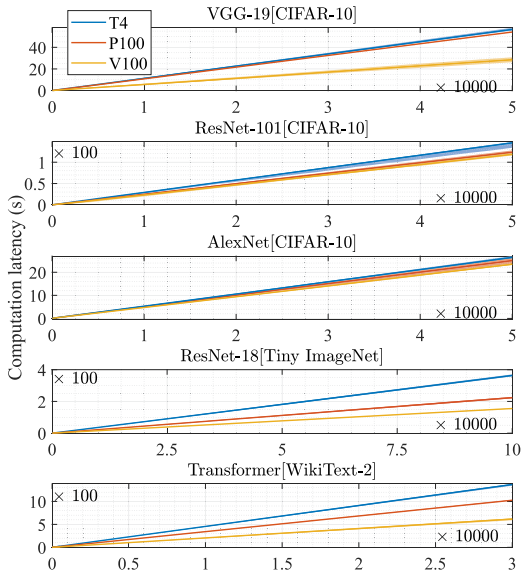


Fig. 2. Relationship between the computation latency and the number of assigned data samples $\pi_{i,k}$ to workers equipped with various types of GPU.

Communication Latency: Each worker sends $2(K_i-1)$ chunks to each of its two neighbors. The first round of K_i-1 chunks received is added to the buffer at the receiving worker, whereas the second round of K_i-1 chunks replaces the values held in the buffer. Let $r_{k,k'}$ denote the data transfer rate between worker k and k', where k, $k' \in \kappa_i$ denote the predecessor and successor workers within a ring communication topology. In Ring-AllReduce, the communication latency is determined by the bottleneck link, i.e.,

$$r_i^{\min} = \min_{k,k'\in\kappa_i}\{r_{k,k'}\}. \quad (5)$$

The communication latency of one training epoch is

$$l_i^{[\mathrm{c}]} = \frac{2(\sum_{k\in\mathcal{K}}\sigma_{i,k}-1)D_i}{r_i^{\min}\cdot\sum_{k\in\mathcal{K}}\sigma_{i,k}}. \quad (6)$$

Please note that the latency of reduce and model update at the worker is omitted due to the low computational complexity. The JCT of learning job i is

$$\Lambda_i = T_i \cdot (l_i^{[\mathrm{p}]} + l_i^{[\mathrm{c}]}), \quad (7)$$

where T_i denotes the number of training epochs. Previous research indicates that gradient descent converges at a rate of $O(1/T_i)$ [8]. Therefore, T_i can be empirically determined as in [8] to achieve the desired training accuracy. If the learning model does not converge to the desired value after T_i epochs, more training epochs will be assigned, which triggers the recomputing of the parallelization policy.

C. Throughput Estimator

To find the optimal policy, an $S \times K$ throughput matrix needs to be maintained, which contains the sample processing throughput $\rho_{i,k}$ of each learning job at each worker. We deploy a throughput estimator, similar to those found in prior work [4],

[33], to predict the performance of distributed training. We could train a few data samples at the worker and use the latency as the initial value to calculate the throughput. Moreover, during the training process, the throughput estimation can be refined on-the-fly as jobs run on different types of workers. After one training epoch, the well-known Exponentially Weighted Moving Average (EWMA) method [34] could be used to refine the value of training throughput.

D. Job Placement and Load Balancing

Given the throughput matrix as input, the computation latency $l_i^{[\mathrm{p}]}$ and the communication latency $l_i^{[\mathrm{c}]}$ of a learning job for one training epoch can be calculated as (3) and (6). Given the number of training epochs T_i and the number of learning jobs S, the heterogeneity-aware job placement and load balancing problem is formulated as

$$\mathbf{P1}: \min \ \Lambda(\sigma_{i,k}, \pi_{i,k}) = \frac{1}{S} \sum_{i \in \mathcal{S}} T_i \cdot (l_i^{[\mathrm{p}]} + l_i^{[\mathrm{c}]}) \tag{8}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{K}} \sigma_{i,k} \cdot |\pi_{i,k}| = |\Pi_i|, \forall i \in \mathcal{S}, \tag{8a}$$

$$\sum_{k \in \mathcal{K}} \sigma_{i,k} > 0, \sigma_{i,k} \in \{0,1\}, \forall i \in \mathcal{S}, \tag{8b}$$

$$\sum_{i \in \mathcal{S}} \sum_{k \in \mathcal{K}} \sigma_{i,k} = K, \tag{8c}$$

$$\sum_{i \in \mathcal{S}} \sigma_{i,k} \leq 1, \forall k \in \mathcal{K}. \tag{8d}$$

The constraints of **P1** ensure that (8a) all data samples are assigned for training; (8b) at least one worker is selected for each learning job to avoid starvation;[2] (8c) all workers can participate in training to fully utilize the computing resources; (8d) a worker can only be assigned to one learning job at a time. In this paper, we assume two learning jobs will not be placed on the same worker for training at the same time. This is because different jobs may have different resource and security requirements; severe and unpredictable performance degradation can be observed among co-located learning jobs [4], [9]. To solve $\mathbf{P1}$, an algorithm must decide 1) how to select the optimal subset of workers from a pool of heterogeneous workers; 2) how many data samples $|\pi_{i,k}|$ shall be assigned to each worker to achieve the desired objective in (8). **P1** can be classified into the resource allocation problem with integer variables, which is known to be NP-hard [35].

IV. Heterogeneity-Aware Scheduler Design

First, let us use a numerical example to show why **P1** cannot be well solved by optimizing the load balancing sequentially after the job placement decision has been found with existing solutions. Then, we introduce the design details of our solution.

[2]This practice ensures that all learning jobs, including smaller ones, receive a minimum allocation of computational resources. It prevents scenarios where a lengthy job might delay a series of shorter jobs, leading to a long JCT. Assigning at least one worker to each learning job is a widely adopted approach [8], which improves scheduling fairness.

A. Motivating Example

We consider a heterogeneous computing cluster containing two T4 and two V100 GPU workers. Two learning jobs selected from Table III, i.e., ResNet-18 trained on Tiny ImageNet and VGG-19 trained on CIFAR-10, are submitted to the computing cluster at the same time for parallel training. The training throughput is

$$\{\rho_{i,k}\}_{S \times K} = \begin{matrix} & \overbrace{}^{\text{T4}} & \overbrace{}^{\text{T4}} & \overbrace{}^{\text{V100}} & \overbrace{}^{\text{V100}} & \\ \text{ResNet-18}\{ & 275 & 275 & 644 & 644 & \} \\ \text{VGG-19}\{ & 884 & 884 & 1754 & 1754 & \}. \end{matrix} \tag{9}$$

The classical Least Attained Service (LAS) policy, used by Tiresias [6], is adopted here to obtain job placement decisions. Let V_i^{Equal} denote the throughput of job i assuming it receives an equal share of computing resources in the cluster. LAS maximizes $\min_{i \in \mathcal{S}} \sum_{k \in \mathcal{K}} \sigma_{i,k} \cdot \rho_{i,k} / V_i^{\text{Equal}}$ to achieve max-min fairness of training throughput across learning jobs. The job placement decisions are ResNet-18: {T4, V100} and VGG-19: {T4, V100}. Given the job placement decisions, load balancing can be achieved if we assign data samples to each selected worker according to the training throughput. Then, the computation latency of a learning job for one training epoch can be calculated using (2) and (3). Considering the number of training epochs T_i given in Table III, the average JCT can be calculated as in (8). In this example, the average JCT is 12,776.8 s if we overlook the communication latency of model synchronization. Through exhaustive search, the optimal job placement decisions ResNet-18: {V100, V100} and VGG-19: {T4, T4} can be obtained with the lowest JCT of 10,592 s. The sequential optimization overlooks the impact of load balancing on training latency when making job placement decisions, introducing a non-negligible performance loss of 20.6%. This motivates us to design better solutions to achieve low JCT.

Generally speaking, we propose to solve $\mathbf{P1}$ by decomposing the problem into sub-problems with the following three stages: 1) through the design of proportional training workload assignment, $\mathbf{P1}$ with two types of variables $\sigma_{i,k}$ and $\pi_{i,k}$ can be simplified into a problem with a single type of variable $\sigma_{i,k}$; 2) rather than directly determining which workers are assigned to which jobs, we categorize the feasible solutions based on the potential partition over the number of selected workers for each learning job, i.e., $\Psi = \{(K_1, \ldots, K_S) | \sum_{i \in \mathcal{S}} K_i = K, K_i \in \mathbb{Z}^+\}$; 3) each sub-problem defined by a feasible solution category in Ψ can be solved by applying the matching markets method [21] to obtain the highest training throughput with theoretical guarantees. At the same time, the near-optimal JCT can be obtained.

B. Proportional Workload Assignment

For learning job $\forall i \in \mathcal{S}$, the computation latency is determined by the straggler in the selected workers with the maximum delay $l_i^{[\mathrm{p}]} = \max_{k \in \mathcal{K}} \{l_{i,k}^{[\mathrm{p}]} \cdot \sigma_{i,k}\}$. The computation latency can be minimized if the workloads can be proportionally assigned based on the worker processing capacity. According to (2), the number of assigned data samples for worker $k \in \kappa_i$ can be

Algorithm 1: Systematic Enumeration for Feasible Solution Categories.

Input: Number of learning jobs S, number of workers K.
Output: Set of feasible solution categories Ψ.
Initialization: $\Psi \leftarrow \varnothing$,
$\{K_1, K_2, \ldots, K_S\} \leftarrow \{K - S + 1, 1, \ldots, 1\}$.
while $\{K_1, \ldots, K_S\} \notin \Psi$ **do**
  Add $\{K_1, \ldots, K_S\}$ to Ψ;
  $K_2 \leftarrow K_2 + 1$ if $K_2 < \hat{K}_2$ else $K_2 \leftarrow 1$;
  for $i = 3$ to S **do**
    if K_{i-1} is set to 1 **then**
      $K_i \leftarrow K_i + 1$ if $K_i < \hat{K}_i$ else $K_i \leftarrow 1$;
    end if
  end for
  $K_1 = K - \sum_{j=2}^{S} K_j$;
end while

obtained from

$$\begin{cases} l_i^{[\mathrm{p}]} = \frac{|\pi_{i,1}|}{\rho_{i,1}} = \ldots = \frac{|\pi_{i,K_i}|}{\rho_{i,K_i}}, \\ \sum_{k \in \mathcal{K}} \sigma_{i,k} \cdot |\pi_{i,k}| = |\Pi_i|. \end{cases} \tag{10}$$

From (10), we have $l_i^{[\mathrm{p}]} \cdot \rho_{i,k} = |\pi_{i,k}|$, $k \in \kappa_i$. Then, by adding them up, we get $l_i^{[\mathrm{p}]} \cdot \sum_{k \in \mathcal{K}} \sigma_{i,k} \cdot \rho_{i,k} = |\Pi_i|$. This means

$$l_i^{[\mathrm{p}]} = \frac{|\Pi_i|}{\sum_{k \in \mathcal{K}} \sigma_{i,k} \cdot \rho_{i,k}}, \tag{11}$$

$$|\pi_{i,k}| = |\Pi_i| \cdot \frac{\rho_{i,k}}{\sum_{k \in \mathcal{K}} \sigma_{i,k} \cdot \rho_{i,k}}, \tag{12}$$

$$V_i = \sum_{k \in \mathcal{K}} \sigma_{i,k} \cdot \rho_{i,k}. \tag{13}$$

The desired objective in (8) can be rewritten as

$$\Lambda(\sigma_{i,k}) = \sum_{i \in \mathcal{S}} \frac{T_i}{S} \left(\frac{2(\sum_{k \in \mathcal{K}} \sigma_{i,k} - 1) D_i}{r_i^{\min} \cdot \sum_{k \in \mathcal{K}} \sigma_{i,k}} + \frac{|\Pi_i|}{\sum_{k \in \mathcal{K}} \sigma_{i,k} \cdot \rho_{i,k}} \right). \tag{14}$$

Through the proportional workload assignment, we only need to determine which workers shall be selected $\sigma_{i,k} \in \{0, 1\}$. Then, the load balancing decision can be obtained based on (12).

C. Feasible Solutions Categorization

As shown in Algorithm 1, the set of all feasible solution categories Ψ can be derived through systematic enumeration on Constraints (8b), (8c), and (8d), which determines how many positive integer solutions exist. Assuming that the values of $\{K_{i+1}, \ldots, K_S\}$ are given, the maximum value that K_i can be assigned is

$$\hat{K}_i = K - (i - 1) - \sum_{j=i+1}^{S} K_j, \tag{15}$$

where the term $i - 1$ ensures the constraint $K_j > 0$ is still satisfied for learning job $j \in \{1, \ldots, i-1\}$. Initially, $\{K_1, K_2, \ldots, K_S\} = \{K - S + 1, 1, \ldots, 1\}$ is a feasible solution.[3] The value of K_2 is continuously increased by 1 until $K_2 = \hat{K}_2$. Then, K_2 is set to 1 in the next phase. For any other learning job $i \in \{3, \ldots, S\}$, if K_{i-1} is set from $\hat{K}_{i-1}$ to 1, K_i is incremented by 1. If $K_i = \hat{K}_i$, K_i will also be set to 1 next. At last, K_1 is set to $K - \sum_{j=2}^{S} K_j$, ensuring (8c) always holds. We use a simple example to show the systematic enumeration process. When $K = 5$ and $S = 3$, $|\Psi| = 6$ feasible solution categories, i.e., $\{3, 1, 1\}$, $\{2, 2, 1\}$, $\{1, 3, 1\}$, $\{2, 1, 2\}$, $\{1, 2, 2\}$ and $\{1, 1, 3\}$, are sequentially added. Theoretical analysis in previous work shows the set size $|\Psi| = \binom{K-1}{S-1}$ for this stars and bars problem [36]. As we need S steps to generate a feasible solution category, the computational complexity of Algorithm 1 is $O(S \cdot \binom{K-1}{S-1})$.

For a given set $\{K_1, \ldots, K_S\} \in \Psi$ and the average JCT as calculated in (14), the subproblem of $\mathbf{P1}$ is

$$\mathbf{P2}: \ \min \ \Lambda(\sigma_{i,k}) = \sum_{i \in \mathcal{S}} \frac{T_i}{S} \left(\frac{2(K_i - 1) D_i}{r_i^{\min} K_i} + \frac{|\Pi_i|}{\sum_{k \in \mathcal{K}} \sigma_{i,k} \rho_{i,k}} \right) \tag{16}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{K}} \sigma_{i,k} = K_i, \forall i \in \mathcal{S}. \tag{17}$$

For all feasible solution categories in Ψ, we have $\binom{K-1}{S-1}$ subproblems in total. Later, we will show how the proposed sampling scheme can greatly reduce the number of examined categories with a small performance loss in Section V-B.

D. Matching Markets

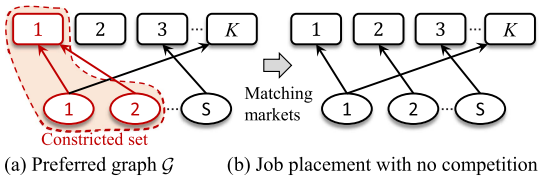
The difficulty of solving $\mathbf{P2}$ lies in the nonlinearity and non-convexity of the objective function (16). By observing (13) and (16), we know that each learning job prefers to select workers with higher training throughput $V_i = \sum_{k \in \mathcal{K}} \sigma_{i,k} \cdot \rho_{i,k}$ for a lower JCT. This means that high training throughput is positively correlated with low JCT. Therefore, we map $\mathbf{P2}$ to the following optimization problem

$$\mathbf{P3}: \max \ V(\sigma_{i,k}) = \sum_{i \in \mathcal{S}} \sum_{k \in \mathcal{K}} \sigma_{i,k} \cdot \rho_{i,k}$$
$$\text{s.t.} \quad (17), \tag{18}$$

which maximizes the training throughput of learning jobs. We propose an optimal scheme to solve $\mathbf{P3}$ with theoretical guarantees. Given the throughput matrix and the desired objective in (14), a K-dimensional array $\{\rho_{i,1}, \ldots, \rho_{i,K}\}$ is maintained for each learning job.

As shown in Fig. 3(a), a preferred graph $\mathcal{G}$ is constructed with function `preferred_graph`, which selects K_i workers with larger values in $\{\rho_{i,1}, \ldots, \rho_{i,K}\}$. In $\mathcal{G}$, different learning jobs may compete for the same worker, but each worker only processes one learning job at a time. Then, the constricted set $\{\mathcal{K}_c, \mathcal{S}_c\}$ is created using function `constricted_set`, where $\mathcal{K}_c$ represents the set of competed workers, and $\mathcal{S}_c$ represents the set of competing jobs.

[3]We assume $K \geq S$ to ensure that at least one worker will be assigned to each learning job.



(a) Preferred graph $\mathcal{G}$ (b) Job placement with no competition

Fig. 3. Illustration of matching markets.

Then, a scheme based on matching markets is proposed to tackle the competition problem. The pseudo code is shown in Algorithm 2. The workers and learning jobs are considered as "sellers" and "buyers", respectively. To obtain the optimal matching between "sellers" and "buyers", the prices of all workers $\{p_1, \ldots, p_K\}$ are introduced. Initially, we set $\{p_1, \ldots, p_K\} = \{0, \ldots, 0\}$. The basic idea of matching markets is to gradually increase the prices of competed workers $k \in \mathcal{K}_c$ until the constricted set is empty. Next, we show how to set the price p_k in detail. The valuation array Γ_i is initialized as the payoff array, i.e.,

$$\Gamma_i \leftarrow \{\rho_{i,1}, \ldots, \rho_{i,K}\}, \forall i \in \mathcal{S}. \tag{19}$$

Let V_i denote the payoff of assigning all preferred workers (including the competed worker k) to job $i \in \mathcal{S}_c$, i.e.,

$$V_i = \texttt{sum\_top}(\Gamma_i, K_i), \tag{20}$$

where function `sum_top` represents the sum of the largest K_i elements in array Γ_i. If worker k is not assigned to job i, the payoff is

$$V_i^k = \texttt{sum\_top}(\Gamma_i \setminus \{\Gamma_{i,k}\}, K_i). \tag{21}$$

Then, the price of worker k is updated as

$$p_k \leftarrow p_k + \max\{1, \max_{i \in \mathcal{K}_c}\{V_i - V_i^k\}\}, \tag{22}$$

which means if $\max_{i \in \mathcal{K}_c}\{V_i - V_i^k\} = 0$, p_k is increased by the unit price 1 to ensure the pricing process will continue. The corresponding payoff of assigning worker k to job i is updated as

$$\Gamma_{i,k} \leftarrow \rho_{i,k} - p_k, \forall i \in \mathcal{S}, \forall k \in \mathcal{K}. \tag{23}$$

Unlike the pricing scheme in [21] which always increases the price by one unit, the proposed pricing scheme ensures that worker k will only be assigned to one job $i = \arg\max\{V_i - V_i^k\}$ in each `while` loop. This improves the efficiency of the solution. The above process is repeated until the constricted set is empty. Then, the decision $\sigma_{i,k}$ for one feasible solution category can be obtained from the updated preferred graph $\mathcal{G}$ with no competition among jobs. For a feasible solution category $\{K_1, \ldots, K_S\} \in \Psi$, the highest training throughput can be obtained. By considering all feasible solution categories in Ψ, the near-optimal JCT can be obtained.

For the example in Section IV-A with two jobs and four workers, three feasible solution categories, i.e., {3, 1}, {2, 2}, and {1, 3}, are considered. Table II shows the numerical results yielded by Algorithm 2. The results indicate that if the learning job with higher computational complexity, i.e., ResNet-18, is assigned more computational resources, the average JCT is more likely to be reduced. Algorithm 2 examines all feasible solution categories, thereby ensuring that the scenario where ResNet-18 is assigned more workers is considered to approximate the optimal solution better. In this case, the job placement decisions ResNet-18: {T4, T4, V100} and VGG-19: {V100} with the JCT of 11,225.8 s are selected as the solution. When compared with the optimal solution, the performance loss in terms of JCT is only 6.0%.

TABLE II
NUMERICAL RESULTS OF THE EXAMPLE IN SECTION IV-A YIELDED BY HETEROGENEITY-AWARE SCHEDULER

Ψ	ResNet-18	VGG-19	throughput	avg. JCT
{3, 1}	{T4, T4, V100}	{V100}	{1,194, 1,754}	11,225.8 s
{2, 2}	{T4, T4}	{V100, V100}	{550, 3,508}	19,607.1 s
{1, 3}	{T4}	{T4, T4, V100}	{275, 4,392}	37,502.1 s
Optimal	{V100, V100}	{T4, T4}	{1,288, 1,768}	10,592 s

Algorithm 2: Heterogeneity-Aware Scheduler.

Input: Set of feasible solution categories Ψ.
Output: $\sigma_{i,k}^*$, $\pi_{i,k}^*$.
Initialization: $p_k, \sigma_{i,k}, \sigma_{i,k}^* \leftarrow 0, \Lambda^* \leftarrow \inf, \forall i \in \mathcal{S}$, $\forall k \in \mathcal{K}$.
1: **for** feasible solution category $\{K_1, \ldots, K_S\} \in \Psi$ **do**
2: $\Gamma_i \leftarrow \{\rho_{i,1}, \ldots, \rho_{i,K}\}, \forall i \in \mathcal{S}$;
3: $\mathcal{G} \leftarrow \texttt{preferred\_graph}(\{\Gamma_i, K_i\}, \forall i \in \mathcal{S})$;
4: $\{\mathcal{K}_c, \mathcal{S}_c\} \leftarrow \texttt{constricted\_set}(\mathcal{G})$;
5: **while** $\{\mathcal{K}_c, \mathcal{S}_c\} \neq \varnothing$ **do**
6: Select $k \in \mathcal{K}_c$;
7: **for** $i \in \mathcal{S}_c$ **do**
8: $V_i \leftarrow \texttt{sum\_top}(\Gamma_i, K_i)$;
9: $V_i^k \leftarrow \texttt{sum\_top}(\Gamma_i \setminus \{\Gamma_{i,k}\}, K_i)$;
10: **end for**
11: $p_k \leftarrow p_k + \max\{1, \max\{V_i - V_i^k\}\}$;
12: $\Gamma_{i,k} \leftarrow \rho_{i,k} - p_k, \forall i \in \mathcal{S}$;
13: $\mathcal{G} \leftarrow \texttt{preferred\_graph}(\{\Gamma_i, K_i\}, \forall i \in \mathcal{S})$;
14: $\{\mathcal{K}_c, \mathcal{S}_c\} \leftarrow \texttt{constricted\_set}(\mathcal{G})$;
15: **end while**
16: Obtain $\sigma_{i,k}$ based on $\mathcal{G}$ and calculate Λ based on (8);
17: If $\Lambda < \Lambda^*$, $\sigma_{i,k}^* \leftarrow \sigma_{i,k}$, $\Lambda^* \leftarrow \Lambda$, $\forall i \in \mathcal{S}, \forall k \in \mathcal{K}$;
18: **end for**
19: Obtain load balancing decision $\pi_{i,k}^*$ based on (12);

E. Theoretical Analysis

First, the convergence of our design is proved.

Theorem 1: The pricing process in Algorithm 2 must come to an end within a limited number of steps.

Proof: During the pricing process, each worker is either assigned to a learning job or remains idle. The `while` loop in Algorithm 2 (Lines 5–15) only ends when the constricted set is empty; otherwise, it continues with increased prices $p_k \geq 0$, $k \in \mathcal{K}$. We define

$$P = \sum_{i \in \mathcal{S}} V_i + \sum_{k \in \mathcal{K}} p_k. \tag{24}$$

Then, we prove that P has the following two properties:

1) $P \geq 0$ always holds during the pricing process. Let $\{i_1, k_1\}$ be an arbitrary assignment in the preferred graph $\mathcal{G}$.

- If $\{i_1, k_1\} \notin \{\mathcal{K}_c, \mathcal{S}_c\}$, no other jobs are competing for worker k_1. For $\{i_1, k_1\}$, we have

$$\Gamma_{i_1,k_1} + p_{k_1} = \rho_{i_1,k_1} \geq 0. \tag{25}$$

- If $\{i_1, k_1\} \in \{\mathcal{K}_c, \mathcal{S}_c\}$, other jobs are competing for worker k_1. Let i_2 be an arbitrary competitor, $\{i_2, k_1\} \in \{\mathcal{K}_c, \mathcal{S}_c\}$. For an arbitrary worker k_2 which is not in the preferred graph $\mathcal{G}$, we have

$$\rho_{i_2,k_1} - p_{k_1} \geq \rho_{i_2,k_2} - p_{k_2}. \tag{26}$$

For i_2 and k_2, we have

$$\Gamma_{i_2,k_2} + p_{k_2} = \rho_{i_2,k_1} - p_{k_1} + p_{k_2} \geq \rho_{i_2,k_2} \geq 0. \tag{27}$$

As $K \geq S$, we can always find a unique unassigned worker for each competitor with $\Gamma_{i_2,k_2} + p_{k_2} \geq 0$. For all other unassigned workers, $p_k \geq 0$, $k \in \mathcal{M}$. According to (25) and (27), $P \geq 0$ always holds.

2) P decreases with the increase of prices: The prices are initialized as $\{p_1, \ldots, p_K\} = \{0, \ldots, 0\}$. Initially, we have

$$P_{\max} = \sum_{i \in \mathcal{S}} \texttt{sum\_top}(\{\rho_{i,1}, \ldots, \rho_{i,K}\}, K_i). \tag{28}$$

Then, in each round of pricing, the worker in the constricted set raises its price by at least one unit. Please note that multiple jobs are competing for the worker. According to (20) and (23), $\sum_{i \in \mathcal{S}} V_i - \sum_{k \in \mathcal{K}} p_k$ decreases by at least one unit in each round of pricing. To sum up, the pricing scheme starts with $P = P_{\max}$, and P cannot drop below 0. So the pricing process must end within $P_{\max}$ steps.

Theorem 2 verifies the optimality of our design.

Theorem 2: Algorithm 2 yields the highest training throughput.

Proof: We prove the optimal solution to **P3** can be obtained for each feasible solution category. Constraint (8c) ensures all workers are assigned to learning jobs. When the pricing process ends with $\{\mathcal{K}_c, \mathcal{S}_c\} = \varnothing$, each worker is assigned to a learning job. As shown in (29), let $\{i_1, k_1\}$ and $\{i_2, k_2\}$ denote two randomly selected job placement decisions.

$$\begin{aligned} &\{\ldots, \underline{\rho_{i_1,k_1}}, \ldots, \rho_{i_1,k_2}, \ldots\}, \\ &\{\ldots, \rho_{i_2,k_1}, \ldots, \underline{\rho_{i_2,k_2}}, \ldots\}. \end{aligned} \tag{29}$$

To verify the optimality, we need to prove that interchanging any two pairs of job placement decisions cannot further increase the total valuations V in (14), i.e.,

$$\rho_{i_1,k_1} + \rho_{i_2,k_2} \geq \rho_{i_1,k_2} + \rho_{i_2,k_1}. \tag{30}$$

With the pricing method in Algorithm 2, we have

$$\begin{cases} \rho_{i_1,k_1} - p_{k_1} \geq \rho_{i_1,k_2} - p_{k_2}, \\ \rho_{i_2,k_2} - p_{k_2} \geq \rho_{i_2,k_1} - p_{k_1}. \end{cases} \tag{31}$$

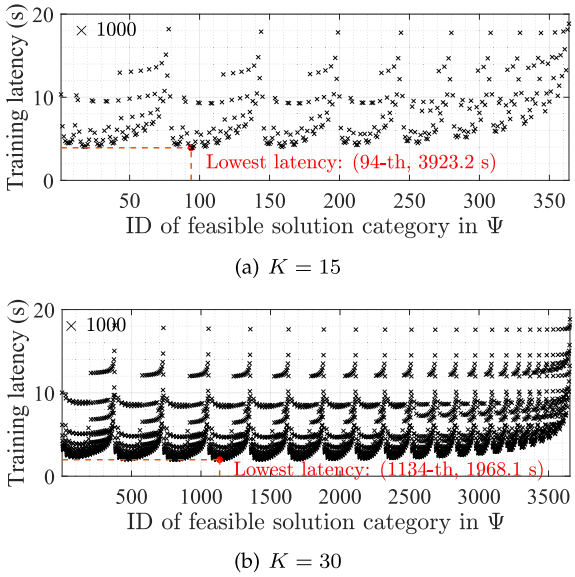


Fig. 4. Average JCT of learning jobs for all feasible solution categories in Ψ without job prioritizing.

This means that (30) holds. We conclude that the optimal solution can be obtained for each feasible solution category in Ψ.

Proposition 1: The computational complexity of Algorithm 2 is $O(S \cdot K \cdot P_{\max} \cdot \binom{K-1}{S-1})$.

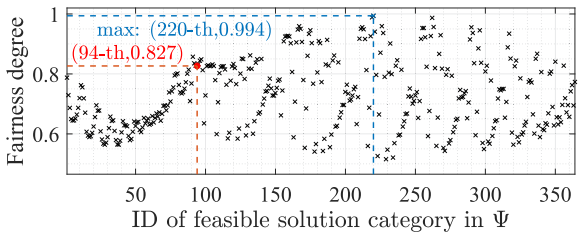
Proof: The computational complexity of calculating the K-dimensional array Γ_i for all S learning jobs is $O(S \cdot K)$ (Line 2). Then, to obtain the preferred graph, all training throughput arrays are sorted via the radix sort algorithm. The sorting complexity is $O(S \cdot K)$ (Line 3). Then, all workers need to be considered to determine the constricted set with the complexity of $O(K)$ (Line 4). As all learning jobs may compete for a worker, the calculation of the worker price needs S steps at most (Lines 6–11). Furthermore, the preferred graph and the constricted set are updated with the complexity of $O(K + S \cdot K)$ (Lines 12–14). As discussed above, the `while` loop is performed for $P_{\max}$ times at most. Furthermore, the worker assignment (Lines 17–18) needs K steps at most. The optimal assignment for a feasible solution category needs $(S \cdot K + S + K) \cdot P_{\max} + 2S \cdot K + 2K$ steps at most. The computational complexity for a feasible solution category is $O(S \cdot K \cdot P_{\max})$. Considering all feasible solution categories in Ψ, the computational complexity of Algorithm 2 is $O(S \cdot K \cdot P_{\max} \cdot \binom{K-1}{S-1})$.

V. Fair Scheduling With Low Complexity

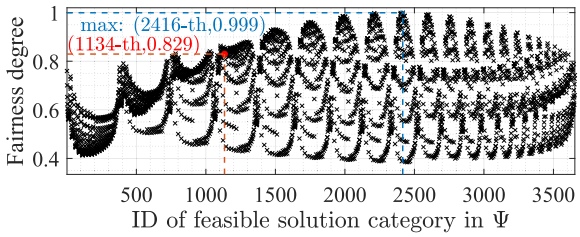
Algorithm 2 has the limitations of 1) high computational complexity, and 2) ignoring the fairness of scheduling among learning jobs. In this section, we analyze the fairness of Algorithm 2 and then propose to improve the scalability and fairness of our design through a sampling scheme.

A. Fairness Analysis

Algorithm 2 yields the placement decision for each feasible solution category $\{K_1, \ldots, K_S\} \in \Psi$. Fig. 4 shows the



(a) $K = 15$



(b) $K = 30$

Fig. 5. Fairness degrees of all feasible solution categories in Ψ without job prioritizing.

average JCT of all feasible solution categories. We consider $S = 4$ learning jobs trained by $K = 15$ and $K = 30$ workers, respectively.[4] In Fig. 4, the IDs of feasible solution categories are sequentially generated by Algorithm 1. For example, with $S = 4$ and $K = 15$, we have the first feasible solution category (ID = 1) $\{K_1, \ldots, K_S\} = \{11, 1, 1, 1\}$ and the last feasible solution category (ID = 364) $\{K_1, \ldots, K_S\} = \{1, 1, 1, 11\}$. Then, let us examine the fairness degree of the placement decision yielded by Algorithm 2 for all feasible solution categories.

The fairness can be achieved if system resources are equally shared among all learning jobs. With the ideally equal resource allocation, the JCT of learning job i is

$$\Lambda_i^{[e]} = \sum_{i \in \mathcal{S}} T_i \left(\frac{2(K/S-1)D_i}{r \cdot K/S} + \frac{|\Pi_i|}{S \cdot \sum_{k \in \mathcal{K}} \rho_{i,k}} \right). \quad (32)$$

Recall that Λ_i denote the JCT of learning job i yielded by Algorithm 2 for a feasible solution category $\{K_1, \ldots, K_S\} \in \Psi$. Then, $\Lambda_i/\Lambda_i^{[e]}$ represents the deviation degree of scheduling from the equal resource allocation for job i. According to Jain's fairness index [37], the fairness degree for a feasible solution category is

$$F(K_1, \ldots, K_S) = \frac{(\sum_{i \in \mathcal{S}} \Lambda_i/\Lambda_i^{[e]})^2}{S \cdot \sum_{i \in \mathcal{S}} (\Lambda_i/\Lambda_i^{[e]})^2}. \quad (33)$$

The result ranges from $1/S$ (the worst case) to 1 (the best case) and reaches its maximum when the resources are equally allocated ($\forall \Lambda_i/\Lambda_i^{[e]} = 1$). Fig. 5 shows the fairness degrees of all feasible solution categories. The range of fairness degrees is $[0.527, 0.994]$ for $K = 15$ and $[0.384, 0.999]$ for $K = 30$. Recall that for a feasible solution category $\{K_1, \ldots, K_S\} \in \Psi$, K_i denotes the number of assigned workers for job i. Algorithm 2 considers all feasible solution categories. In heterogeneous computing clusters, we can always find a feasible solution category that achieves nearly equal resource allocation. This explains why some feasible solution categories produce high scheduling fairness, as in Fig. 5.

[4] See Section VI-A for details of the experimental setup.

Algorithm 3: Job Prioritizing for Sampling.

Input: $\rho_{i,k}$, Ψ.
Output: $\sigma_{i,k}^*$, $\pi_{i,k}^*$.
1: Sort jobs $\mathcal{S}$ based on $\frac{T_i \cdot |\Pi_i|}{S \cdot \sum_{k \in \mathcal{K}} \rho_{i,k}}$ in ascending order;
2: Select N samples uniformly from set $\Psi[\alpha\binom{K-1}{S-1}, \binom{K-1}{S-1}]$, $\alpha \in (0, 1)$;
3: Invoke Algorithm 2 on N samples to calculate the JCT Λ_n and the fairness degree F_n, $n \in \mathcal{N}$;
4: Select the sample $n \leftarrow \arg\max\{\beta \frac{\min_{n \in \mathcal{N}}\{\Lambda_n\}}{\Lambda_n} + (1-\beta)F_n\}$ as the solution and obtain near-optimal decisions $\sigma_{i,k}^*$, $\pi_{i,k}^*$;

B. Sampling for Low Complexity and Fairness

The computational complexity of the proposed scheme is mainly determined by the total number of feasible solution categories $|\Psi| = \binom{K-1}{S-1}$, which increases rapidly with more workers and learning jobs. For example, we have $\binom{14}{3} = 364$ and $\binom{29}{3} = 3,654$. Intuitively, to obtain near-optimal decisions, we do not need to consider all feasible solution categories in Ψ. In this way, the computational complexity is reduced with a smaller set size.

As shown in Fig. 4, a rough and changing periodicity of the incurred JCT can be observed.[5] The main reason for the periodicity is that resources are fungible or substitutable (i.e., learning jobs can make similar progress using different resources). For example, with $K = 15$, feasible solution categories $\{6, 6, 2, 1\}$ (ID = 18) and $\{5, 5, 2, 3\}$ (ID = 159) incur similar average JCT of 4,047.59 s and 4,046.35 s. The periodicity motivates the use of sampling for low computational complexity. If we randomly select a set of samples $\mathcal{N}$ (with size $|\mathcal{N}| = N$) from Ψ and choose the sample with the lowest JCT as the solution, non-negligible performance loss can be observed. Experimental results in Section VI-B show that when $N = 20$, the average JCT is increased by 7.65% and 10.43% for $K = 15$ and 30, respectively.

For better approximation, the Job Prioritizing for Sampling scheme is proposed in Algorithm 3. In Ψ, the values of $\{K_1, \ldots, K_S\}$ change from $\{K - S + 1, \ldots, 1\}$ to $\{1, \ldots, K - S + 1\}$. The value of K_i in the rear part of $\{K_1, \ldots, K_S\}$ gradually increases. The value $\frac{T_i \cdot |\Pi_i|}{\sum_{k \in \mathcal{K}} \rho_{i,k}}$ denotes the computational latency if all workers are assigned to job i, which

[5] We believe the characteristic of periodicity remains applicable to other resource allocation problems, irrespective of the specific experimental configurations. We test the performance of feasible solutions categorization and matching markets to other resource allocation problems, e.g., caching in distributed storage systems [38]. Similar periodicity and "dense" structure can be observed. This indicates that our design might be a potential solution to solve other resource allocation problems.

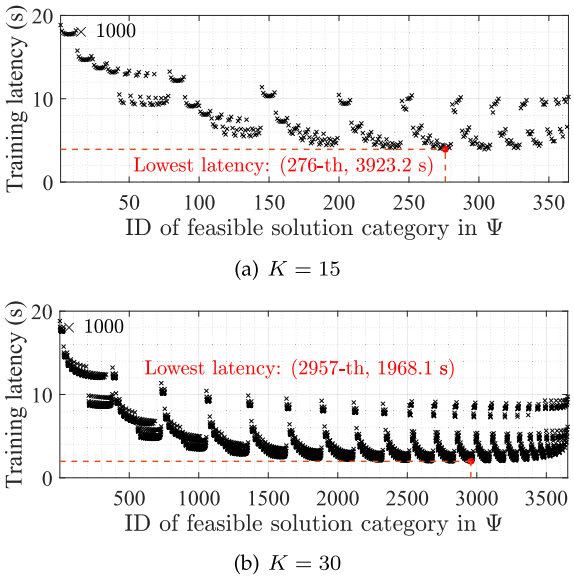


Fig. 6. Average JCT of learning jobs for all feasible solution categories in Ψ with job prioritizing.

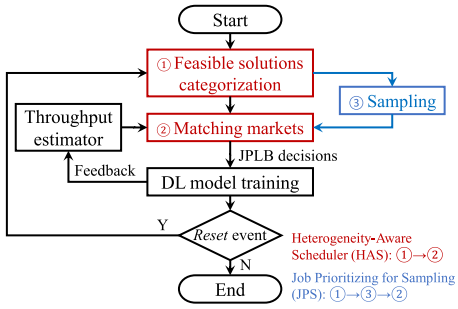


Fig. 7. Workflow of our design.

TABLE III
DEEP LEARNING JOBS USED IN THE EXPERIMENTS

Model	Dataset	Dataset size (# of examples)	Training epochs	Validation accuracy
ResNet101	CIFAR-10 [45]	50,000	200	80.04%
ResNet18	Tiny ImageNet [46]	100,000	200	60.68%
VGG19	CIFAR-10 [45]	50,000	200	93.63%
AlexNet	CIFAR-10 [45]	50,000	100	81.13%
Transformer	WikiText-2 [47]	36,718	3	98.91%

could be used to estimate its computational complexity. All jobs $\mathcal{S}$ are sorted based on their value in ascending order. Then, for a feasible solution category in the rear part of Ψ, jobs with higher computational complexities are assigned more workers, which helps reduce the JCT. Fig. 6 shows the average JCT after job prioritizing; a trend of decline can be observed. Therefore, we only select N samples in the rear part of Ψ, i.e., set $\Psi[\alpha\binom{K-1}{S-1}, \binom{K-1}{S-1}]$, $\alpha \in (0,1)$. Experimental results show that when $N = 20$ and $\alpha = 0.7$, the average JCT is increased by 2.95% and 5.10% for $K = 15$ and 30, respectively. More samples help improve the performance of approximation.

Another observation we get from Figs. 4 and 5 is that the feasible solution category with the lowest JCT may not yield the highest fairness at the same time. For example, with $K = 15$, the 94th feasible solution category $\{6, 5, 2, 2\}$ incurs the lowest JCT of 3,948.9 s with the fairness degree of 0.824. Let Λ_n and F_n denote the incurred average JCT and fairness degree of feasible solution category n, $n \in \mathcal{N}$. To strike a balance between JCT and fairness, we will select the sample

$$n \leftarrow \arg\max \left\{ \beta \frac{\min_{n \in \mathcal{N}}\{\Lambda_n\}}{\Lambda_n} + (1-\beta) F_n \right\}, \tag{34}$$

as the solution for near-optimal decisions, $\beta \in [0, 1]$. When $\beta = 1$, we focus on low JCT. When $\beta = 0$, we focus on high fairness. As the matching markets method needs $O(S \cdot K \cdot P_{\max})$ steps to finish, the computational complexity of Algorithm 3 is reduced to $O(S \cdot K \cdot P_{\max} \cdot N)$.

C. Implementation

Let us integrate all the proposed schemes and show how they could be implemented in practice. The design workflow is illustrated in Fig. 7. *Heterogeneity-Aware Scheduler (HAS)*: Given the numbers of active jobs S and workers K, the set of feasible solution categories Ψ can be derived through the systematic enumeration in Algorithm 1. Using Ψ and the throughput matrix as input, the placement decision can be obtained based on matching markets in Algorithm 2. *Job Prioritizing for Sampling (JPS)*: Based on the design of HAS, the sampling scheme is applied to construct the sparsification of feasible solution categories, reducing the computational complexity with improved scheduling fairness. Moreover, the throughput matrix is refined on-the-fly based on the feedback from DL model training. The parallelization policy will be recomputed when a *reset* event occurs, e.g., job arrival or completion, worker failure, etc. For job completion, if the learning model does not converge to the desired value after T_i epochs, more epochs are assigned for a new round of training.

VI. PERFORMANCE EVALUATION

We deploy a testbed of heterogeneous deep learning cluster hosted on Compute Canada [39]. Our solutions are implemented in Python to generate the parallelization policy for real-world deep learning applications.

A. Experiment Setup

Testbed: We built a heterogeneous deep learning cluster containing 15 and 30 workers on Compute Canada. Each worker is equipped with one of three kinds of GPU cards, i.e., NVIDIA Tesla V100, P100, or T4. Under default settings, the ratio of numbers for the three kinds of workers is 1:1:1. Subsequently, we employ data parallelism to train real-world deep learning applications as listed in Table III, utilizing the DistributedDataParallel framework provided by PyTorch [44]. These training jobs are submitted and executed on the Compute Canada infrastructure, with the management facilitated by the Slurm Workload Manager [40].

Simulator: To evaluate system scalability, we use the data of training jobs collected from the testbed, i.e., the training

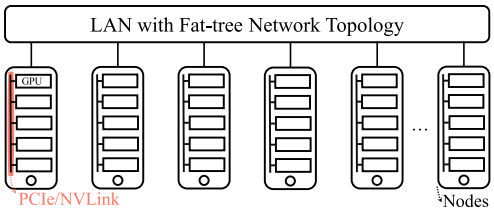


Fig. 8. Topology of the simulated learning system.

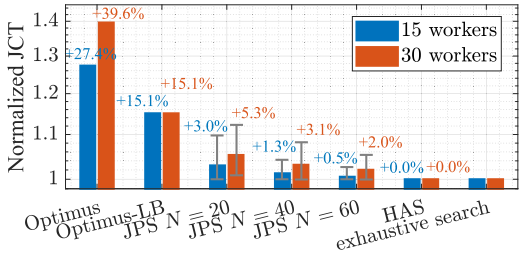


Fig. 9. Normalized JCT (w.r.t. exhaustive search) of various schemes.

throughput of various workers, to simulate a learning system with more workers. Unless otherwise specified, the hardware configuration follows the same distribution as the 15-worker scenario. The heterogeneity in the GPU-oriented communication network is also considered as the data transmission rates within a single node (via PCIe or NVLink) and across nodes (via LAN) could be different. As shown in Fig. 8, multiple GPUs accommodated within a node are interconnected using PCIe or NVLink links [41]. We assume that every computing node is equipped with 5 GPU workers, which is below the upper limit of GPUs that can be directly connected [41]. The GPU-to-GPU bandwidth inside a node is set to 300 Gbps. Then, all computing nodes are interconnected with switches arranged in a multi-tier Fat-tree network topology. This is because traditional data center clusters are the leading architectures to support DNN training, providing uniform bandwidth of 10 Gbps between node pairs [18], [42], [43].

Learning Jobs: We evaluate the performance of our design with five well-known deep learning applications imported from PyTorch [44]. The five deep learning jobs can be classified into two types, i.e., CNN and Transformer. The features of five learning jobs, the used datasets for training, and the number of training epochs T_i are shown in Table III. The validation accuracy after T_i epochs of training is also provided. Under default settings, $S = 4$ learning jobs are considered. Then, in Section VI-B1, we evaluate the scalability of our design by incrementally increasing S from 3 to 5.

Parameter Setting: Under default settings, we configure α and β to be 0.7 and 1.0, respectively, as these values result in the lowest JCT for the proposed scheme JPS. The impact of other parameter settings is assessed in Section VI-B1.

Baselines: Four other schemes are introduced as baselines. The first is *Optimus* [8]—It is a greedy method designed to solve the learning job placement problem. First, each learning job is allocated with one worker to avoid starvation. Then, all jobs are sorted according to their marginal gains of reduced JCT with added workers. *Optimus* iteratively selects the job with the largest marginal gain and adds one worker to it. Please note that load balancing is not considered in [8], i.e., the training dataset is divided into equal-sized parts to feed the workers. By using heuristics, *Optimus* is highly efficient with the computational complexity of $O(S \cdot K)$. The second is *Optimus* with Load Balancing (*Optimus*-LB)—For a fair performance comparison, we extend *Optimus* with the Proportional Workload Assignment scheme designed in Section IV-A to accommodate the worker heterogeneity. The third is *exhaustive search*—We consider all feasible worker selection decisions to find the global optimal solution, minimizing the training latency of all learning tasks. Considering the heterogeneity of workers, *exhaustive search* also proportionally assigns training workloads to the selected workers for a fair performance comparison. The time-consuming *exhaustive search* scheme is used as a lower bound to evaluate the performance of our design.

The three baselines above are static policies, which precompute placement decisions at the beginning of job submission. The decisions will not change during the training process. Similar to the proposed HAS and JPS schemes, the static policies can be extended to dynamic policies by considering *reset* events. For a fair performance comparison with the dynamic policies, *Gavel* [4] is introduced as the fourth baseline since the placement decisions of these schemes will be recomputed occasionally during the training process. *Gavel* predefines the number of requested GPUs scale_factor$_i$ for each learning job $i \in \mathcal{S}$, and calculates the fraction of wall-clock time a learning job should spend on each type of GPU. Then, *Gavel* uses a round-based scheduling scheme to ensure that the average resource allocation each learning job receives over multiple rounds is close to the computed allocation. Please note that *Gavel* only supports distributed training over a single type of GPU in a round, although different types of GPUs may be selected over different rounds. Therefore, the training dataset is divided into equal-sized parts for load balancing in a round.

B. Evaluation Results

1) Without Reset *Events:* Figs. 9–14 compare the performance of various schemes without considering *reset* events first. All four schemes generate static policies at the beginning of job submission. Fig. 9 shows the normalized JCT with regard to the lowest value yielded by *exhaustive search*. Note that *exhaustive search* incurs unacceptable long algorithm running time (ART), as shown in Table IV. By using heuristics with no performance guarantee, *Optimus*-LB incurs performance losses of 15.1% for 15-worker and 30-worker scenarios, with negligible computational overheads.

In contrast, the proposed HAS scheme classifies all feasible solutions into different categories and uses matching markets for decision making, which incurs the same average JCT as *exhaustive search* but with a much lower ART. The computational complexity of HAS can be further reduced via sampling without incurring much performance loss. Fig. 9 illustrates the average JCT of the proposed JPS scheme over 100 rounds of execution and its 90% confidence interval. Compared with

TABLE IV
THE AVERAGE ALGORITHM RUNNING TIME (ART) OF VARIOUS SCHEMES

Scheme		*Optimus*	*Optimus-LB*	$\|\Psi\|$	JPS $N=20$	JPS $N=40$	JPS $N=60$	HAS	*exhaustive search*
$S=4$	$K=15$	0.96 ms	0.99 ms	364	0.07 s	0.14 s	0.22 s	1.07 s	47.89 s
	$K=30$	1.99 ms	1.99 ms	3,654	0.38 s	0.74 s	1.13 s	55.54 s	2.21 h
$K=15$	$S=3$	0.91 ms	0.92 ms	91	0.04 s	0.06 s	0.06 s	0.17 s	1.55 s
	$S=5$	1.01 ms	1.07 ms	1,001	0.06 s	0.12 s	0.19 s	24.34 s	127.36 s

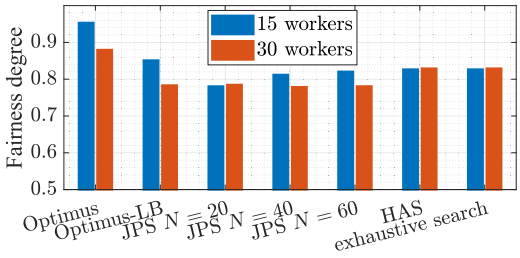


Fig. 10. Fairness degree of various schemes.

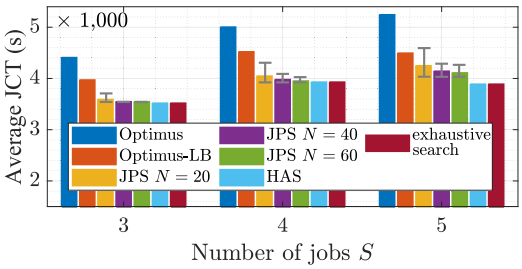


Fig. 11. Average JCT of various schemes with the increased number of jobs S.

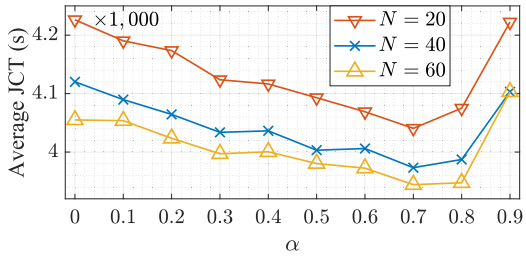


Fig. 12. Impact of α with $K=15$.

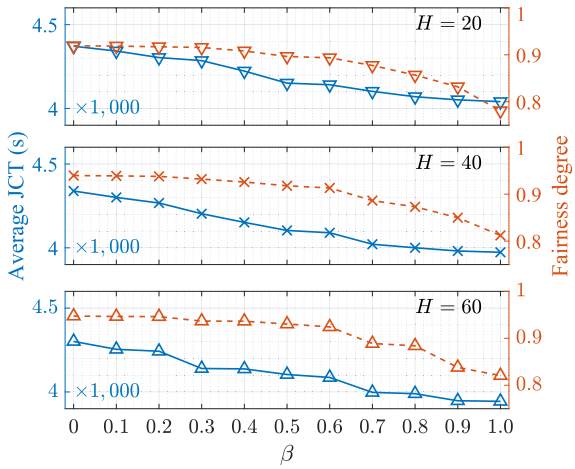


Fig. 13. Impact of β with $K=15$ and $N=40$.

exhaustive search, JPS ($N=60$) incurs performance losses of 0.54% and 2.04% in JCT for 15-worker and 30-worker scenarios, respectively. More importantly, compared with HAS, the ART is reduced by 4.86× and 49.15×. This indicates that compared with *exhaustive search* and HAS, JPS scales well with an increasing number of workers. Fig. 10 shows the fairness of various scheduling schemes. Under the default settings with β, the proposed HAS and JPS schemes only focus on low JCT, incurring the fairness degree ranging from 0.779 to 0.829.

Impact of S: Fig. 11 shows the average JCT of various schemes with S increasing from 3 to 5. Compared with *Optimus*-LB, JPS ($N=60$) incurs lower JCT ranging from 9.38% to 14.5%. Compared with *exhaustive search*, JPS ($N=60$) incurs performance loss ranging from 0.53% to 5.68%. As shown in Table IV, the ART of *exhaustive search* and HAS is increased by 82.17× and 143.18×, respectively. In contrast, the ART of JPS ($N=60$) is only increased by 3.16×, which demonstrates the efficiency of the proposed sampling scheme.

Impact of α on JPS: The value of α affects the performance of sampling. Let $K=15$ and $\alpha=[0,0.9]$, $(1-\alpha)\binom{K-1}{S-1}$ decreases from 364 to 36. As shown in Fig. 12, with the increase of α from 0 to 0.7, more categories in the front part of Ψ with higher JCT are not considered in sampling. The incurred JCT decreases. However, the lowest JCT is yielded in the 276th category. With the further increase of α, more and more categories with low JCT are also eliminated from sampling. The incurred JCT increases with the further increase of α from 0.7 to 0.9.

Impact of β on JPS: The weight β strikes a trade-off between JCT and fairness. For example, taking $N=60$ samples from Ψ, with the increase of β from 0 to 1.0, the average JCT decreases from 4,301.35 s to 3944.17 s. The fairness degree decreases from 0.947 to 0.821.

Impact of Heterogeneity Degree: Based on the throughput $\rho_{i,k}$, the heterogeneity degree among workers is defined as

$$H=\frac{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{S}}\rho_{i,k}}{K\cdot\min_{k\in\mathcal{K}}\sum_{i\in\mathcal{S}}\rho_{i,k}}. \tag{35}$$

The intuition of (35) is that if the workers are homogeneous with the same training throughput, the lowest heterogeneity degree $H=1$ can be achieved. Under the default setting, the heterogeneity degree is $H=1.15$, which becomes higher when more workers are equipped with the more powerful Tesla V100 as opposed to T4 in the experiments. Fig. 14 shows the average JCT of various schemes with the increase of heterogeneity degree from 1.1 to 1.26. With more powerful workers, the average

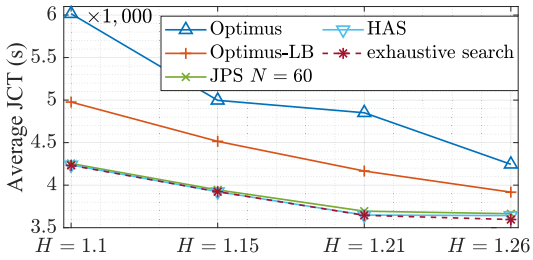


Fig. 14. Impact of heterogeneity degree H with $K = 15$.

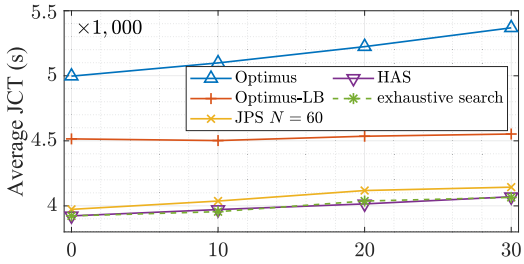


Fig. 15. Impact of prediction errors ε (%).

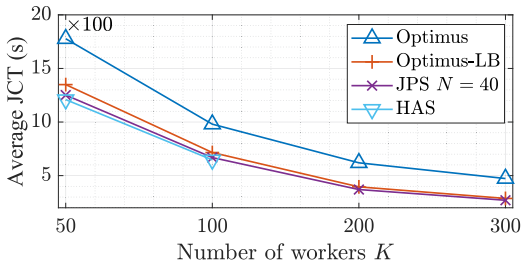


Fig. 16. Impact of the number of workers.

JCT decreases. Compared with *exhaustive search*, *Optimus* incurs a performance loss ranging from 42.09% to 18.07% In contrast, HAS achieves the optimal JCT when $H = 1.1$, 1.15, and 1.21, and incurs a performance loss of 1.28% when $H = 1.26$. Moreover, JPS ($H = 60$) incurs a performance loss ranging from 0.49% to 1.9%. Compared with *Optimus*, JPS ($H = 60$) reduces the average JCT ranging from 41.4% to 15.87% at the cost of acceptable computational overhead.

Impact of Prediction Errors: As we cannot ensure that the throughput estimator is always accurate, how the prediction error affects the performance is quantitatively analyzed. Let $\rho^*_{i,k}$ and ε denote the actual throughput and the prediction error upper bound, respectively. The predicted throughput is randomly generated in interval $[(1-\varepsilon)\cdot\rho^*_{i,k}, (1+\varepsilon)\cdot\rho^*_{i,k}]$. Fig. 15 shows the average JCT of various schemes over 100 rounds of execution. With the increase of ε from 0 to 30%, the JCT yielded by the proposed HAS and JPS schemes is only increased by 3.75% and 4.3%, respectively. The results demonstrate the robustness of our design against prediction errors.

Impact of K: Fig. 16 shows the average JCT of various schemes with the increasing number of workers. Please note that HAS is not applied to the scenarios of $K = 200$ and $K = 300$ due to the prohibitively high computational overheads. With the increase of K, the average JCT produced by all four schemes decreases as more computational resources help to reduce the computation latency. As shown in Fig. 16, JPS consistently achieves lower JCT compared to *Optimus* and *Optimus*-LB.

However, the performance gain of JPS over *Optimus*-LB diminishes with the continued increase of K. This is because the communication latency, which is determined by the bottleneck link, occupies an increasing proportion of the overall training latency. For example, the JPS scheme ($N = 40$) yields an average communication latency of 68.86 s and an average computation latency of 201.30 s when $K = 300$. In contrast, *Optimus*-LB yields an average communication latency of 69.13 s and an average computation latency of 217.96 s while with reduced ART as shown in Table V. The results show that the proposed JPS scheme works well when applied to a cluster comprising tens to hundreds of GPU workers. For larger-scale worker scenarios, it may be more advantageous to use *Optimus* extended with the proposed Proportional Workload Assignment scheme, i.e., *Optimus*-LB.

TABLE V
THE AVERAGE ALGORITHM RUNNING TIME (ART) OF VARIOUS SCHEMES

Scheme	*Optimus*	*Optimus-LB*	JPS $N = 40$	HAS
$K = 50$	4.0 ms	5.05 ms	2.11 s	857.81 s
$K = 100$	8.31 ms	10.03 ms	8.35 s	12.98 h
$K = 200$	22.56 ms	24.99 ms	35.92 s	N/A
$K = 300$	44.16 ms	48.68 ms	114.61 s	N/A

13

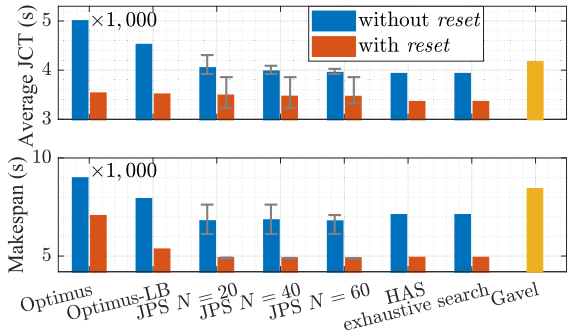


Fig. 17. Performance comparison of various schemes with/without *reset* events.

2) With Reset *Events:* When a job completion happens, the policy will be recomputed among the remaining active jobs. As shown in Fig. 17, compared with the static policy, the average JCT is reduced ranging from 17.01% to 41.67%. Policy recomputation with *reset* events is more beneficial for reducing the makespan of learning jobs, where makespan represents the maximum JCT of learning jobs. During the training process, the jobs requiring higher computational capacity are assigned more workers later. As shown in Fig. 17, the makespan is reduced ranging from 27.34% to 44.35%.

Then, we compare the performance of the dynamic policies considering *reset* events with *Gavel*. *Gavel* requires the number of requested GPUs scale_factor$_i$ as prior information. As *Gavel* supports distributed training over a single type of GPU in a round, the maximum value of scale_factor$_i$ is 5 to ensure the allocation does not oversubscribe GPU workers. Fig. 18 shows the average JCT of learning jobs for different predefined values

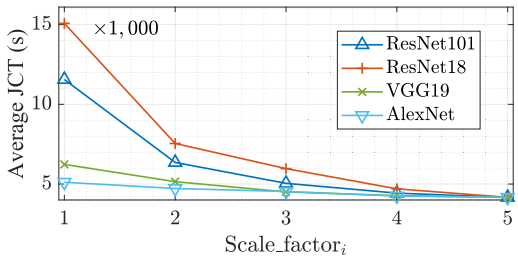


Fig. 18. Average JCT of using *Gavel* with the increased value of predefined scale_factor$_i$ for a learning job is shown. During this period, the scale factors of other learning jobs are set to 5.

of scale_factor$_i$. The average JCT decreases with the increased value of scale_factor$_i$ from 1 to 5, while keeping the scale factors of other learning jobs at 5. The lowest JCT of using *Gavel* can be achieved when the scale factors of four learning jobs are set to [5, 5, 5, 5].

However, limiting distributed training to a single type of GPU in a round cannot fully utilize the computational resources in heterogeneous computing clusters. Unlike *Gavel*, the proposed HAS and JPS schemes can schedule distributed training over multiple types of GPUs at a time. This means the number of allocated GPUs could be greater than 5 for computationally intensive learning jobs. As shown in Fig. 17, compared with *Gavel*, the average JCT and the makespan yielded by HAS with *reset* events are decreased by 21.2% and 71.3%, respectively.

VII. Conclusion and Future Work

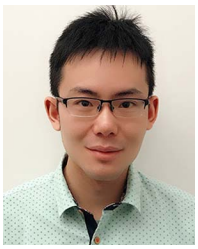
This article investigated the multi-job placement problem while taking into account job sizing and load balancing in heterogeneous deep learning clusters, aiming to reduce the average JCT of learning jobs. To be specific, we considered the widely used Ring-AllReduce architecture with data parallelism for distributed training. To accelerate the training process, multiple learning jobs may compete for computational resources. Considering the performance heterogeneity of workers, this paper proposed a novel scheme based on proportional training workload assignment, feasible solution categorization, and matching markets with theoretical guarantees. After job prioritizing, we identified the declining trend of incurred JCT among feasible solution categories. A sampling-based scheme was proposed to reduce the computational complexity while ensuring scheduling fairness. We deploy a testbed of heterogeneous deep learning cluster hosted on Compute Canada for performance evaluation. Evaluation results demonstrated that, when compared with the globally optimal policy, the sampling-based scheme only incurs a performance loss of about 2% with much higher solution efficiency.

In future work, we plan to 1) focus on the auto-tuning of α and β to improve the performance of sampling, and 2) extend the considered scenario from data parallelism to model/hybrid parallelism.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[2] J. Wang, K. Liu, G. Tzanetakis, and J. Pan, "Learning-based cooperative sound event detection with edge computing," in *Proc. IEEE 37th Int. Perform. Comput. Commun. Conf.*, 2018, pp. 1–8.
[3] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv: 1810.04805*.
[4] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *Proc. 14th USENIX Conf. Operating Syst. Des. Implementation*, 2020, pp. 481–498.
[5] C. Chen, Q. Weng, W. Wang, B. Li, and B. Li, "Accelerating distributed learning in non-dedicated environments," *IEEE Trans. Cloud Comput.*, vol. 11, no. 1, pp. 515–531, First Quarter 2023.
[6] J. Gu et al., "Tiresias: A GPU cluster manager for distributed deep learning," in *Proc. 16th USENIX Conf. Netw. Syst. Des. Implementation*, 2019, pp. 485–500.
[7] A. Sultana, L. Chen, F. Xu, and X. Yuan, "E-LAS: Design and analysis of completion-time agnostic scheduling for distributed deep learning cluster," in *Proc. 49th Int. Conf. Parallel Process.*, 2020, pp. 1–11.
[8] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–14.
[9] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 505–513.
[10] K. Mahajan et al., "THEMIS: Fair and efficient GPU cluster scheduling," in *Proc. 17th USENIX Conf. Netw. Syst. Des. Implementation*, 2020, pp. 289–304.
[11] D. Narayanan et al., "Solving large-scale granular resource allocation problems efficiently with POP," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, 2021, pp. 521–537.
[12] K. Nagrecha and A. Kumar, "Saturn: An optimized data system for multi-large-model deep learning workloads," *Proc. VLDB Endowment*, vol. 17, no. 4, pp. 712–725, 2023.
[13] W. Gao et al., "Deep learning workload scheduling in GPU datacenters: Taxonomy, challenges and vision," 2022, *arXiv:2205.11913*.
[14] P. Samal and P. Mishra, "Analysis of variants in round robin algorithms for load balancing in cloud computing," *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, no. 3, pp. 416–419, 2013.
[15] A. N. Tantawi and D. Towsley, "Optimal static load balancing in distributed computer systems," *J. ACM*, vol. 32, no. 2, pp. 445–465, 1985.
[16] A. Harlap et al., "Addressing the straggler problem for iterative convergent parallel ML," in *Proc. 7th ACM Symp. Cloud Comput.*, 2016, pp. 98–111.
[17] Q. Su, M. Wang, D. Zheng, and Z. Zhang, "Adaptive load balancing for parallel GNN training," in *Proc. MLSys Workshop GNNSys*, 2021, pp. 1–8.
[18] W. Wang et al., "TopoOpt: Co-optimizing network topology and parallelization strategy for distributed training jobs," in *Proc. 20th USENIX Conf. Netw. Syst. Des. Implementation*, 2023, pp. 739–767.
[19] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," in *Proc. Mach. Learn. Syst.*, 2019, pp. 1–13.
[20] L. Suresh et al., "Building scalable and flexible cluster managers using declarative programming," in *Proc. 14th USENIX Conf. Operating Syst. Des. Implementation*, 2020, pp. 827–844.
[21] D. Easley and J. Kleinberg, *Networks, Crowds, and Markets*. Cambridge, U.K.: Cambridge Univ. Press, 2010.
[22] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Operating Syst. Des. Implementation*, 2016, pp. 265–283.
[23] Y. Jia et al., "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.
[24] T. Chen et al., "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015, *arXiv:1512.01274*.
[25] V. K. Vavilapalli et al., "Apache hadoop YARN: Yet another resource negotiator," in *Proc. 4th ACM Symp. Cloud Comput.*, 2013, pp. 1–16.
[26] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, "Communication-efficient distributed deep learning: A comprehensive survey," 2020. [Online]. Available: https://arxiv.org/abs/2003.06307
[27] NCCL, 2024. [Online]. Available: https://developer.nvidia.com/nccl
[28] Gloo, 2024. [Online]. Available: https://github.com/facebookincubator/gloo

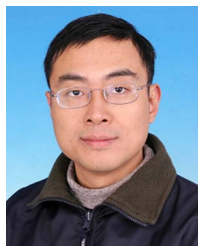
[29] Baidu-AllReduce, 2017. [Online]. Available: https://github.com/baidu-research/baidu-allreduce
[30] K. Hsieh et al., "Gaia: Geo-distributed machine learning approaching LAN speeds," in *Proc. 14th USENIX Conf. Netw. Syst. Des. Implementation*, 2017, pp. 629–647.
[31] Q. Ho et al., "More effective distributed ML via a stale synchronous parallel parameter server," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 1223–1231.
[32] B. Recht, C. Ré, S. J. Wright, and F. Niu, "HOGWILD: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. 24th Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 693–701.
[33] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 127–144, 2014.
[34] J. S. Hunter, "The exponentially weighted moving average," *J. Qual. Technol.*, vol. 18, no. 4, pp. 203–210, 1986.
[35] N. Katoh and T. Ibaraki, "Resource allocation problems," in *Handbook of Combinatorial Optimization*, Berlin, Germany: Springer, 2013, pp. 905–1006.
[36] P. Flajolet and R. Sedgewick, *Analytic Combinatorics*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[37] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," DEC Research Report TR-301, pp. 1–38, Sep. 1984.
[38] K. Liu, J. Peng, J. Wang, Z. Huang, and J. Pan, "Adaptive and scalable caching with erasure codes in distributed cloud-edge storage systems," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1840–1853, Second Quarter, 2023.
[39] Compute Canada, 2024. [Online]. Available: https://alliancecan.ca/en/services/advanced-research-computing/acknowledging-alliance
[40] Slurm, 2024. [Online]. Available: https://slurm.schedmd.com/documentation.html
[41] NVLink and NVSwitch, 2024. [Online]. Available: https://www.nvidia.com/en-us/data-center/nvlink/
[42] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters," in *Proc. 14th USENIX Conf. Operating Syst. Des. Implementation*, 2020, pp. 463–479.
[43] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, pp. 63–74.
[44] PyTorch, 2024. [Online]. Available: https://pytorch.org/
[45] CIFAR10, 2009. [Online]. Available: http://www.cs.toronto.edu/kriz/cifar.html
[46] Tiny ImageNet, 2017. [Online]. Available: https://www.kaggle.com/c/tiny-imagenet
[47] WikiText-2, 2016. [Online]. Available: https://pytorch.org/text/stable/datasets.html#torchtext.datasets.WikiText2



Kaiyang Liu (Member, IEEE) received the PhD degree from the School of Information Science and Engineering, Central South University, in 2019. He is currently an assistant professor with the Department of Computer Science, Memorial University of Newfoundland (MUN), St. John's, Newfoundland and Labrador, Canada. Before joining MUN, he worked as a postdoctoral fellow with the University of Victoria, Victoria, British Columbia, Canada. His current research areas include distributed cloud/edge computing and storage networks, data center networks, and distributed machine learning. One of his papers is one of three IEEE LCN 2018 Best Paper Award candidates. In recognition of his contributions to cost-effective services in cloud data centers, he received the IEEE Technical Committee on Cloud Computing (TCCLD) Outstanding PhD Thesis Award, in 2020.

Jingrong Wang (Student Member, IEEE) received the bachelor's degree from the School of Electronic and Information Engineering, Beijing Jiaotong University, in 2017, and the MSc degree in computer science from the University of Victoria, in 2019. She is currently working toward the PhD degree with the University of Toronto. Her research interests cover wireless communications, mobile edge computing, and distributed machine learning.

Zhiming Huang (Student Member, IEEE) received the bachelor's degree in communication engineering from Northwestern Polytechnical University, Xi'an, China, in 2018, and the master's degree in computer science from the University of Victoria, BC, Canada, in 2020, where he is currently working toward the PhD degree in computer science. His research interest includes design and analysis of online learning in communication networks.



Jianping Pan (Fellow, IEEE) received the bachelor's and PhD degrees in computer science from Southeast University, Nanjing, Jiangsu, China, and he did his postdoctoral research with the University of Waterloo, Waterloo, Ontario, Canada. He is currently a professor of computer science with the University of Victoria, Victoria, British Columbia, Canada. He also worked with Fujitsu Labs and NTT Labs. His area of specialization is computer networks and distributed systems, and his current research interests include protocols for advanced networking, performance analysis of networked systems, and applied network security. He received the IEICE Best Paper Award, in 2009, the Telecommunications Advancement Foundation's Telesys Award, in 2010, the WCSP 2011 Best Paper Award, the IEEE Globecom 2011 Best Paper Award, the JSPS Invitation Fellowship, in 2012, the IEEE ICC 2013 Best Paper Award, and the NSERC DAS Award, in 2016, is a coauthor of one of three IEEE LCN 2018 Best Paper Award candidates, and has been serving on the technical program committees of major computer communications and networking conferences including IEEE INFOCOM, ICC, Globecom, WCNC and CCNC. He was the Ad Hoc and Sensor Networking Symposium co-chair of IEEE Globecom 2012 and an associate editor of *IEEE Transactions on Vehicular Technology*. He is a senior member of the ACM.