Non-Clairvoyant Scheduling of Distributed Machine Learning With Inter-Job and Intra-Job Parallelism on Heterogeneous GPUs

Fahao Chen, Peng Li, *Senior Member, IEEE*, Celimuge Wu, *Senior Member, IEEE*, and Song Guo, *Fellow, IEEE*

***Abstract*—Distributed machine learning (DML) has shown great promise in accelerating model training on multiple GPUs. To increase GPU utilization, a common practice is to let multiple learning jobs share GPU clusters, where the most fundamental and critical challenge is how to efficiently schedule these jobs on GPUs. However, existing works about DML job scheduling are constrained to settings with homogeneous GPUs. GPU heterogeneity is common in practice, but its influence on multiple DML job scheduling has been seldom studied. Moreover, DML jobs have internal structures that contain great parallelism potentials, which have not yet been fully exploited in the heterogeneous computing environment. In this paper, we propose *Hare*, a DML job scheduler that exploits both inter-job and intra-job parallelism in a heterogeneous GPU cluster. *Hare* adopts a relaxed fixed-scale synchronization scheme that allows independent tasks to be flexibly scheduled within a training round. Given full knowledge of job arrival time and sizes, we propose a fast heuristic algorithm to minimize the average job completion time and derive its theoretical bound is derived. Without prior knowledge of jobs, we propose an online algorithm based on the Heterogeneity-aware Least-Attained Service (HLAS) policy. We evaluate *Hare* using a small-scale testbed and a trace-driven simulator. The results show that it can outperform the state-of-the-art, achieving a performance improvement of about 2.94×.**

***Index Terms*—Distributed machine learning, heterogeneous GPUs, intra-job parallelism, online scheduling.**

Manuscript received 5 January 2023; revised 16 April 2024; accepted 2 June 2024. Date of publication 14 June 2024; date of current version 6 December 2024. This work was supported in part by Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant 24K02932, in part by Japan Science and Technology Agency (JST) PRESTO under Grant 23828673, in part by ROIS NII Open Collaborative Research under Grant 24S0601, in part by Key-Area Research and Development Program of Guangdong Province under Grant 2021B0101400003, in part by Hong Kong RGC Research Impact Fund under Grant R5060-19, in part by General Research Fund under Grant 152221/19E, Grant 152203/20E, and Grant 152244/21E, in part by the National Natural Science Foundation of China under Grant 61872310, and in part by Shenzhen Science and Technology Innovation Commission under Grant JCYJ20200109142008673. Recommended for acceptance by M. Kandemir. *(Corresponding author: Peng Li.)*

Fahao Chen and Peng Li are with the School of Computer Science and Engineering, University of Aizu, Aizuwakamatsu 965-0006, Japan (e-mail: d8232101@u-aizu.ac.jp; pengli@u-aizu.ac.jp).

Celimuge Wu is with the Meta-Networking Research Center, University of Electro-Communications, Chofu 182-8585, Japan (e-mail: celimuge@uec.ac.jp).

Song Guo is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, SAR, China (e-mail: songguo@cse.ust.hk).

Digital Object Identifier 10.1109/TCC.2024.3414440

I. Introduction

Distributed machine learning (DML) has been widely studied because it is a straightforward but effective way to accelerate the training of complex models using Big Data. In the paradigm of distributed machine learning (DML), a learning job is divided into multiple tasks, which can run on multiple GPUs in parallel. The Parameter Server (PS) [1] scheme has been widely adopted to coordinate the training processes across multiple GPUs.

A critical research challenge of DML is how to efficiently schedule these jobs on GPUs, which is particularly concerned by public or private cloud data centers that offer learning services while desiring high hardware resource utilization. Therefore, the learning job scheduling problem has attracted great research attention, and various solutions [2], [3], [4] have been recently proposed with different objectives. For instance, Gandiva [2] has studied GPU sharing among several jobs to improve GPU utilization. Pollux [3] considers the fairness of learning jobs, and recent work [4] has exploited both intra-job and inter-job parallelism and proposed an efficient DML job scheduling algorithm to minimize the total job completion time.

Most of the existing works, however, are based on an assumption that GPUs are homogeneous. In practice, hardware heterogeneity commonly exists in computing clusters. For example, as the expansion of data centers, new GPUs are continuously added and they should work with existing ones to maximize resource utilization. The heterogeneity refers to the variety and differences in the hardware configurations of computing clusters. Specifically, the heterogeneity of GPUs indicates differences in several features, such as architecture, computation capacities (i.e., CUDA cores), memory size, bandwidth, and others. Some recent works [5], [6], [7] have started to pay attention to the influence of GPU-heterogeneity, which motivates us to re-examine the DML job scheduling problem in such an emerging heterogeneous computing environment.

Hardware heterogeneity brings new challenges as well as opportunities to DML system design. We are excited to see the success of several preliminary studies. For example, Gandiva$_{\text{fair}}$ [5] is designed to ensure the user-level fairness while maximizing the efficiency of heterogeneous GPU clusters. Gavel [7] generalizes existing scheduling policies with consideration of GPU heterogeneity. Allox [6] efficiently schedules ML jobs in a heterogeneous cluster to improve the max-min fairness.

These recent works have extensively studied inter-job parallelism in heterogeneous computing environment, but leaving intra-job parallelism unexplored. They treat each DML job as a unsplittable unit when making scheduling decisions. We are still facing open questions: how to exploit both inter-job and intra-job parallelism on heterogeneous GPUs? How much acceleration can be obtained? And is there strong theoretical support for such acceleration?

To fill this gap, we have proposed *Hare* [8], a sophisticated DML job scheduler that exploits the parallelism at both intra-job and inter-job levels while considering GPU heterogeneity. *Hare* has three key techniques. First, it uses a relaxed scale-fixed synchronization scheme to maximize scheduling flexibility. It fixes the number of tasks in each round but relaxes resource requirement for scheduling, so that we can maintain convergence certainty while maximizing GPU utilization. Second, *Hare* enables fast task switching by optimizing task initialization and cleaning on GPUs, which has been identified as the major source of switching overhead. The final one is a fast heuristic algorithm, with theoretical performance guarantee, to minimize the average job completion time for a set of DML jobs whose arrival time and sizes are given.

Although *Hare* has made a great success, it needs the full information of job sizes and arrival time to make good scheduling decisions. However, this information is not always available because DML jobs are submitted to the cluster at different time, and it is hardly to predict when and what jobs will be submitted in the future. In addition, many jobs use AutoML [9], [10], [11] or early stopping [2], [12] techniques, and thus their job sizes are unknown even though they have already arrived and started to run. This phenomenon has been also reported by [13], [14]. Although online job scheduling has been studied by many existing works [4], [14], they cannot be directly applied here because of new challenges brought by *Hare*'s unique scheduling manner. The relaxed scale-fixed synchronization combined with heterogeneous GPUs could make almost all existing online scheduling algorithms show prohibitively poor performance.

In this paper, we propose a non-clairvoyant scheduling policy to enhance *Hare*, so that it can exploit inter-job and intra-job parallelism of online DML jobs. "Non-clairvoyant scheduling" refers to the algorithm's capability to make scheduling decisions without complete knowledge of job sizes and arrival time. Most of the existing schedulers are based on full information of DML jobs, which is ideal and typically impractical. For instance, Themis [15] requires jobs' finish time to model the finish-time fairness metric for scheduling. Our "non-clairvoyant scheduling" is more applicable to real-time scheduling where future job details are unknown. Our proposal is based on the Least Attained Service (LAS) policy, which has been shown to be effective in handling online jobs [16], [17]. Since existing LAS policy is unaware of GPU heterogeneity, we propose a customized version called Heterogeneity-aware Least-Attained Service (HLAS), which redefines the "service" in *Hare* and uses a GPU grouping scheme to wipe off the influence of hardware heterogeneity.

We further relax the obliviousness assumption about job sizes and examine the potential benefits of using prediction in HLAS, which is motivated by some recent works on predicting DML job sizes. However, existing works are mainly based on fitting training loss curves, so they need to collect sufficient job states before making accurate prediction. If we apply them in HLAS, it is highly possible that we get wrong job sizes in the first few rounds, which would seriously degrade the scheduling performance. For example, if a small job is predicted as a long one, it would misguide the scheduling algorithm to postpone its execution. Therefore, we propose a conservative prediction method that combines loss curve fitting techniques and historical running records. Instead of aiming at predicting full job sizes, we make a partial prediction about a limited number of additional rounds. This conservative method can significantly increase prediction accuracy, to minimize the possibility of making wrong scheduling decisions.

We further propose an extended version of HLAS, called HLAS-P, to use predicted job information for performance improvement. We define a new concept of virtual job size consisting of two parts: the number of already scheduled rounds and that of predicted rounds. We follow the HLAS to assign running priorities according to virtual job sizes. However, for jobs with the same priority, we make fine-grained scheduling decisions by comparing their obtained "service" and predicted job sizes.

Compared to the previous version of *Hare* in [8], this paper makes three new contributions, which are summarized as follows.

- We propose a non-clairvoyant scheduling policy called HLAS for *Hare*, which can efficiently schedule DML jobs without prior knowledge about job sizes and arrival time. The HLAS method can outperform existing works with a performance improvement of about 2.94×.
- We study to use predicted job sizes to improve HLAS. Due to the unique scheduling features of HLAS, we propose a conservative prediction method to make sure we can always have accurate predicted result, instead of pursuing full job size prediction.
- We propose HLAS-P as an enhancement of HLAS to use predicted job sizes for further performance improvement.

The rest of this paper is organized as follows. We introduce the background and motivation in Section II, followed by the system overview in Section III. The non-clairvoyant scheduling policy is presented in Section IV. We evaluate *Hare* in Section VI. The related work is presented in Section VII. Finally, we conclude this paper in Section VIII.

II. Background and Motivation

In this section, we first give the background of distributed machine learning (DML), and then present the unique characters of DML jobs, which motivate us to design *Hare*.

A. Background

Distributed machine learning (DML) on GPUs has been widely adopted to accelerate model training on large datasets. The training goal is to minimize a loss function as follows:

$$\mathcal{L}(w) = \frac{1}{|\mathcal{P}|} \sum_{p_i \in \mathcal{P}} \ell(\mathbf{w}, p_i), \qquad (1)$$

where p_i is a data point in the training dataset $\mathcal{P}$. The loss function $\ell(\cdot)$ is, typically cross-entropy for the classification problem

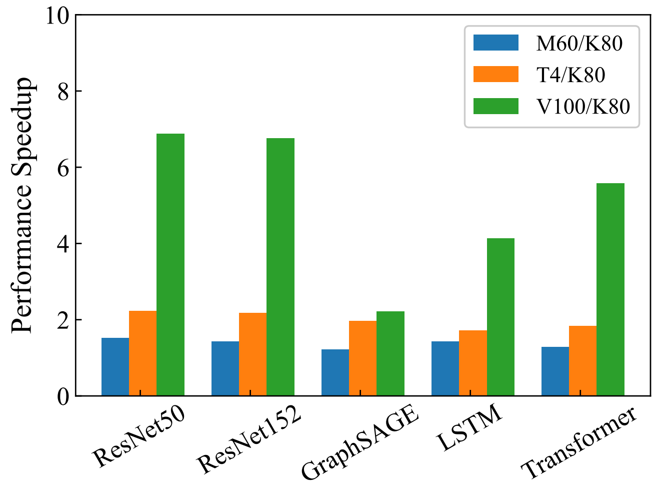

Fig. 1. Training speedup of different jobs on different GPUs.

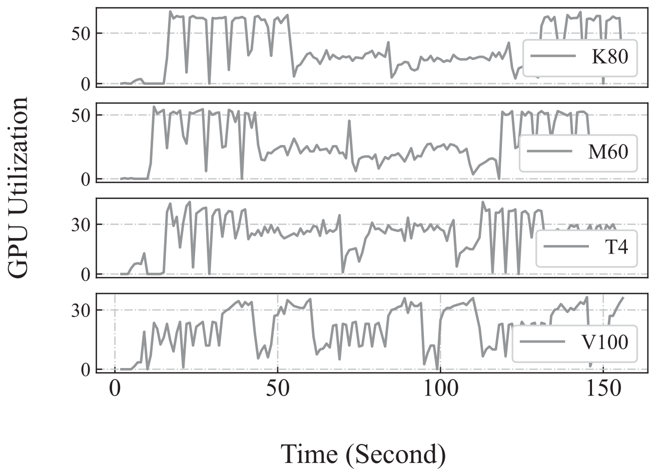

Fig. 2. The GPU utilization of training GraphSAGE model.

or squared error for the regression problem. The trainable model parameters $\mathbf{w}$ are updated iteratively by using stochastic gradient descent (SGD).

In each iteration, training workloads are shared by $\mathcal{K}$ GPUs, which are also called workers. Each worker is assigned a fixed-size mini-batch $\mathcal{B}_k \subseteq \mathcal{P}$ and computes its local gradients g_k^t as:

$$g_k^t = \frac{1}{|\mathcal{B}_k|} \sum_{p_i \in \mathcal{B}_k} \nabla \ell(\mathbf{w}^t, p_i). \tag{2}$$

After the local training, workers send their gradient updates to a parameter server that creates a global model:

$$g^t = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} g_k^t;\ \mathbf{w}^{t+1} = \mathbf{w}^t - \eta g^t, \tag{3}$$

where η is the learning rate. Then workers download the global model and move to the next iteration of training. The training process ends when a required number of rounds is achieved. Typically, the parallelism scale $|\mathcal{K}|$, the batch size $|\mathcal{B}|$, and the learning rate η are chosen by the user.

Typically, multiple DML jobs share GPU resources in a cluster. With the soaring size of the DML jobs, a sophisticated scheduler is needed to shorten the training time and improve GPU utilization. Existing works [2], [4], [5], [7], [15], [18] have made many efforts on scheduling algorithms design with an assumption that GPUs in the cluster are homogeneous. However, existing clusters usually accommodate different types of GPUs with various specifications, which implies the inefficiency of existing homogeneity-based schedulers.

B. Motivation

1) GPU Heterogeneity and Inter-Job Parallelism: We find that different GPUs provide different performance speedups for learning jobs, mainly because of the heterogeneity of model (such as model architecture) and hardware. As shown in Fig. 1, we use the training time per mini-batch on a K80 GPU as the baseline and evaluate the speedup for other GPUs. For example, "M60/K80" means the performance speedup on the M60 GPU, compared to the training speed on a K80 GPU. Training the ResNet50 model can be sped up by 2x on a T4 GPU, while with 7x significant speedup on a V100 GPU. However, the graph learning model GraphSAGE shows the heterogeneous performance on different GPUs. Specifically, GraphSAGE can only be sped up by about 2x, even on the most advanced V100 GPU. That is because the required FLOPS of GraphSAGE are much smaller than other models. Moreover, the data pre-processing speed is slower than the GPU computation speed. The GPU spends more time to wait for input data, resulting in low GPU utilization. As shown in Fig. 2, we find that utilization of GPU is less than 30% when we train GraphSAGE on a V100 GPU. There is a little improvement when training GraphSAGE on a V100 GPU. Therefore, giving a high priority for assigning V100 GPUs to the ResNet50 job is more efficient since it shows a high-performance speedup than other jobs.

This empirical study gives us important hints about accelerating learning jobs and increasing GPU utilization. On the other hand, it throws challenges about how to schedule jobs on GPUs, considering massive learning workloads and hardware resources in modern data centers. Moreover, the intra-job parallelism, which will presented in the following, further complicates this problem.

2) GPU Heterogeneity and Intra-Job Parallelism: Each DML job consists of multiple tasks, which are periodically synchronized to share gradients, via a parameter server or exchanging gradients directly. Although a single advanced GPU can provide a performance speedup for gradients computing, the training speed of the whole DML jobs is constrained by the synchronization. We train the ResNet152 on five different distributed settings and show the epoch time in Fig. 3. We find that mixing different GPUs is not always helpful. For example, compared to a pure K80 cluster, adding faster T4 or V100 brings no acceleration. That is because the gradient synchronization impedes early completed GPUs to move to the next-round training. There is much idle time on V100 GPUs when they waits for the gradients update from K80 GPUs. This low efficiency can be also reflected by GPU utilization as shown in Fig. 4, where we can see that K80 is always busy while V100's utilization is rarely over 50%.

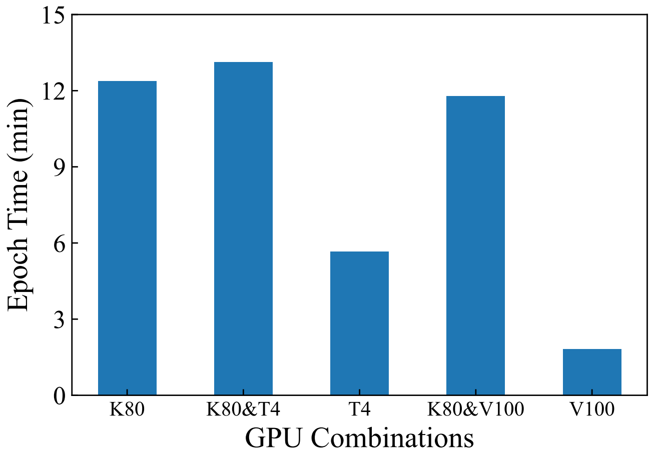

Fig. 3. Epoch time of ResNet152 under different GPU combinations.

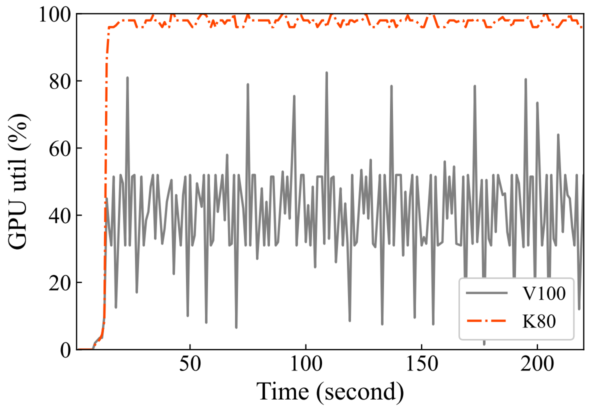

Fig. 4. GPU utilization of V100 and K80 when training ResNet152.

A straightforward idea to address this challenge is to schedule parallel tasks belonging to the same job on similar GPUs. However, it is hardly to have such a perfect allocation in practice because of limited GPU resources in the cluster. Since it is inevitable to use heterogeneous GPUs for intra-job parallelism, it is desired an algorithm that can well schedule fine-grained tasks to reduce idle time.

3) Scale-Fixed Synchronization Versus Scale-Adaptive Synchronization: Existing intra-job parallelism methods can be categorized into two types, scale-fixed and scale-adaptive, according to how many tasks are synchronized. Scale-fixed methods, adopted by Tiresias [18] and Gandiva [2], fix the number of synchronized tasks and always try to allocate the same number of GPUs so that they can achieve full parallelism. If the number of available GPUs is insufficient, all tasks need to wait until required GPU number is satisfied. In contrast, scale-adaptive methods [7], [15], [19], [20] dynamically change the number of synchronized tasks according to available GPU resources. Although these methods are flexible and tasks are not blocked by strict resource requirement, we may need more training epochs to achieve competitive accuracy of scale-fixed methods. Moreover, it is hard to build theories to predict how many epochs are needed. Due to this uncertainty, we do not use scale-adaptive design in *Hare*.

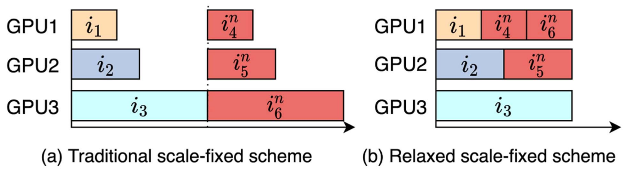
GPU1
GPU2
GPU3
i_1
i_2
i_3
i_4^n
i_5^n
i_6^n
(a) Traditional scale-fixed scheme
(b) Relaxed scale-fixed scheme

Fig. 5. An example showing the benefit of relaxed scale-fixed synchronization scheme adopted by *Hare*.

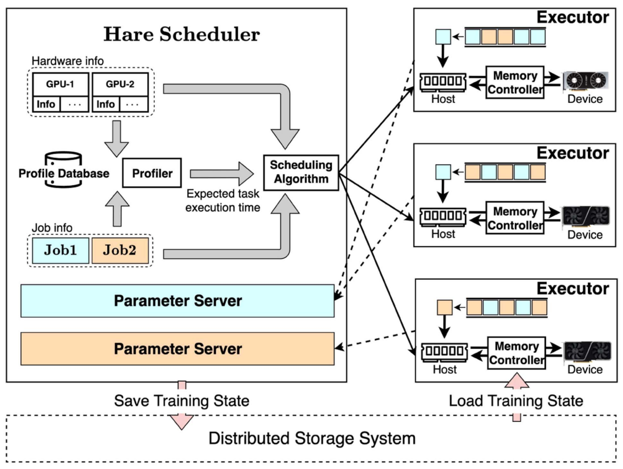

Fig. 6. System overview.

Motivated by the above analysis, we would like to follow the scale-fixed idea but relax the parallelism requirement. An example is shown in Fig. 5, where three tasks, i_1, i_2 and i_3, are running on 3 GPUs respectively. Now a new job n consisting of 3 tasks (i.e., synchronization scale is 3) comes. As illustrated in Fig. 5(a), traditional scale-fixed methods start job n after the completion of slowest task i_3, when 3 GPUs are available. We find that it is unnecessary to make 3 tasks strictly run in parallel. Two tasks can run sequentially on GPU1, as shown in Fig. 5(b), leading to earlier completion than traditional methods while maintaining the same level of parallelism.

Implementing such a relaxed scale-fixed synchronization method is not easy. We need to address challenge of changing the task assignment and synchronization modules. It also affects task scheduling algorithm design.

III. System Overview of *Hare*

A system overview of *Hare* is shown in Fig. 6, where *Hare* is integrated into the existing PS-based distributed machine learning framework. *Hare* is not only a scheduling algorithm, but also a set of modules that optimize training processes across GPUs. It contains two main components: a logically centralized task scheduler, and executors running on training machines. All data are stored with HDFS [21]. It also collects hardware information, e.g., GPU types, speed and memory, from the under-layer computing infrastructure. These information first goes to a module called profiler that trains a small piece of data to obtain expected task execution time on different GPUs, which will be the input of the task scheduling algorithm. We note that some jobs are usually repeatedly submitted to the training platform. For example, some models are periodically re-trained using latest collected datasets to adapt to emerging cases, which is particularly common in deep reinforcement learning. This observation motivates us to accelerate the profiling by maintaining a database that stores historical profiling results. We first search the database upon receiving job information. If corresponding results can be found, we skip profile training and directly feed searching results to the scheduling algorithm. We then run the scheduling algorithm to generate a task running sequence for each GPU. Finally, these task sequences are sent to corresponding executors.

Each executor schedules tasks and loads checkpoints on GPU according to their order given in the received task sequence. When a task completes, it sends updated gradients to the corresponding parameter server for aggregation. We follow the most of training designs in traditional distributed machine learning frameworks [15], [19], except the task switching mechanism. In existing works, since each job has exclusive use of assigned GPUs, several consecutive tasks on a GPU belongs to the same job and they share the same GPU context, leading to low switching overhead. In contrast, *Hare* allows GPU preemption by alternatively running tasks of different jobs, which involves frequent context switching with high overhead. To reduce the task switching overhead, *Hare* introduces the fast task switching mechanism [8], which deletes intermediate data of each layer once its backward training completes. Thanks to the early task cleaning, the released GPU memory can be used for pre-loading data of the next task, so that it can start earlier.

Given a set of jobs with full information about sizes and arrival time, we formulate a scheduling problem with the objective of minimizing average job completion time. This problem is NP-hard, which can be proved by reducing the well-known SS13 problem [22]. A heuristic algorithm with theoretical performance guarantee has been designed. The details of scheduling with prior knowledge can be found in [8].

IV. Non-Clairvoyant Scheduling Algorithm

Based on the relaxed scale-fixed synchronization and fast task switching of *Hare* [8], we propose an online scheduling algorithm in Section IV-C, called Heterogeneity-aware Least-Attained Service (HLAS), to handle jobs without arrival time and size information.

A. Problem Statement

We consider a heterogeneous GPU cluster similar with the setting in [8], including basic GPU and job information, e.g., heterogeneous training time on different GPUs. Different from [8], we do not require the size and arrival time of jobs must be available. The set of GPUs is denoted by $\mathcal{M}$. Some training jobs, denoted by set $\mathcal{N}$, are submitted to this cluster at different time. The set $\mathcal{N}$ is an ordered set according to the job's arrival time. The arrival time of each job $n \in \mathcal{N}$ is denoted by a_n, which is unknown before job arrival. Each job $n \in \mathcal{N}$ launches a set D_r of training tasks that can run in parallel in every training round, and each task is responsible for training a data batch. After local training, all tasks synchronize their gradients via the PS scheme to obtain an updated model for the next-round training.

Due to GPU heterogeneity, each task may have different training time on different GPUs. Similarly, it may have different

TABLE I
Notations

$\mathcal{N}$	set of training jobs	$\mathcal{M}$	set of heterogeneous GPUs
a_n	arrive time of job n	w_n	weight of job n
R_n	set of training rounds for job $n \in \mathcal{N}$		
D_r	set of parallel tasks in $r \in R_n$		
$T^c_{n,m}$	the single-batch training time of job n on GPU m		
$T^s_{n,m}$	the single-batch synchronization time of job n on GPU m		
$s_{n,m}$	training speed of job n on GPU m		
$\bar{s}_n$	average training speed of job n		
$x^t_{n,m}$	binary variable to indicate whether job n is processed on GPU m at time slot t		
$z_{m,u}$	binary variable to indicate whether GPU m is grouped to group u		
p^{max}_n	job n's maximum speed		
p^{min}_n	job n's minimum speed		
T_n	average one-round time costs for job n		
$R^r_n(t)$	number of rounds that already run for job n at timestep t		
Q	job execution queue		
E^L_k	the lowest job sizes that an be accommodated in Q_k		
E^H_k	the highest job sizes that an be accommodated in Q_k		

synchronization time across GPUs because network condition changes. Besides, we assume that the training time is longer than the synchronization time. That is because GPUs are usually connected by high-speed networks (e.g., NVLink and InfiniBand) in data centers. Note that this is different from coarse-grained job-level non-preemption assumed in existing works [2], [18], [19], i.e., the whole DML job can not be preempted by other DML jobs. We consider a fine-grained non-preemption setting for task running, i.e., a task's execution cannot be preempted once it is scheduled on a GPU, where a task is responsible for training a data batch and each job involves a set of training tasks. Thanks to the fast task switching mechanism, there is tiny task switching cost, which is less than 5% of task training time according to experimental results. Therefore, we ignore the task switching cost in the problem formulation for simplicity.

Different from [8], we make no assumption about the job arrival time a_n and the number of training rounds needed by each job, i.e., R_n is unknown. After each training round, we know whether this job continues to run or not, which is similar with the setting in [18]. In addition, we can collect some basic information of jobs during processing, including start time and time already be used for training. The job completion time of a job is defined as the difference between its finished time and arrival time. The goal of job scheduling algorithm is to minimize the average job completion time (JCT). A list of notations is shown in Table I.

B. Basic Idea

Given job sizes, it is well known that the Shortest-Remaining-Time-First (SRTF) heuristic works well to minimize the average JCT, even without job arrival information. However, SRTF cannot be applied here because neither arrival time nor job sizes are available. We cannot judge which job is the "shortest". Therefore, we turn to the Least-Attained Service (LAS) policy [23], which approximates SRTF by always assigning higher scheduling priorities to jobs that have received least services. LAS and its variants have been widely applied for online scheduling problems without job sizes [17], [24]. However, they cannot

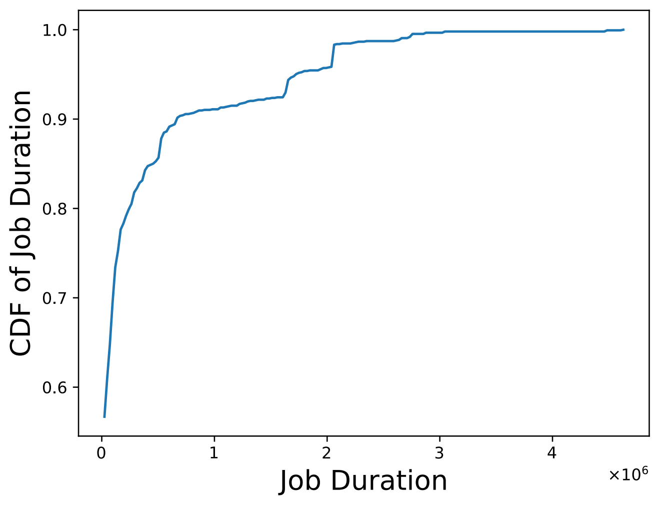

Fig. 7. CDF of job duration in trace.

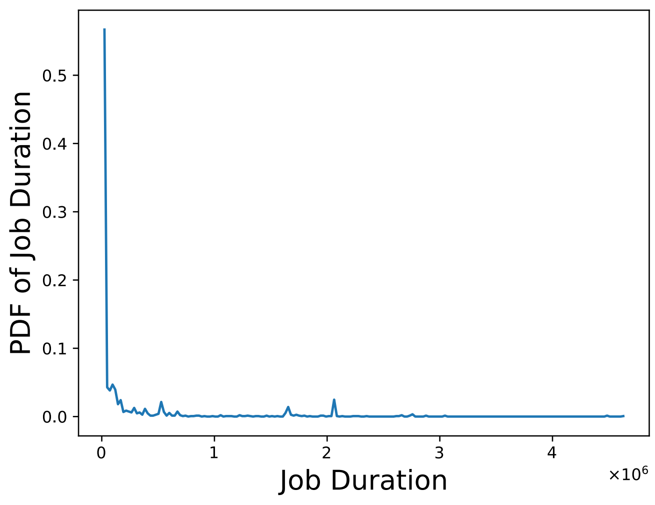

Fig. 8. PDF of job duration in trace.

	GPU1	GPU2	GPU3	GPU4
J_A	3	3	1.8	1.8
J_B	1	2	2.5	1

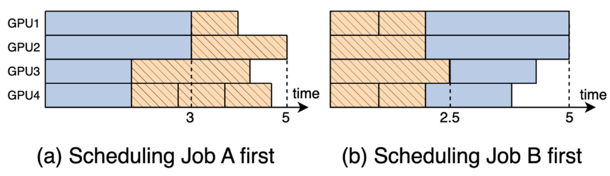

Fig. 9. A toy example to show job scheduling results under different evaluation of attained service.

be directly used here because of unique challenges brought by online DML scheduling.

The first challenge is about job size distribution. It has been proved that LAS can approximate the SRTF heuristic when job sizes obey a heavy-tailed distribution [17]. Therefore, it is important to verify job distribution before applying LAS, so that we can make sure LAS can fully play its power in the target online scheduling problem. For example, in [17], [25], in order to apply LAS for network flow scheduling in datacenters, authors has shown that the sizes of network flows in datacenter exhibit a heavy-tailed distribution. Similarly, we collect DML job traces from Google cluster [26] and plot their CDF and PDF curves in Figs. 7 and 8. We can see that many jobs are small and a few large jobs occupy the most of GPU times, which fits the features of heavy-tailed distribution.

The second challenge is about scheduling granularity. Many existing works [18], [27] using LAS allow arbitrary preemption of job execution and design fine-grained scheduling policy, to pursue better approximation to SRTF. However, they cannot be directly applied here because distinct characteristics of DML jobs and some system constraints imposed by *Hare*. First, DML jobs arrive in terms of rounds, consisting of a set of training tasks. Only when all tasks of a round complete, we can finish this round and update training models. Second, *Hare* does not allow preemption of a task during its execution, to control the task switching overhead. The above facts motivate us to set the scheduling granularity at the level of a training round. Instead of making scheduling decision for individual task, we assign its tasks of the current rounds to GPUs once we decide to schedule a job.

The final challenge is how to evaluate the amount of "attained service". A straightforward idea is to count the number of scheduled tasks belonging to each job. However, this is an inaccurate estimation because tasks have different running time on GPUs. An example is shown in Fig. 9. Suppose two jobs, J_A and J_B, arrive a cluster of 4 GPUs at the same time. The single-batch training time on 4 different GPUs is shown in the table. For example, processing one task for JA on GPU1 is 3 seconds. We further assume that both jobs will run the same number of rounds. In each round, they have 4 and 6 tasks, respectively. Of course, the number of total training rounds is unknown when jobs arrive. After each training round, we only know whether one additional round is needed or not. It is unclear whether more rounds are needed in the future. If we use the number of scheduled tasks as "attained service", Job A would have higher priority than B, because it has less tasks. However, we find that Job B can run faster on these GPUs, and the average JCT can be shorten if we give B higher priority, as shown in Fig. 9(b).

The above example offers an important hint for us. The "service" should include not only the number of tasks, but also the running time. Formally, both spatial and temporal features of DML jobs should be considered in scheduling algorithm design. A similar observation has been also claimed by Tiresias [18]. However, *Hare* has two important differences from Tiresias. First, Tiresias does not exploit the relaxed scale-fixed synchronization. To run a round of tasks, it requires that the same number of GPUs should be available. Second, Tiresias has proposed the 2DAS (Two-Dimensional Attained Service-Based Scheduler), using the metric of $W_n t_n$ (W_n is the number of assigned GPUs and t_n is GPU running time) to evaluate "attained service". To extend this idea for relaxed scale-fixed synchronization, we let $\mathcal{R}_n$ denote the set of tasks already scheduled on GPUs, and the metric becomes task-GPU time $\sum_{n \in \mathcal{R}_n} t_n$. However, this extension does not work because of the GPU heterogeneity. We use the example in Fig. 10 to explain the reason, where two

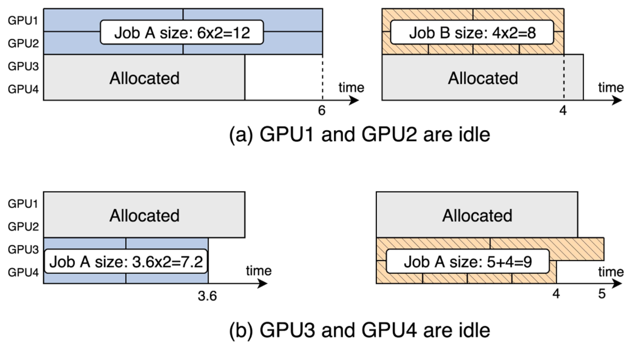

Fig. 10. A toy example to show the uncertain attained services in heterogeneous computing environment.

Algorithm 1: Heterogeneity-Aware Least-Attained Service Algorithm.

```
1: procedure HLAS$\mathcal{N}, \mathcal{M}, \mathcal{Q}$
2:   for $t = 1, 2, \ldots, T$ do
3:     Add newly arrived job to $\mathcal{N}$;
4:     $x^t_{n,m} = 0, \forall n \in \mathcal{N}, m \in \mathcal{M}$;
5:     QueueUpdate($\mathcal{Q}$);
6:     for $m \in \mathcal{M}$ do
7:       if $m$ is idle then
8:         $E = $ JobScheduling($\mathcal{Q}, m$)
9:         $x^t_{n,m} = 1, \forall n \in E$;
10:      end if
11:    end for
12:  end for
13: end procedure
14: Return: $x^t_{n,m}$;
```

Algorithm 2: Queue Update Procedure in HLAS.

```
1: procedure QueueUpdate$\mathcal{N}, \mathcal{Q}$
2:   Update job size $\mathcal{H}_n(t), \forall n \in \mathcal{N}$;
3:   Put jobs to queues $\mathcal{Q}$ according to thresholds;
4: end procedure
```

jobs are the same as in Fig. 9. We assume not all GPUs are available in the beginning. When GPUs 1 and 2 are idle, Job B's task-GPU time is 8 and it has higher priority than Job A whose task-GPU time is 12. However, if GPUs 3 and 4 become idle, the task-GPU time of Jobs A and B is 7.2 and 9, respectively, and Job A's priority is higher than B. This example demonstrates that job priorities depend on which GPUs are available, which is uncertain in heterogeneous computing environment. These observation motivates us to eliminate the influence of GPU heterogeneity before applying LAS.

C. Heterogeneity-Aware Least-Attained Service (HLAS)

Motivated by the above findings, we propose a Heterogeneity-aware Least-Attained Service (HLAS) policy for online job scheduling in *Hare*. The proposed HLAS scheme has two stages: a preparation stage that groups GPUs to wipe off hardware heterogeneity, and a run-time stage that schedules tasks according to priorities. The workflow of HLAS is shown in Fig. 11 and Algorithm 1. The output of our scheduling algorithm is the

Algorithm 3: Job Scheduling in HLAS.

```
1: procedure JobScheduling$\mathcal{Q}, m$
2:   Execution task list $E = \emptyset$;
3:   for $Q_i \in \mathcal{Q}$ do
4:     if $Q_i$ is not $\emptyset$ then
5:       if there is a job $n \in Q_i$ is partially scheduled
         then
6:         Execute $n$ in $m$;
7:       else
8:         $n = $ FIFO($Q_i$);
9:         Execute $n$ in $m$;
10:       end if
11:       $E.append(n)$;
12:     end if
13:   end for
14: end procedure
15: Return: $E$;
```

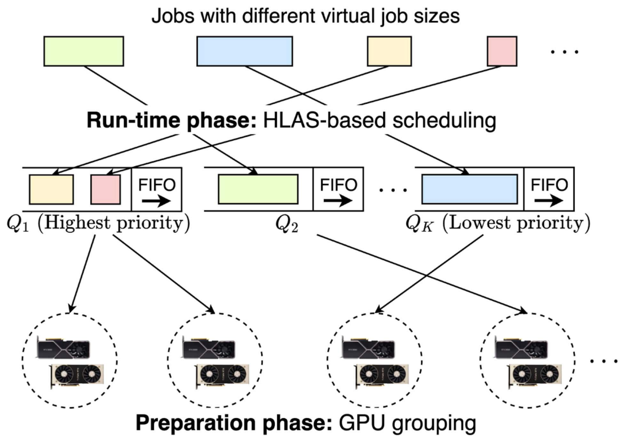
Jobs with different virtual job sizes
Run-time phase: HLAS-based scheduling
FIFO
FIFO
FIFO
Q_1 (Highest priority)
Q_2
Q_K (Lowest priority)
Preparation phase: GPU grouping

Fig. 11. HLAS workflow.

execution decision of tasks, i.e., $x^t_{n,m}$. Specifically, $x^t_{n,m} = 1$ indicates that task n should be executed on GPU m at time slot t, otherwise $x^t_{n,m} = 0$.

Preparation phase (GPU grouping): The objective of GPU grouping is to let all groups exhibit a similar speed for each job. Specifically, we measure the training speed $s_{n,m} \propto \frac{1}{T^c_{n,m}+T^s_{n,m}}$ of each job n running on each GPU m, and feed it to the following optimization problem.

$$\textbf{GPU\_Grouping}: \quad \min_z \max_n (p_n^{\max} - p_n^{\min}),$$

$$\sum_{u \in \mathcal{U}} z_{m,u} = 1, \forall m \in \mathcal{M}; \tag{4}$$

$$p_n^{max} = \max_u \left\{ \sum_{m \in \mathcal{M}} z_{m,u} s_{n,m} \right\}; \tag{5}$$

$$p_n^{min} = \min_u \left\{ \sum_{m \in \mathcal{M}} z_{m,u} s_{n,m} \right\}. \tag{6}$$

In the above formulation, the binary variable $z_{m,u}$ determines whether the GPU m should be assigned to the group u. We let p_n^{max} and p_n^{min} denote the job n's maximum and minimum speeds, respectively, among all groups. After solving this optimization problem, we get a GPU grouping scheme and the average speed of all groups is denoted by $\overline{s}_n$. Based on the average speed $\overline{s}_n$, we can calculate the average one-round time costs for job n, denoted by $\bar{T}_n$.

Run-time phase (task scheduling): We define the job size of n as: $\mathcal{H}_n(t) = R_n^r(t)\bar{T}_n$, where $R_n^r(t)$ is the number of rounds that already run at current timestep t. In addition, we set up K queues $(Q_1, Q_2, \ldots Q_K)$, where Q_1 has the highest priority and Q_K is with the lowest one. Each queue Q_k is associated with two thresholds, E_k^L and E_k^H, which represents the lowest and highest job sizes that can be accommodated in this queue. Some key operations are as follows.

1) **When a job newly arrives**, we always put it into the highest-priority queue Q_1.
2) **When a GPU group is idle**, we find a non-empty queue with the highest priority. Then, we check jobs in this queue to see whether there is a job is partially scheduled, i.e., some tasks of its current round have been assigned to a GPU group and the rest are still in the queue. If yes, we assign the rest tasks of this job to the idle GPU groups. Otherwise, we pick up jobs and schedule them in a first-in-first-out (FIFO) manner. After the execution, we calculate the accumulated job size and demote it to the corresponding queue.

Algorithm analysis: The time complexity of QueueUpdate is $\mathcal{O}(|\mathcal{N}|\log K)$. Checking all queues involves the time complexity of $\mathcal{O}(K)$ and finding a partially scheduled job has the complexity of $\mathcal{O}(|Q|)$. Assuming that $|Q|$ could be as large as $|\mathcal{N}|$ in the worst case, the worst-case time complexity of JobScheduling could be $\mathcal{O}(K|\mathcal{N}|)$. For our proposed HLAS, in each time slot, it updates the queue and then schedules jobs for $|\tilde{\mathcal{M}}|$ GPU groups. Overall, the time complexity of HLAS is $\mathcal{O}(|\mathcal{N}|\log K + |\tilde{\mathcal{M}}|K|\mathcal{N}|)$.

The space complexity of Algorithm 1 is analyzed as follows. To run Algorithm 1, we need to trace the status of queues and jobs. To store queues, the space complexity is proportional to the total number of jobs, i.e., $\mathcal{O}(|\mathcal{N}|)$. For the job size tracking, we need to store each job's size, adding an additional $\mathcal{O}(|\mathcal{N}|)$ space complexity. Overall, the space complexity of Algorithm 1 is $\mathcal{O}(|\mathcal{N}|)$.

V. Enhancement of HLAS With Job Size Prediction

Some recent works [19], [28] have shown the possibility of predicting machine learning job sizes. We find that such a prediction can improve the performance of the HLAS algorithm. A toy example is shown in Fig. 12. Suppose there are three jobs running on a single GPU. They have different numbers of training rounds, but each round contains only a single task with similar running time. This simple setting can let us better focus on studying the benefit of job size prediction. Without job size information, we can obtain the scheduling result shown in Fig. 12(a) by following the HLAS algorithm. We count the running time of a single task as unit time and the average JCT is 6.7. If we know full information of job sizes, an optimal solution, with average JCT of 5.3, can be generated by the shortest first scheduling policy, as shown in Fig. 12(b). However, an accurate

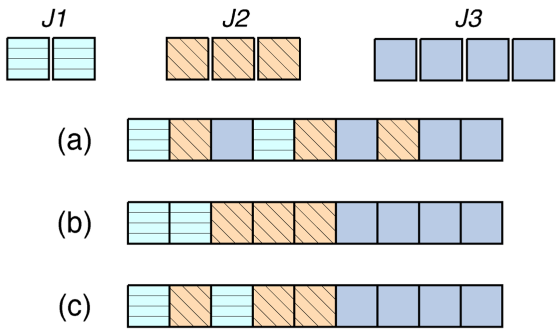

Fig. 12. A toy example to show benefits of using additional prediction information. Suppose there are three jobs that run 2, 3, and 4 rounds, respectively, on a GPU. (a) The running sequence generated by HLAS. (b) The running sequence with full knowledge of job sizes. (c) The running sequence with partial knowledge that job $J3$ will run at least 3 rounds and the sizes of $J1$ and $J2$ are unknown.

prediction of full job size information is almost impossible in practice. Thus, we turn to see whether a partial prediction can bring performance improvement. Suppose that there exists a prediction method, which can give us a hint about job $J3$'s size: $J3$ will run at least 3 rounds. With this information, we can have a scheduling as shown in Fig. 12(c), whose average job completion time is 5.7. In the beginning, we let $J1$ and $J2$ run first, because their sizes are unknown and we suppose they are smaller than $J3$. Both jobs run by following the HLAS until their sizes grows to 3. Since $J1$ and $J2$ have already completed, we assign all GPU time to $J3$ until it finishes. This example shows that job size prediction, even a partial prediction, can help us to make better scheduling decisions to reduce average JCT. Meanwhile, the amount of improvement depends on how much information can be accurately predicted. In the following, we will elaborate how we make the prediction and how to use predicted job sizes in the scheduling algorithm.

A. Conservative Job Size Prediction

Machine learning job size prediction is not new. Peng et al. [19] exploits the convergence trend of loss curves to predict how many training rounds are needed. However, this method requires to accumulate sufficient training losses to catch the convergence trend. Experimental results show that this method has very low accuracy in the first a few rounds. Moreover, this method ignores the relationship among jobs. Consider a case of using neural architecture search (NAS) technique, which launches a group of training jobs running in parallel for competition. Even though we can make a good prediction for a job by fitting its loss curve, this job may be soon killed because its loss descending speed falls behind other competitors.

The above finding motivates us to design a prediction method by jointly using job running states and historical information. As shown in Fig. 13, we also use a loss fitting method to make an initial prediction. Different from existing work aiming at predicting full job sizes, our method is rather conservative and it predicts whether a job will run additional α rounds. Then, we adjust this initial prediction by comparing it with historical records and states of other related jobs, if they exist. Specifically, we search the historical records to see whether there are similar

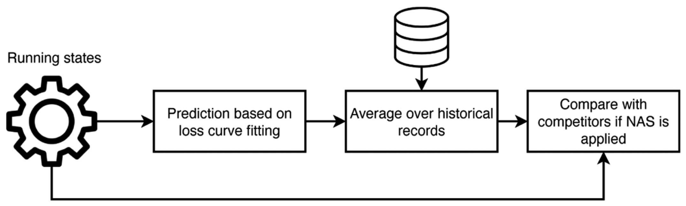
Running states
Prediction based on loss curve fitting
Average over historical records
Compare with competitors if NAS is applied

Fig. 13. Conservative prediction process.

jobs (with the same names, or submitted by the same users) in history. If yes, we update the prediction by averaging α and practical rounds in history. After that, we continue to check whether this job belongs to a NAS group. We make more conservative prediction by reducing α if this job belongs to a NAS group and it falls far behinds others.

B. HLAS-P Scheduling

We propose an HLAS-P scheduling algorithm to further improve the performance by using prediction information. For each job, we denote its expected job size as $H_n(t) = (R_n^r(t) + R_n^p(t))\bar{T}_n$, where $R_n^r(t)$ is the number of rounds that already run and $R_n^p(t)$ is the predicted additional number of rounds. Similar with HLAS, we still maintain K queues $\{Q_1, Q_2, \ldots, Q_K\}$, but in each queue, we have two sub-queues q_k^1 and q_k^2, where q_k^1 has higher priority than q_k^2. In addition, tasks in q_k^1 are sorted according to $R_n^p(t)$ in a descending order. The job with larger $R_n^p(t)$ is scheduled first. The tasks in q_k^2 are scheduled using a FIFO policy.

When a job goes out of the prediction module, we put it into corresponding queue Q_k according to its expected job size $R_n(t)$. Then we check whether $R_n^p(t)$ is zero. If yes, this job is insert into sub-queue q_k^2. Otherwise, it is put into the queue q_k^1. Note that when a job in the queue q_k^1 is scheduled to run, the value of $R_i^p(t)$ decreases while $R_n^r(t)$ grows, but keep the same $R_n(t)$. The value of $R_n(t)$ changes only when this job belongs to q_k^2.

VI. Performance Evaluation

In this section, we first introduce our experimental settings and then present the results of the testbed and simulations.

A. Experimental Settings

We build a testbed consisting of 15 heterogeneous GPUs (8 V100 s, 4 T4 s, 1 K80, and 2 M60 s), which are deployed on 4 Amazon EC2 instances. All GPUs are equipped with PCIe-3$\times$16 (15.75 GB/s). Each instance is powered by NVIDIA driver 418.21, CUDA 10.1 and cuDNN 8.0.4, running Ubuntu 18.04 with Linux kernel version 5.4. All instances are connected via the 25 Gbps Ethernet.

We have developed a trace-driven simulator to evaluate *Hare* in large-scale settings. The simulator is built in Python, and emulates the execution of DML jobs using the traces collected from the testbed. The job arrival time is set according to the trace in Google cluster [26].

We create some DML jobs based on 8 popular models across domains of computer vision (CV), natural language processing

TABLE II
Deep Learning Jobs Used in Our Experiments.

Type	Model	Dataset	Batch Size	Percentage
CV	VGG-19 [29]	Cifar10	128	25%
CV	ResNet50 [30]	Cifar100	64	
CV	Inception V3 [31]	Cifar100 [32]	32	
NLP	Bert_base [33]	SQuAD	32	25%
NLP	Transformer [34]	WMT16	128	
Speech	DeepSpeech [35]	ComVoice	8	25%
Rec.	FastGCN [36]	Cora	128	25%
Rec.	GraphSAGE [37]	Cora	16	

(NLP), speech, and recognition (Rec.). The details of these models are shown in Table II. In the default setting, each type of jobs accounts for 25% of the total workload. All jobs are implemented in PyTorch 1.8.1, and they are trained using synchronous PS scheme. Since the original datasets of SQuAD and WMT16 are too large and the corresponding training would run for days, we downscale them so that they can complete within hours.

We compare *Hare* with following schemes.

Gavel_FIFO: FIFO (First In First Out) is a default job scheduling algorithm in many traditional batch job processing systems [39]. It schedules jobs in an order according to their arrival time. Gavel [7] customizes FIFO for heterogeneous GPUs by assigning jobs to fastest available GPUs. If the number of idle GPUs is insufficient, this job needs to wait until demanded GPUs are available.

SRTF (Shortest Remaining Time First): SRTF has been widely adopted to minimize total job completion time. It always schedule jobs that could complete earlier.

Sched_Homo [4]: We denote a recent scheduling algorithm [4] designed for homogeneous GPUs by Sched_Homo. Similar to *Hare*, it aims to minimize the weighted ML job completion time by exploiting both inter-job and intra-job parallelism. However, job-level preemption is not allowed.

Sched_Allox [6]: We consider the ML job scheduling algorithm proposed by Allox [6]. The GPU heterogeneity has been fully exploited, but it does not consider the intra-job parallelism.

B. Resutls on Testbed

We first study the benefits of fast task switching by showing the average switching time of different jobs in Table III. A default task switching scheme, without any optimization, needs more than 3000 ms for all jobs. PipeSwitch can reduce the average switching time to 12.57 ms for Bert_base and less for others. The maximum switching time of *Hare* is no more than 6 ms. The proportion of task switching time to the total task time is also shown in the table. We can see that *Hare* constrains the task switching overhead within 2% for most of models, and the largest overhead under FastGCN is no more than 5%. These results justify our assumption that task switching time is negligible in the scheduling algorithm design.

The total weighted job completion time (JCT) of several schemes running on the testbed and the simulator is shown in Fig. 14. Compared with other schemes, *Hare* can reduce total weighted JCT by 47.6% to 75.3%, significantly outperforming other schemes. Fig. 15 shows the cumulative distributed function (CDF) of JCT of all jobs. We can see that about 90.5% of jobs can complete within 25 minutes by *Hare*, while Sched_Allox

TABLE III
AVERAGE TASK SWITCHING TIME OF DIFFERENT JOBS

	VGG19	ResNet50	Inception V3	Bert_base	Transformer	DeepSpeech	FastGCN	GraphSAGE
Default	3288.94 ms (98.21%)	5961.16ms (97.37%)	7807.43 ms (96.99%)	9016.99 ms (93.95%)	5257.17 ms (95.41%)	5125.64 ms (94.15%)	5327.24 ms (98.47%)	5213.54 ms (98.29%)
PipeSwitch [38]	4.01 ms (2.40%)	4.75 ms (5.46%)	5.03 ms (2.39%)	12.57 ms (1.99%)	10.34 ms (2.03%)	8.91 ms (1.59%)	2.86 ms (7.56%)	2.42 ms (8.64%)
Hare	2.77 ms (1.82%)	2.04 ms (3.71%)	2.46 ms (1.43%)	5.03 ms (1.13%)	5.79 ms (1.36%)	4.27 ms (1.25%)	1.83 ms (4.53%)	0.96 ms (3.36%)

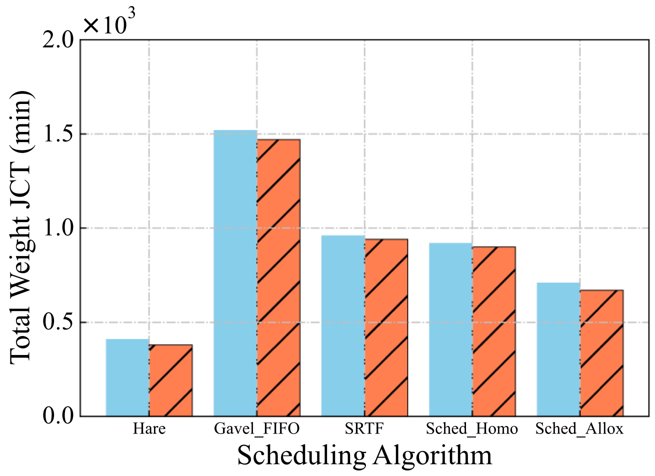

Fig. 14. The results in testbed.

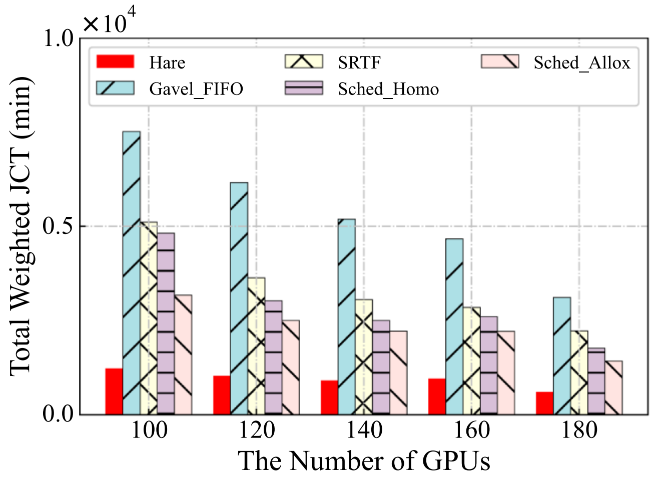

Fig. 16. Performance under different number of GPUs.

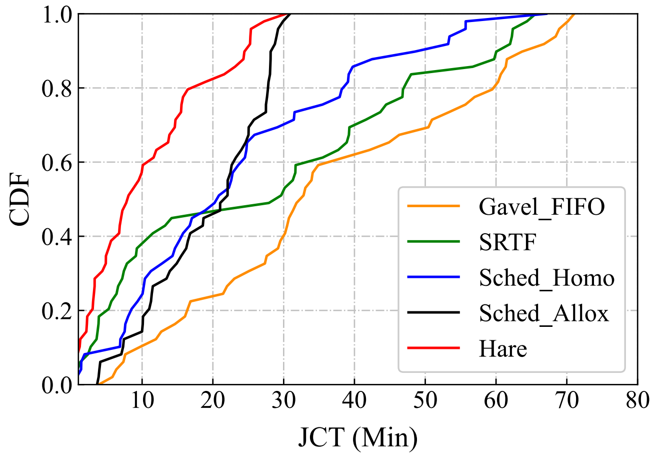

Fig. 15. CDF of job completion time.

and Sched_Homo can complete only 66.7% and 56.5%, respectively. That is because Allox misses the optimization chances brought by intra-job parallelism, and Sched_Homo is GPU-heterogeneity-oblivious, leading to low GPU utilization.

C. Simulation Results of Offline Scheduling

Large-scale experiments are conducted using the simulator. As we have shown in Fig. 14, the maximum performance gap between the testbed and simulator is only 5%, which demonstrates that the simulator can offer sufficient simulation accuracy. The gap is mainly because the error in prediction of training time and switching cost.

We study the influence of number of GPUs in Fig. 16. The number of ML jobs is set to 200. The weighted JCT of all schemes decreases as more GPUs are used. *Hare* always outperforms other schemes under all cases. Sched_Allox is slower than *Hare* by about 2x, but it is still significantly faster than others, thanks to its heterogeneity-aware design. Although Gavel_FIFO schedules jobs with the consideration of heterogeneity, it still has the largest weighted JCT since it has no optimization in scheduling.

We then consider 160 GPUs and change the number of jobs from 100 to 300 to see how it affects the performance. As shown in Fig. 17, as the number of jobs increases, the total weighted JCT grows under all schemes. Meanwhile, the performance gaps between *Hare* and other schemes become bigger. For example, *Hare* outperforms others by 54.6%–80.5% when processing 300 jobs. It demonstrates that *Hare* can use these GPUs in a more efficient way, to minimize the total weighted JCT.

We study the influence of GPU heterogeneity in Fig. 18. We consider 160 GPUs and 200 jobs. We set different heterogeneity levels by selecting a different combination of GPUs. For the low heterogeneity level, we only choose V100 GPUs for training. We select the combination of (V100×K80) GPUs as the middle heterogeneity level while selecting the combination of (V100×T4×K80×M60) GPUs as the high heterogeneity level. We find the gaps between *Hare* and other schemes become bigger as the increasing of heterogeneity level. The main reason is the higher heterogeneity level results in lower resource utilization in heterogeneity-oblivious schemes. Although Sched_Allox suffers a slight influence from the heterogeneity

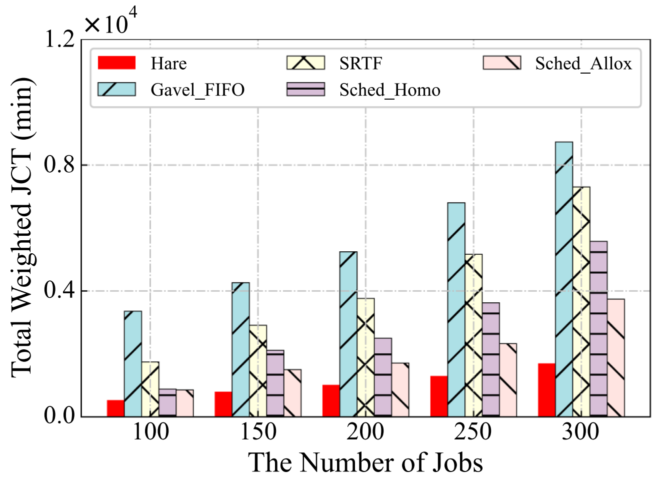

Fig. 17. Performance under different number of jobs.

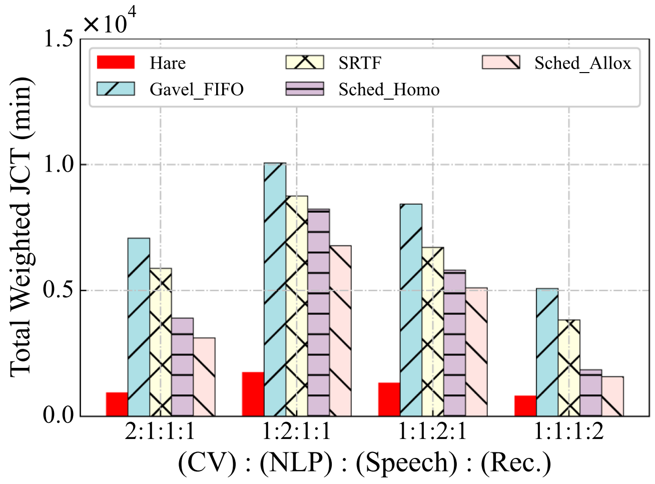

Fig. 19. Performance under different fractions of jobs.

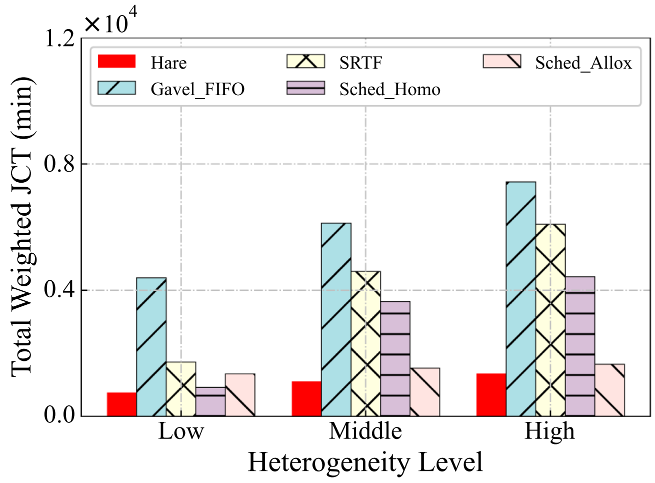

Fig. 18. Performance under different heterogeneity levels.

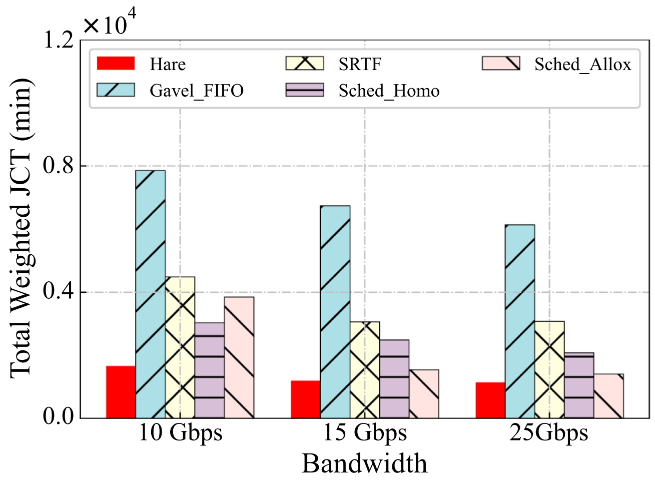

Fig. 20. Performance under different bandwidth.

level, its performance still lags behind *Hare* by 2× since there is no consideration of intra-job parallelism optimization.

We investigate how job type affects the performance by changing their proportions. The results are shown in Fig. 19. In the default setting, each type of jobs account for 25%. In each experiment, we then increase one of them and keep others the same. The x axis of Fig. 19 shows the ratio of different job types. When we increase the proportion of NLP jobs, the total weighted JCT of all schemes increases since NLP jobs involve heavier training workloads (i.e., more training rounds and more training time). On the other hand, all schemes have smaller weighted JCT when more recognition jobs are added, because they have less workloads. Although *Hare* is affected by the job proportion, it always achieves the best performance due to the sophisticated scheduling algorithm.

We change the speed of the network connecting GPUs and study its influence in Fig. 20. The results are in alignment with our intuition that faster networks can accelerate the ML training. However, such acceleration is not linear with the network speed since the training time becomes the main bottleneck as the decreasing of the synchronization time. For example, *Hare*'s weighted JCT decreases by only 31.2%, even though increase the network speed from 10 Gbps to 25 Gbps.

Fig. 21 shows the performance under different batch sizes, where B_0 stands for the default batch size configuration. We can see that batch size has no big influence to all schemes except Sched_Homo. That is because larger batch size leads to longer training time, and there is more GPU idle time in Sched_Homo.

D. Simulation Results of Online Scheduling

We conduct simulations of online scheduling using the traces from [26]. We compare our online scheduling algorithm HLAS with four other algorithms: FIFO, Sched_Homo [4], Sched_Allox [6], and 2D-LAS. For Sched_Homo and Sched_Allox algorithms, we use a unit job size to disable the job size information. The other two algorithms, i.e., FIFO and 2D-LAS [18], do not need job size information, the same as our HLAS. Overall, HLAS always outperforms others since it can capture the online information by defining the attained service as the job sizes while considering GPU heterogeneity.

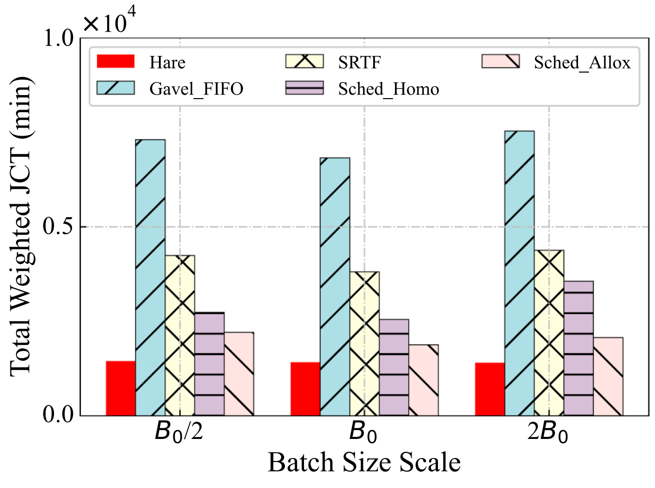

Fig. 21. Performance under different batch sizes.

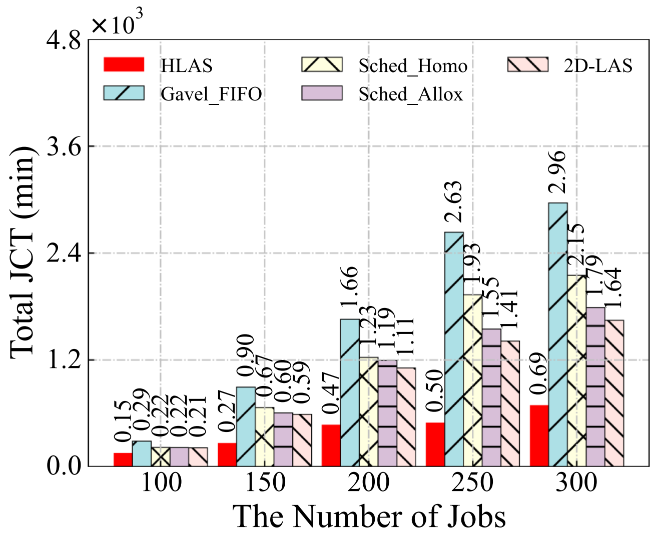

Fig. 23. Performance under different number of jobs.

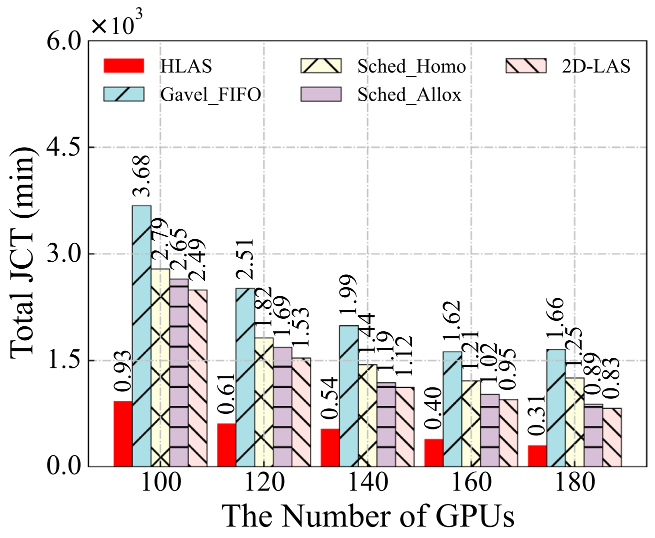

Fig. 22. Performance under different number of GPUs.

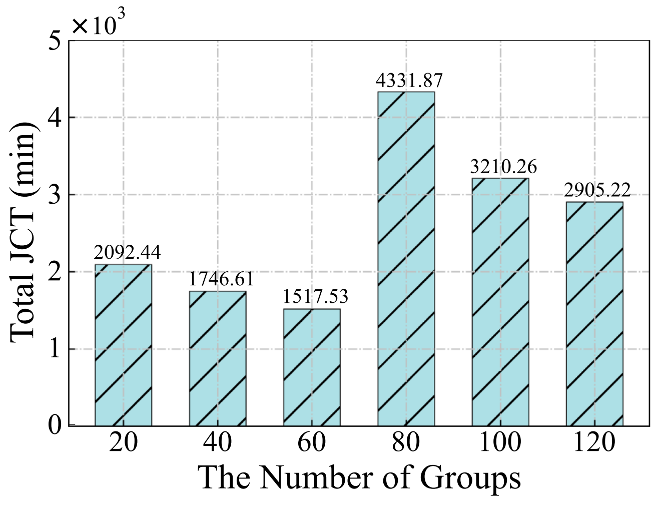

Fig. 24. Impact of the number of groups.

The workloads and GPU types are the same with the previous setting.

We first study the impact of the number of GPUs in the online scenario, as shown in Fig. 22. We set the number of jobs as 200 and increase the number of GPUs from 100 to 180. The results show that our online scheduling algorithm can achieve significant performance benefits compared with others. Specifically, our online scheduling algorithm reduce the average JCT by about 2.04× compared with 2D-LAS. Although 2D-LAS can outperform than FIFO by scheduling jobs with least attained services in high priorities, they define the attained service without consideration of GPU heterogeneity.

We then study the impact of the number of jobs and show the results in Fig. 23. We increase the number of jobs from 100 to 300 and fix the number of GPUs as 160. Similar to the offline scenario, the average JCT grows as the number of jobs increases. We can also find that our online scheduling algorithm can always outperform than others. Specifically, our online scheduling algorithm outperforms others by about 1.69× and 1.1×, respectively. The gap between FIFO and our HLAS increases since there are more large jobs will be scheduled first in FIFO, making a worse average JCT.

We then set the number of GPUs and jobs as 160 and 200, respectively. As shown in Fig. 24, we can find the total JCT decreases with more GPU groups. It is because that the more GPU groups provides larger inter-job parallelism. However, we also find that the average JCT increases with more GPU groups. The reason is that it is hard to ensure homogeneity between groups, leading more group idle time.

We further analyze the algorithm cost with different number of jobs. We set the arrival time of all jobs as 0, which means all jobs can be scheduled once at the start time slot. We set the number of GPUs and groups as 160 and 40, respectively. The results are shown in Fig. 25. Although the algorithm cost grows when the number of jobs increases, the trend is sub-linear, which means that our scheduling algorithm can handle a large number of jobs.

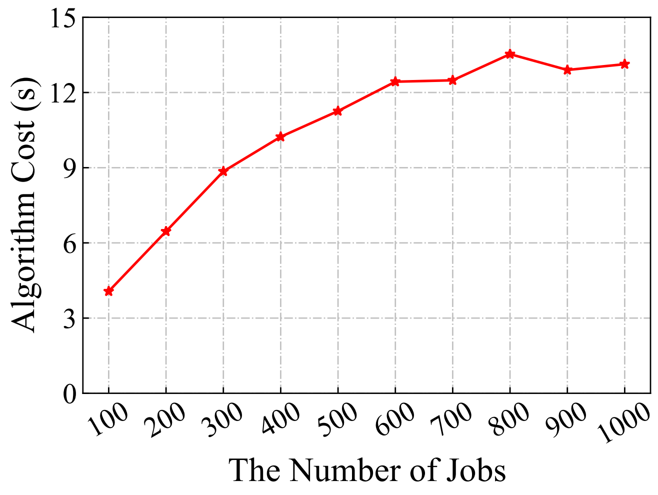

Fig. 25. Algorithm costs with different number of jobs.

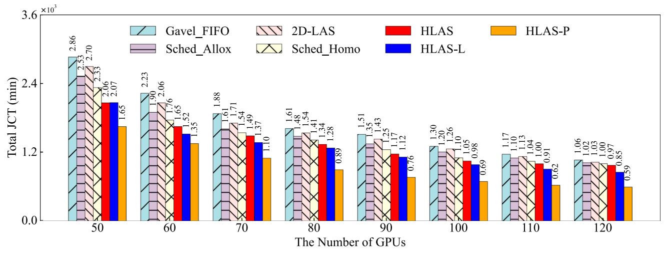

Fig. 26. The performance of HLAS-P with different number of GPUs.

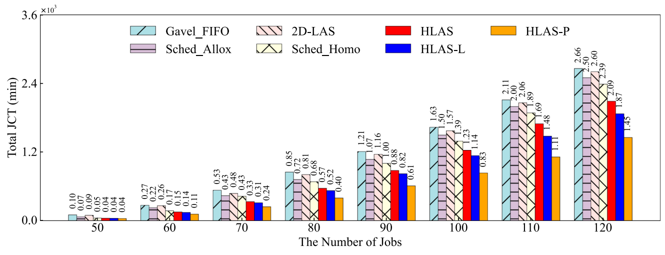

Fig. 27. The performance of HLAS-P with different number of jobs.

We finally study the performance of HLAS-P. We compare the total JCT of HLAS-P with three methods, i.e., FIFO, HLAS, and HLAS-L. For the HLAS, we calculate virtual job sizes with only attained services. In contrast, for HLAS-L, we add predicted job size by the loss fitting technique [19]. The results are shown in Figs. 26 and 27. We can find that HLAS-P method can always outperform others, and achieves additional improvement of 1.42×, compared to HLAS. The HLAS-L method provides a trivial improvement compared with HLAS, which is caused by the fitting errors. Specifically, at the first a few rounds, the fitting model has not been well-tuned, and it will give wrong prediction errors for DML jobs, which significantly degrades the scheduling performance. In contrast, HLAS-P adopted in *Hare* utilizes not only loss curves but also traces information, which gives a more accurate prediction. Therefore, our HLAS-P method can always outperform others. When the number of GPU doubles, the performance cannot double as well. The key reason is that DML jobs require synchronization at each training round. This synchronization process ensures that all the updates from different GPUs are aggregated to update the global model. The total round training time is often limited by the slowest (straggler) GPU in the cluster. Hence, even if the number of GPUs doubles, the speedup in the completion time of each DML job is not directly proportional due to this bottleneck.

VII. Related Work

Distributed Machine Learning: Distributed machine learning on GPUs has been widely adopted to accelerate model training on large datasets. Typically, We can assign and synchronize workloads on GPUs in two different ways, which are referred to as model parallelism [40] and data parallelism [41]. In the model parallelism, each GPU trains a partition of the model with the entire dataset. In data parallelism, each GPU maintains a complete model and trains it using a subset of data. The model gradients are periodically synchronized across GPUs using All-Reduce [33] or Parameter Server (PS) [42] scheme. In particular, the PS scheme is popular due to its simplicity, and we also use it in our work. Specifically, the training process contains multiple rounds. In each round, training workloads are shared by multiple GPUs, which are also called workers. Each worker computes its local gradients by using mini-batch stochastic gradient descent (SGD) method. Then they send gradients to the parameter server, which updates the model for the next-round training.

Job scheduling for machine learning: Job scheduling, which determines when and where each job should run, is the most fundamental and critical issue for distributed machine learning. Early studies follow the idea of traditional batch job scheduling by treating each job as an unsplittable unit and schedule them on different GPUs [39]. Later, some works have exploited the intra-job parallelism, i.e., tasks in the same training round of a job can run in parallel, which can significantly enhance learning performance. Optimus [19] allocates resources to ML jobs by learning a throughput model with respect to various resource allocation. Pollux [3] studies different resource allocation for ML jobs by observing the throughput and statistical efficiency during training. Zhang et al. [4] design an online algorithm that selects the amount of resources for each job to minimize the total job completion time. Although the above works have exploited both inter-job and intra-job parallelism, they consider homogeneous GPUs and forbid GPU preemption during job execution. More recently, some works [14], [28], [43] propose multi-resource scheduling for DML jobs, which schedule not only GPUs but also auxiliary resources, such as CPUs, memory, and so on. These works also consider homogeneous GPUs and are orthogonal to *Hare*.

Recently, GPU-heterogeneity becomes popular as the expansion of data centers and it has attracted significant research attention. Gandiva$_{\text{fair}}$ [5] proposes an automated trading mechanism to support time-slicing resource sharing among different jobs while improving the cluster efficiency. Gavel [7] develops a heterogeneity-aware scheduler to generate different scheduling policies for different kinds of jobs. However, Gandiva$_{\text{fair}}$ and Gavel schedule jobs based on given time slice length. Such a coarse-grained scheduling manner leaves a large optimization space for performance improvement. Moreover, they ignore the task switching cost. Allox [6] transforms the job scheduling problem into a min-cost bipartite matching to provide dynamic fair allocation, but it conducts job-level scheduling and ignores the intra-job parallelism.

VIII. Conclusion

We present Hare, a system enabling efficient multiple DML job scheduling on the heterogeneous GPU cluster. We propose both offline and online task scheduling algorithms to minimize the average job completion time while improving the GPU resource utilization. For online task scheduling, our proposed algorithm is based on the Heterogeneity-aware Least-Attained Service (HLAS) policy and schedules DML jobs without any knowledge of job sizes. In addition, we further enhance the proposed online scheduling algorithm by introducing additional prediction information. We find that even a part of prediction knowledge can benefit the scheduling performance. We demonstrate the performance of Hare through experiments on both the small-scale testbed and the large-scale trace-driven simulator. Hare can significantly outperform existing works.

References

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

[2] W. Xiao et al., "Gandiva: Introspective cluster scheduling for deep learning," in *Proc. 13th USENIX Symp. Operating Syst. Des. Implementation*, 2018, pp. 595–610.

[3] A. Qiao et al., "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *Proc. 15th USENIX Symp. Operating Syst. Des. Implementation*, 2021, pp. 1–18.

[4] Q. Zhang, R. Zhou, C. Wu, L. Jiao, and Z. Li, "Online scheduling of heterogeneous distributed machine learning jobs," in *Proc. Proc. 21st Int. Symp. Theory, Algorithmic Found. Protocol Des. Mobile Netw. Mobile Comput.*, 2020, pp. 111–120.

[5] S. Chaudhary, R. Ramjee, M. Sivathanu, N. Kwatra, and S. Viswanatha, "Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning," in *Proc. Proc. 15th Eur. Conf. Comput. Syst.*, 2020, pp. 1–16.

[6] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "Allox: Compute allocation in hybrid clusters," in *Proc. Proc. 15th Eur. Conf. Comput. Syst.*, 2020, pp. 1–16.

[7] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *Proc. 14th USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 481–498.

[8] F. Chen, P. Li, C. Wu, and S. Guo, "Hare: Exploiting inter-job and intra-job parallelism of distributed machine learning on heterogeneous GPUs," in *Proc. 31st Int. Symp. High- Perform. Parallel Distrib. Comput.*, 2022, pp. 253–264.

[9] "Automl." [Online]. Available: http://www.ml4aad.org/automl/

[10] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2815–2823.

[11] L. Yang et al., "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *Proc. 57th ACM/IEEE Des. Automat. Conf.*, 2020, pp. 1–6.

[12] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "SLAQ: Quality-driven scheduling for distributed machine learning," in *Proc. Symp. Cloud Comput.*, 2017, pp. 390–404.

[13] W. Gao, Z. Ye, P. Sun, Y. Wen, and T. Zhang, "Chronus: A novel deadline-aware scheduler for deep learning training jobs," in *Proc. ACM Symp. Cloud Comput.*, 2021, pp. 609–623.

[14] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, and X. Jin, "Multi-resource interleaving for deep learning training," in *Proc. ACM SIGCOMM Conf.*, 2022, pp. 428–440.

[15] K. Mahajan et al., "Themis: Fair and efficient GPU cluster scheduling," in *Proc. 17th USENIX Symp. Networked Syst. Des. Implementation*, 2020, pp. 289–304.

[16] M. Nuyens and A. Wierman, "The foreground–background queue: A survey," *Perform. Eval.*, vol. 65, no. 3/4, pp. 286–307, 2008.

[17] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 393–406, 2015.

[18] J. Gu et al., "Tiresias: A GPU cluster manager for distributed deep learning," in *Proc. 16th USENIX Symp. Networked Syst. Des. Implementation*, 2019, pp. 485–500.

[19] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. 13th Eur. Conf. Comput. Syst.*, 2018, pp. 1–14.

[20] W. Xiao et al., "Antman: Dynamic scaling on GPU clusters for deep learning," in *Proc. 14th USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 533–548.

[21] "Apache hadoop," 2021. [Online]. Available: http://hadoop.apache.org/

[22] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of np-completeness," *J. Symbolic Log.*, vol. 48, no. 2, 1983.

[23] I. A. Rai, G. Urvoy-Keller, and E. W. Biersack, "Analysis of LAS scheduling for job size distributions with high variance," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2003, pp. 218–228.

[24] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "{Information-Agnostic} flow scheduling for commodity data centers," in *Proc. 12th USENIX Symp. Networked Syst. Des. Implementation*, 2015, pp. 455–468.

[25] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 443–454.

[26] "Google cluster traces," 2019. [Online]. Available: http://github.com/google/cluster-data

[27] 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.

[28] Q. Weng et al., "{MLaaS } in the wild: Workload analysis and scheduling in { Large-Scale } heterogeneous { GPU} clusters," in *Proc. 19th USENIX Symp. Networked Syst. Des. Implementation*, 2022, pp. 945–960.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015, pp. 1–14. [Online]. Available: http://arxiv.org/abs/1409.1556

[30] 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.

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.

[32] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. 8th IEEE Int. Conf. Comput. Vis.*, 2001, pp. 416–423.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics*, 2019, pp. 4171–4186.

[34] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[35] A. Hannun et al., "Deep speech: Scaling up end-to-end speech recognition," 2014, *arXiv:1412.5567*.

[36] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *Proc. 6th Int. Conf. Learn. Representations*, 2018, pp. 1–15. [Online]. Available: https://openreview.net/forum?id=rytstxWAW

[37] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[38] Z. Bai, Z. Zhang, Y. Zhu, and X. Jin, "PipeSwitch: Fast pipelined context switching for deep learning applications," in *Proc. 14th USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 499–514.

[39] M. Zaharia et al., "Spark: Cluster computing with working sets," *USENIX HotCloud*, vol. 10, no. 10/10, 2010, Art. no. 95.

[40] D. Narayanan et al., "PipeDream: Generalized pipeline parallelism for dnn training," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, 2019, pp. 1–15.

[41] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in tensorflow," 2018, *arXiv: 1802.05799*.

[42] E. P. Xing et al., "Petuum: A new platform for distributed machine learning on Big Data," *IEEE Trans. Big Data*, vol. 1, no. 2, pp. 49–67, Jun. 2015.

[43] J. Mohan, A. Phanishayee, J. Kulkarni, and V. Chidambaram, "Looking beyond {GPUs } for { DNN} scheduling on {Multi-Tenant} clusters," in *Proc. 16th USENIX Symp. Operating Syst. Des. Implementation*, 2022, pp. 579–596.

Fahao Chen is currently working toward the PhD degree in the Graduate School of Computer Science and Engineering, The University of Aizu, Japan. His research interests mainly focus on cloud/edge computing, graph learning, and distributed machine learning systems. He is awarded the Japan Society for the Promotion of Science (JSPS) Research Fellowship for Young Scientists.

Peng Li (Senior Member, IEEE) is currently a senior associate professor in the University of Aizu, Japan. His research interests mainly focus on cloud/edge computing, Internet-of-Things, machine learning systems, as well as related wired and wireless networking problems. He has published more than 100 technical papers on prestigious journals and conferences. He serves as the chair of SIG on Green Computing and Data Processing in IEEE ComSoc Green Communications and Computing Technical Committee. He won the Best Paper Award of IEEE TrustCom 2016. He supervised students to win the First Prize of IEEE ComSoc Student Competition, in 2016. He is the editor of *IEICE Transactions on Communications*, and *IEEE Open Journal of the Computer Society*.

Celimuge Wu (Senior Member, IEEE) received the PhD degree from the University of Electro-Communications, Tokyo, Japan, in 2010. He is a professor with the University of Electro-Communications. His research interests include Vehicular Networks, Internet-of-Things, Edge Computing, and Application of Machine Learning in Wireless Networking and Computing. He serves as an associate editor of IEEE Transactions on Network Science and Engineering, IEEE Transactions on Green Communications and Networking, and IEEE Open Journal of the Computer Society. He is a recipient of the 2021 IEEE Communications Society Outstanding Paper Award, 2021 IEEE Internet of Things Journal Best Paper Award, IEEE Computer Society 2020 Best Paper Award, and IEEE Computer Society 2019 Best Paper Award Runner-Up.

Song Guo (Fellow, IEEE) is a full professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology. He also holds a Changjiang Chair Professorship awarded by the Ministry of Education of China. His research interests include Big Data, edge AI, mobile computing, and distributed systems. With many impactful papers published in top venues in these areas, he has been recognized as a Highly Cited Researcher (Web of Science) and received more than 12 Best Paper Awards from IEEE/ACM conferences, journals and technical committees. He is the editorin chief of IEEE Open Journal of the Computer Society. He has served on IEEE Communications Society Board of Governors, IEEE Computer Society Fellow Evaluation Committee, and editorial board of a number of prestigious international journals, such as IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Cloud Computing, and IEEE Internet of Things Journal. He has also served as a chair of organizing and technical committees of many international conferences. He is an ACM Distinguished Member.