Resource Heterogeneity-Aware and Utilization-Enhanced Scheduling for Deep Learning Clusters

Abeda Sultana, Nabin Pakka, *Graduate Student Member, IEEE*, Fei Xu, Xu Yuan, Li Chen, *Senior Member, IEEE*, and Nian-Feng Tzeng

***Abstract*—Scheduling deep learning (DL) models to train on powerful clusters with accelerators like GPUs and TPUs, presently falls short, either lacking fine-grained heterogeneity awareness or leaving resources substantially under-utilized. To fill this gap, we propose a novel task-level heterogeneity-aware scheduler for DL clusters, *Hadar*, based on an optimization framework able to boost cluster resource utilization. *Hadar* leverages the performance traits of DL jobs on a heterogeneous DL cluster to make scheduling decisions across both spatial and temporal dimensions. It characterizes the task-level performance heterogeneity for optimization and involves the primal-dual framework employing a dual subroutine, to solve the optimization problem and guide the scheduling design. Our trace-driven simulation with representative DL model training workloads demonstrates that *Hadar* accelerates the total training time duration by 1.20× when compared with its state-of-the-art heterogeneity-aware counterpart, Gavel. Further, our *Hadar* scheduler is enhanced to *HadarE* by forking each job into multiple copies to let a job train concurrently on heterogeneous GPUs resided on separate available cluster nodes (i.e., machines or servers) for resource utilization enhancement. *HadarE* is evaluated extensively on physical DL clusters for comparison with *Hadar* and Gavel. With substantial enhancement in cluster resource utilization (by 1.45×), *HadarE* exhibits considerable speed-ups in DL model training, reducing the total training time duration by 50% (or 80%) on an Amazon's AWS (or our lab) cluster, while producing trained DL models with consistently better inference quality than those trained by *Hadar*.**

***Index Terms*—Deep learning, optimization, resource heterogeneity, resource utilization, scheduling.**

I. Introduction

DEEP learning (DL) applications are ubiquitous nowadays across various domains, including speech recognition, natural language processing [2], super-computing, social media [3], among others. To facilitate the ever-increasing demand for DL training [4], large enterprises and cloud providers [5], [6], [7] have constructed powerful DL clusters, which usually incorporate specialized accelerators, such as GPUs, TPUs, and FPGAs, to accelerate DL model training with intricate architectures. The wide adoption of DL models calls for training multiple of them concurrently on such DL clusters with expensive resources. Efficiently scheduling multiple DL training jobs is thus required to yield high training performance, measured by cluster-wide resource utilization, the total training time duration, the average job completion time, *etc.*

To this end, existing efforts have proposed a number of GPU cluster schedulers (*e.g.,* [4], [8], [9]) for DL model training. However, those schedulers either lack the awareness of job or under-utilize available resources, leading to undesirably low performance for DL clusters. It has been observed in [10] that DL training jobs show heterogeneous performance behavior across accelerator devices of different types, due to various architectural differences. For example, a ResNet-50 model achieves a nearly 10× speedup when trained on an NVIDIA V100 GPU versus a K80 GPU, while an A3C Deep Reinforcement Learning model only exhibits 2× acceleration. In light of such observations, Gavel has been proposed [10] as a heterogeneity-aware cluster scheduler, which is the first to address the aforementioned performance heterogeneity of DL training jobs across multi-type accelerators in a cluster. It utilizes an optimization-based scheduling framework to specifically account for job placement and performance heterogeneity. However, it does not explicitly characterize performance heterogeneity at a fine-grained task level, with DL workloads scheduled at the task level. If a job requires 4 V100 GPUs, but the cluster has 3 V100 and 3 K80 GPUs available, the job cannot proceed and must wait for the next scheduling round, which signifies that Gavel is scheduling heterogeneity agnostic. This limitation highlights the need for a more sophisticated

Received 22 December 2024; revised 8 February 2026; accepted 4 March 2026. Date of publication 11 March 2026; date of current version 12 May 2026. This work was supported in part by the National Science Foundation under Grant OIA-2019511, Grant OIA-2327452, Grant CNS-2315613, and Grant CNS-2348452, in part by Louisiana Board of Regents under Contract LEQSF(2019-22)-RDA- 21, in part by NSFC under Grant 62372184, and in part by STC of Shanghai Municipality under Grant 22DZ2229004. An earlier version of this paper was presented at the 38th IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2024. [DOI: 10.1109/IPDPS57955.2024.00066]. Recommended for acceptance by A. C. Melo. *(Corresponding author: Li Chen.)*

Abeda Sultana, Nabin Pakka, Li Chen, and Nian-Feng Tzeng are with the University of Louisiana at Lafayette, Lafayette, LA 70504 USA (e-mail: li.chen@louisiana.edu).

Fei Xu is with the East China Normal University, Shanghai 200062, China.

Xu Yuan is with the University of Delaware, Newark, DE 19716 USA.

Digital Object Identifier 10.1109/TC.2026.3672471

and flexible scheduler that can make the best use of the available cluster resources by accommodating job-level performance heterogeneity.

To bridge this gap, we first introduce a new fine-grained heterogeneity-aware scheduler, named *Hadar* [1], for a cluster shared by DL training jobs. The essence of *Hadar* relies on the problem formulation and optimization framework for task-level resource allocation across both temporal and spatial dimensions. Trace-driven simulation is adopted to evaluate *Hadar*, for comparison with its previous counterparts, including Gavel [10], Tirsias [4], and YARN-CS [6]. Simulation results demonstrate that *Hadar* solidly outperforms in terms of such metrics as resource utilization, total training time duration, and scalability.

Next, *Hadar* is further enhanced by forking every training DL job into multiple copies for possible concurrent execution on heterogeneous GPUs residing on different nodes (i.e., machines or servers) of a DL cluster, boosting cluster resource utilization. This way enables a DL training job to run on various types of GPUs at different nodes simultaneously, if available, for enhancing cluster resource utilization, arriving at *Hadar Enhancement* (or *HadarE* for short). Note that the DL jobs are executed as smaller tasks across multiple workers (or GPUs). During the course of training, *HadarE* lets any unfinished task be executed on as many available GPU-equipped nodes as possible due to its forked copies in existence, unlike *Hadar*, which schedules one task to run merely on a single GPU-equipped node even when another GPU-equipped node is idle. Our *HadarE* enables starting its scheduling immediately and effectively without undergoing job profiling *a priori*, common to earlier schedulers. For evaluating *HadarE* in comparison with *Hadar* and Gavel, we conduct real-world experiments on physical DL clusters leased from the AWS Cloud and at our research lab.

Challenges of existing job schedulers (Tiresias [4], Gandiva [8], and Gavel [10]) for DL clusters lie in lacking both scheduling heterogeneity and task-level heterogeneity (as listed in Table I), making them often under-utilize cluster resources and presenting opportunities for scheduling performance improvement. While both Gandiva and Gavel scheduled jobs on heterogeneous DL clusters (unlike earlier Tiresias, which aimed at only homogeneous DL clusters), they do not schedule a job onto any node with heterogeneous resources and wait for enough homogeneous resources able to accommodate the job, to be available. To meet the challenges by embracing scheduling heterogeneity (as can be seen in Table I), *Hadar* lets a job run on available heterogeneous resources in different rounds without waiting for the same type of resource. However, it does not schedule any tasks to heterogeneous resources or multiple nodes in a given round as *HadarE*, which supports task-level heterogeneity (see Table I) to yield a lower average job completion time (JCT), a lower total time duration (TTD), and higher cluster utilization, as compared to *Hadar* (described in Section VI-C).

Our extensive experiments on an AWS (or our lab testbed) cluster exhibit that *Hadar* achieves a 20% (or 21%) increase in cluster resource utilization and the speedup of 1.17× (or 1.16×) in terms of the total time duration across seven different workload mixes with 1 to 12 jobs. *HadarE* enjoys the further performance gains of 30% (or 34%) in cluster resource utilization, 90% (or 124%) in the mean job completion time, and more than 50% (or 80%) in the total time duration on an AWS (or our testbed) cluster. Besides its training acceleration, *HadarE* is demonstrated also to train DL models with better inference quality than *Hadar*. Overall, the main contributions of this work are as follows

- An efficient scheduler for DL training jobs in GPU is proposed to address the performance heterogeneity of multi-type accelerators at task-level granularity, arriving at *Hadar*.
- An optimization algorithm is developed following the primal-dual framework which employs a dual subroutine to analyze and tackle the scheduling problem on multiple heterogeneous cluster nodes.
- We prove the polynomial runtime complexity of our algorithm and also perform a detailed analysis to provide a long-term performance guarantee that approximates the optimal solutions within proven constant bounds.
- Each model training job is forked into multiple copies for possible concurrent executions on separate heterogeneous DL cluster nodes, to further enhance cluster resource utilization, realizing *HadarE* (*Hadar Enhancement*).
- Extensive experiments are conducted on two physical DL clusters: one leased from the AWS Cloud and the other available at our research lab, with results consistently demonstrating the solid advantages of *HadarE* over *Hadar* and Gavel.

TABLE I
COMPARISON OF VARIOUS SCHEDULERS

Schedulers	Tiresias	Gandiva	Gavel	Hadar	HadarE
Resource heterogeneity	✗	✓	✓	✓	✓
Scheduling heterogeneity	✗	✗	✗	✓	✓
Task-level heterogeneity	✗	✗	✗	✗	✓

II. Related Work and Motivation

Under prevailing data parallel training, a deep learning (DL) job is typically trained on multiple devices (i.e., GPUs or other accelerators) to process voluminous input data iteratively. Model training job is conducted on the machines of DL clusters, often employing a stochastic gradient descent (SGD) mechanism to keep improving the model's learnable parameters. The training data is fragmented into chunks (i.e., mini-batches) for training acceleration under SGD. A complete pass through the whole training data, with one chunk at a time, is referred to as an epoch [11]. A DL training job usually involves many epochs, and it stops after finishing a pre-specified number of epochs (say, 60) or finding the model prediction loss stabilized

for a given number of consecutive epochs, known as early stopping.

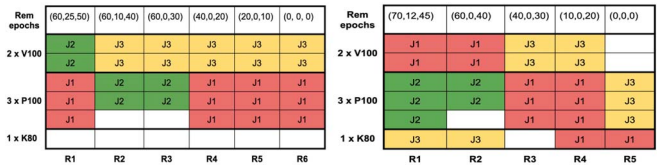
To accommodate multiple DL training jobs, traditional CPU-based cluster schedulers fall short since they fail to consider the unique characteristics of distributed DNN training. Recent production-scale workload studies [12], [13] confirm that the characteristics of the temporal (job runtimes) and spatial (requested resources) patterns exist in a DL cluster. In addition, the heavy-tailed nature of requested resources, along with the wide range of queuing delay and job runtime, has increasingly drawn research attention to DL cluster scheduling (exemplified by the pursuits at [4], [8], [14]). Those pursuits focus on designing customized cluster schedulers for DNN training jobs to enhance job performance and resource utilization. Still, they fail to address the adverse impact of resource heterogeneity effectively.

Some recent schedulers [10], [15], [16], [17] have considered device and model workload heterogeneity to some extent, but not at a fine granularity. Gandiva$_{fair}$ [15] focuses merely on fairness among jobs, while Hydra [16] aims at meeting deadlines. Gavel [10] accounts for the heterogeneity of both DNN training workloads and hardware devices when allocating resources to jobs. It presents a general optimization framework to characterize several scheduling policies. Lately, studies have treated both CPUs and accelerators as prime factors upon assigning workloads in a DL cluster [18], [19], [20], [21], [22], [23]. For cloud computing, Ghorbani et al. [19] presented a fractal model to capture the complex dynamics of the computing workloads for reducing energy consumption without affecting network performance. Recently, graphs have been prevalently used for assigning ML jobs to different devices [24], [25]. Multiple variants of graphs, such as Directed Acyclic Graphs (DAGs) [24] and Graph Neural Networks (GNN) [25], have been applied to obtain the optimal assignment of ML jobs across considered devices. However, the high computational cost of graph structures limits their effectiveness on task-level resource allocation at fine granularity. A recent study, called shockwave [26], predicts resource utility amid dynamic adaptation. Being model-agnostic, PPS [27] treats the DL training job as a closed box and predicts expected resource usage, considering only job statistics.

Unlike all prior schedulers, our *Hadar* (or *HadarE*) is aware of task-level (or job-level) performance heterogeneity. *Hadar* and *HadarE* allocate resources at finer granularity across an additional temporal dimension, yielding marked overall performance improvement.

A. Motivational Example

A cluster with two V100, three P100, and one K80 GPUs is considered. Three jobs arrive at the beginning to be scheduled. Job 1 (J1) requests 3 GPUs and requires 80 epochs to complete its training. Job 2 (J2) requests 2 GPUs and has a total of 30 epochs. Job 3 (J3) requires 2 GPUs for 50 epochs. The training speedups of those jobs on GPUs of different types, expressed as a matrix X, and the optimal allocation matrix Y^{Gavel}, under the assumption that the cluster has sufficient capacity, are expressed as



(a) Round-based simulation of Gavel (b) Round-based simulation of *Hadar*

Fig. 1. Comparative simulation results of scheduling three jobs in a cluster with 2× V100, 3× P100, and 1× K80 GPUs under Gavel [10] and *Hadar* [1].

$$X = \begin{pmatrix} & \text{V100} & \text{P100} & \text{K80} \\ \text{J1} & 40 & 20 & 30 \\ \text{J2} & 5 & 15 & 5 \\ \text{J3} & 10 & 2 & 20 \end{pmatrix}$$

$$Y_{\text{Gavel}} = \begin{pmatrix} & \text{V100} & \text{P100} & \text{K80} \\ \text{J1} & 0.6 & 0.4 & 0.0 \\ \text{J2} & 0.2 & 0.6 & 0.2 \\ \text{J3} & 0.2 & 0.0 & 0.8 \end{pmatrix}$$

Each element of this matrix represents the proportion of time that a job should run on a specific type of device. To achieve a near-optimal allocation, Gavel uses a priority matrix to schedule jobs on GPUs. The priority of a specific job on a particular type of GPUs is defined as the corresponding element of Y^{Gavel} divided by the number of rounds received (*i.e.*, resource allocation received). Fig. 1(a) illustrates the scheduling outcome over 6 rounds according to Gavel, where the first row represents the number of remaining epochs (i.e., Rem epochs) for each of the three jobs at a particular round. For example, in round 1 (R1 in the figure), J1, J2, and J3 have 60, 25, and 50 epochs to complete, respectively, represented by (60, 25, 50) in Fig. 1(a).

Each job under Gavel is assigned to a given number of GPUs throughout its execution rounds (i.e., two GPUs for J2, as depicted in Fig. 1(a)), and hence no job under Gavel can run on different numbers of GPUs during its execution. In contrast, we exploit the flexibility of task-level allocation to maximize the overall performance of the cluster. As shown in Fig. 1(b), *Hadar* strategically assigns the tasks of job $J1$ to all GPUs (i.e., two $V100$ GPUs, three $P100$ GPUs, and one $K80$ GPU), tasks of job $J2$ to three $P100$ GPUs, and tasks of $J3$ to all GPUs during the whole process. In comparison, Gavel's policy adopts the homogeneous allocation of tasks strictly, causing jobs $J1$, $J2$, and $J3$ to achieve lower cluster resource utilization (CRU) in the long run. For example, the CRU values of *Hadar* (or Gavel) in the first two rounds approach 100% (or 83%) and 83% (or 67%), respectively. Over the entire process of scheduling the three jobs, *Hadar* achieves its CRU of some 87%, versus $\sim$ 78% under Gavel, besides shortening the total training time by one round, as depicted in Fig. 1.

III. Design of *Hadar*

Our overall objective is to design an effective scheduler for distributed DL training jobs with the awareness of resource

heterogeneity and best performance. In this section, we present the theoretical foundation of our scheduler design.

A. System Model and Problem Formulation

Consider a cluster of machines equipped with different accelerator devices. A machine h has a capacity of c_h^r for type-r device. In a slotted time spectrum $(1, 2, \cdots, T)$, a DL job j arrives at time $a_j \in [T]$, requesting a number of worker devices W_j for model training. The device heterogeneity impacts the job training throughput, denoted by X_j^r which represents the number of iterations per second by job j on type-r accelerator. $E_j N_j$ denotes the total number of iterations to complete job j, where E_j refers to the total number of epochs and N_j is the total number of data chunks to be processed in each epoch of job j.

Upon arrival, the job joins a global queue managed by the scheduler, waiting to be assigned to available machine(s) for execution in subsequent time slots. The scheduler makes scheduling decisions, $w_{jh}^r(t)$, representing the number of type-r devices at machine h assigned to job j in time slot t. Let f_j denote the finish time of job j, and thus the job completion time can be expressed as $f_j - a_j$. We start with finding the optimal resource allocation and scheduling to maximize the overall utility across all jobs. The utility of a job j could be characterized by a general non-negative function $\mathcal{U}_j(\cdot)$ which is non-increasing with its completion time. The effective throughput [10], defined as the averaged number of iterations completed per second over the lifetime, can be a special case of the job utility, expressed as $E_j N_j$ divided by j's completion time. Given these notations, we can formulate the following optimization problem, P1:

$$\max \quad \sum_j \mathcal{U}_j(f_j - a_j) \tag{1}$$

$$s.t. \quad \sum_t x_j(t) \sum_r \sum_h w_{jh}^r(t) L \geq E_j N_j, \quad \forall j \tag{1a}$$

$$x_j(t) = \min\{X_j^r \,|\, \sum_h w_{jh}^r(t) > 0\}, \quad \forall j, \ \forall t \tag{1b}$$

$$f_j = \max\{t \in [T] \,|\, \sum_h \sum_r w_{jh}^r(t) > 0\}, \quad \forall j \tag{1c}$$

$$0 \leq \sum_j w_{jh}^r(t) \leq c_h^r, \quad \forall h, \ \forall r, \ \forall t \tag{1d}$$

$$\sum_h \sum_r w_{jh}^r(t) \in \{0, W_j\}, \quad \forall j, \ \forall t \geq a_j$$

$$w_{jh}^r(t) = 0, \quad \forall j, \ \forall h, \ \forall t < a_j. \tag{1e}$$

Constraints (1a) and (1b) regulate that the total number of iterations accomplished across time is no smaller than $E_j N_j$ to complete job j. Specifically, L is the length of a time slot, $x_j(t)$ expresses the bottleneck throughput across tasks, *i.e.*, the number of iterations per second at the slowest device, due to the parameter synchronization barrier. Constraint (1c) by definition, represents the last time slot when a job receives non-zero allocation to run. Constraint (1d) indicates resource capacity limits at each machine, while (1e) regulates resource requirements for each job, *i.e.*, the All-or-Nothing property (Gang scheduling), following the conventional practice [12]. A brief notation summary is presented in Table II.

TABLE II
NOTATION

J	# of jobs
R	# of GPU types
a_j	arrival time of job j
f_j	finish time of job j
W_j	# of GPUs requested by job j
E_j	# of total training epochs specified by job j
N_j	# of data chunks (iterations) per epoch in job j
c_h^r	# of type-r GPUs on machine h
X_j^r	# of training iterations per *sec* for job j on type-r GPU
$w_{jh}^r(t)$	# of type-r GPUs on machine h allocated to job j at time t
$\mathcal{U}_j(.)$	utility of job j

B. Problem Solving Based on Primal-Dual

The optimization problem P1 is difficult to solve since it involves integer variables and non-conventional constraints (1b), (1c). To address these challenges, we first reformulate Problem P1 into the following integer linear program (ILP). Suppose $\mathbb{S}_j$ is the set of feasible schedule for job j which corresponds to the set of decisions $(w_{jh}^r(t), \forall h \in [H], j \in [J], t \in [T])$. It satisfies constraints (1a), (1b), and (1e). Due to the combinatorial nature of these constraints, there is an exponential number of feasible schedules for each job. For a schedule $s \in \mathbb{S}_j$, the decision variable in the ILP is a binary variable y_{js} which indicates whether the job is admitted to the cluster under schedule s. With schedule s, job j's finish time is denoted as f_{js}, and its allocation $w_{jh}^{rs}(t)$ represents the number of type-r workers in server h at time t. Thus, P1 can be reformulated to P2 as follows:

$$\max \quad \sum_j \sum_s y_{js} \mathcal{U}_j(f_{js} - a_j) \tag{2}$$

$$s.t. \quad \sum_j \sum_{s:t\in s, h\in(t,s)} w_{jh}^{rs}(t) y_{js} \leq c_h^r, \forall h, \forall r, \forall t \tag{2a}$$

$$\sum_s y_{js} \leq 1, \forall j \tag{2b}$$

$$y_{js} \in \{0, 1\}, \forall j, \forall s. \tag{2c}$$

We use $t \in s, h \in (t, s)$ to indicate that schedule s uses server h to deploy a worker for job j in time t. Eq. (2) and constraint (2a) are equivalent to Eq. (1) and constraint (1d), respectively. Constraints (2b)-(2c) are equivalent to constraints (1a)-(1c) and (1e). We can easily check that P1 and P2 are equivalent, since a feasible solution to one has a corresponding feasible solution to the other, with the same objective values.

After sidestepping non-conventional constraints, we next solve Problem P2 based on the primal-dual framework [28], by relaxing its integer constraints (2c) and formulating its dual problem designated as P3 below:

$$\min \quad \sum_j \mu_j + \sum_t \sum_h \sum_r k_h^r(t) c_h^r(t) \tag{3}$$

$$s.t. \quad \mu_j \geq \mathcal{U}_j(f_{js} - a_j) - \sum_{t\in s} \sum_{h\in(t,s)} \sum_r k_h^r(t) w_{jh}^{rs}(t), \forall j, \forall s$$

$$k_h^r(t) \geq 0, \forall h, \forall r, \forall j, \forall t, \qquad \mu_j \geq 0, \forall j. \tag{3a}$$

In this problem, $k_h^r(t)$ and μ_j are the dual variables associated with constraints (2a) and (2b). $k_h^r(t)$ can be interpreted as the unit cost for type-r accelerators on server h at time t. Thus, the right-hand side of (3a) is the job utility minus the overall resource cost for job j with schedule s at time t, which indicates the payoff of the job. Let $\phi_j(s)$ denote this term, *i.e.*, $\phi_j(s) = \mathcal{U}_j(f_{js} - a_j) - \sum_{t\in s}\sum_{h\in(t,s)}\sum_r k_h^r(t) w_{jh}^{rs}(t)$. To minimize the dual objective, μ_j^* should be expressed as $\mu_j^* = \max\{0, \max_{s\in\mathbb{S}_j}\phi_j(s)\}$, based on its constraints. The corresponding best schedule s^* can be written as

$$s^* = argmax_{s\in\mathbb{S}_j}\phi_j(s). \tag{4}$$

To solve Eq. (4), we design an efficient subroutine to be elaborated later (Algorithm 2). With respect to $k_h^r(t)$, based on its resource price interpretation, we hope to compute its value to ensure that a high-utility job gets a positive payoff (if the resource demand can be satisfied) and a job with a low utility or without available resources gets a non-positive payoff. Let $\gamma_h^r(t)$ denote the number of type-r accelerators allocated on server h at time slot t. The dual price resource is designed to be dynamically updated using the following price function:

$$k_h^r(\gamma_h^r(t)) = U_{min}^r\left(\frac{U_{max}^r}{U_{min}^r}\right)^{\frac{\gamma_h^r(t)}{c_h^r}}, \tag{5}$$

$$with \quad U_{max}^r = \max_j \frac{\mathcal{U}_j(t_j^{min} - a_j)}{w_j^r}, \quad \forall r \tag{6}$$

$$U_{min}^r = \frac{1}{4\eta}\min_j \frac{\mathcal{U}_j(T - a_j)}{t_j^{max}\sum\limits_{r\in[R]} w_j^r}, \quad \forall r$$

$$t_j^{min} = \frac{N_j E_j}{M_j \max_r(X_j^r)}, \quad t_j^{max} = \frac{N_j E_j}{M_j \min_r(X_j^r)}, \tag{7}$$

where U_{max}^r and U_{min}^r imply the maximum and the minimum per-unit-resource job utility values for type-r accelerators to execute tasks among all jobs. $\mathcal{U}_j(T - a_j)$ is the smallest utility that job j may achieve, when it ends at T. η is the scaling factor to bound the initial value of the dual objective.

The intuition is stated as follows. The price starts to be low enough to accept the incoming job: when $\gamma_h^r = 0$, we have $k_h^r(t) = U_{min}^r$, lowest to admit any job. The price increases exponentially with the growing amount of allocated accelerators, so as to filter out low-utility jobs. When a server is out of free resources, $\gamma_h^r(t) = c_h^r$, reaching the price $k_h^r(t) = U_{max}^r$, high enough to block other jobs from getting these resources. Such a price function is crucial to guarantee a good competitive ratio for our effective algorithm, to be presented in the next section. U_{max}^r and U_{min}^r are calculated based on the cluster's workload in the algorithm.

C. Algorithm Design

Based on the resource price and job payoff interpretations, we next present our algorithm (Algorithm 1), which generates optimal scheduling decisions for the jobs in the queue in each round-based scheduling event (`Line 5`). Specifically, a greedy algorithm and a dynamic programming approach are presented in Algorithm 2, to calculate s^* in Eq. (4) by solving the following equivalent form:

Algorithm 1 *Hadar* Scheduling

Input: $c_h^r, \forall h \in [H], r \in [R]$
1: **Initialize:** $w_{jh}^r(t)(t) = 0, \gamma_h^r(t) = 0, k_h^r(t) = k_h^r(0) \forall j \in [J], t \in [T], h \in [H]$
2: **while** true **do**
3: Upon the arrival of each job, admit it to the queue Q
4: In each round t:
5: $\{Q_s, c_h^r, \{w_{jh}^r(t)(t)\}\} = DP\_allocation(0, Q, c_h^r, null, \gamma_h^r(t), k_h^r(t))$
6: **for** job $j \in [Q_s]$ **do**
7: Run job j until round $t+1$ according to $(\{w_{jh}^r(t)\})$
8: **end for**
9: If j is complete, remove it from Q
10: **end while**

$$\begin{aligned} \max \quad & \mathcal{U}_j(f_j - a_j) - \sum_t\sum_h\sum_r k_h^r(t) w_{jh}^r(t) \\ s.t. \quad & \gamma_h^r(t) + w_{jh}^r(t) \le c_h^r, \quad \forall j \ in \ queue, \forall r, \forall h, \forall t \\ & \text{Constraints } (1a - 1e). \end{aligned} \tag{8}$$

If we fix f_j, the optimization objective can be further transformed to $\min\sum_h\sum_r k_h^r(t) w_{jh}^r(t)$, which can be interpreted as minimizing a cost function at each round.

In Algorithm 2, waiting jobs in the current round are in queue Q. According to the recursive dynamic programming solution in each state, there are two possible choices for a certain job, either calculating the cost and allocation by selecting the job for scheduling or proceeding without selecting the job in `Lines 14-15`. The set of jobs and the allocations with minimum cost is returned from the DP function call `Lines 16-21`. Note that we always save the result if $cost_Q$ and $cost_{Q/j}$ are compared for different subsets of jobs to avoid recomputing the same subproblem in later recursive function call. The `FIND_ALLOC` function selects the best possible allocation within the current state of the server ($srvr$). Initially, the server's state is sorted according to the descending order of throughput (iterations per second) on each GPU type for the job (`Line 23`). The algorithm produces the allocations on different settings - by consolidating tasks of the job in the minimum possible server (`Line 24`) and allocating the tasks of the job in different servers (`Line 25`). The costs are calculated using the cost function aforementioned. For non-consolidated setting, communication cost (the cost of bandwidth utilization while communicating among different servers) is also added (`Lines 26-27`). The allocation with the minimum cost is calculated, and μ_j is calculated to determine allocation's feasibility (`Lines 28-32`). According to the selected allocation, allocated resource $\gamma_h^{rc}(t)$, price function $k_h^{rc}(t)$, and the server state are updated (`Lines 10-12`).

Algorithm 2 DP_allocation($idx, Q, srvr, \{w^r_{jh}(t)\}, \gamma^r_h(t), k^r_h(t), \forall h, \forall r$)

if $(index >= Q.length())||is\_Server\_Full(srvr)$ **then**
  return $Q, \{w^r_{jh}(t)\}, srvr$
end if
$job = Q[idx]$
$\{w\_prev^r_{jh}\} \leftarrow \{w^r_{jh}(t)\}$
$\{w\_job^r_{jh}\}$ = **FIND_ALLOC**$(job, srvr)$
if $\{w\_job^r_{jh}\}$ = $null$ **then**
  return $Q, \{w^r_{jh}(t)\}, srvr$
end if
$\gamma^{rc}_h(t)$ = $\gamma^r_h(t) + w\_job^r_{jh}, \forall h, \forall r$
$k^{rc}_h(t) \leftarrow$ Update $k^r_h(t)$ according to Eq. (5), $\forall h, \forall r$
$srvr^c \leftarrow$ Update $srvr$ according to $\{w\_job^r_{jh}\}$
$\{w^r_{jh}(t)\}.append(w\_job^r_{jh}), \forall h, \forall r$
$(Q, \{w^r_{jh}(t)\}, srvr^c) = DP\_allocation$
$((idx+1), Q, srvr^c, \{w^r_{jh}(t)\}, \gamma^{rc}_h(t), k^{rc}_h(t)), \forall h, \forall r$
$(Q, \{w^{cr}_{jh}(t)\}, srvr) = DP\_allocation$
$((idx+1), Q, srvr, \{w\_prev^r_{jh}\}, \gamma^r_h(t), k^r_h(t)), \forall h, \forall r$
$cost_Q += \sum_h \sum_r k^{rc}_h(t)\ w^r_{jh}(t)$
$cost_{Q/j} += \sum_h \sum_r k^r_h(t)\ w^{cr}_{jh}(t)$
if $cost_Q < cost_{Q/j}$ **then**
  return $Q, \{w^r_{jh}(t)\}, srvr^c$
end if
return $Q, \{w^{cr}_{jh}(t)\}, srvr$
procedure FIND_ALLOC($job, srvr$):
  $srvr \leftarrow$ sort GPU type according to the descending order of $x^r_j,\ \forall h \in H$
  $\{all\_alloc_{packed}\} \leftarrow$ find allocations considering consolidated setting
  $\{all\_alloc_{!packed}\} \leftarrow$ find allocations without consideration of consolidated setting
  $\{cost_{packed}\} \leftarrow \sum_h \sum_r k^r_h(t) w^r_{jh}(t), \forall all\_alloc_{packed}$
  $\{cost_{!packed}\} \leftarrow \sum_h \sum_r k^r_h(t) w^r_{jh}(t)$
  $+ comm.\ cost, \forall all\_alloc_{!packed}$
  $alloc \leftarrow$ allocation corresponding to $\min(\{cost_{packed}\}, \{cost_{!packed}\})$
  $\mu_j = \mathcal{U}_j(f_{js} - a_j) - \min(\{cost_{packed}\}, \{cost_{!packed}\})$
  if $\mu_j > 0$ **then**
    return $alloc$
  end if
  return $null$
end procedure

D. Theoretical Analysis

Theorem 1 (Runtime Complexity): Algorithm 2 can make scheduling decisions in polynomial time for a set of jobs in an execution round.

Proof: The function FIND_ALLOC($job, srvr$) has a time complexity of $\mathcal{O}(R(H \log H))$ to sort the servers based on job throughput on GPUs of different types. This sorting calculation is only done once during the lifespan of a job in the system. For calculating allocation in both consolidated (all_alloc_{packed}) and non-consolidated ($all\_alloc_{!packed}$) settings, all servers need to be iterated for each GPU type, resulting in a complexity of $\mathcal{O}(HR)$. In our dynamic programming (DP) algorithm, we have two states: job ID and the current server state. We need to calculate $n(Q)HR$ combinations or function calls, with a time complexity of $\mathcal{O}(HR)$ for each call. It should be noted that we pre-calculate and save $cost(DP\_allocation(jobs, srvr))$ for all $j \in Q$. Therefore, the time complexity of the DP is $\mathcal{O}(n(Q)(HR)^2 + R(H \log H))$.

Theorem 2 (Competitive Ratio): *Hadar* is 2α competitive, where $\alpha = \max_{r\in[R]}(1, ln\frac{U^r_{max}}{U^r_{min}})$ and U^r_{max}, U^r_{min} are defined in Eqs. (6), (7).

Proof: We define OPT as the optimal objective value of Problem P1. P_j and D_j represent the objective values of the primal problem P2 and of the dual problem P3, respectively, returned by Algorithm 1 after deciding the schedule of job j. The initial values of Eqs. (2) and (3) are denoted by P_0 and D_0. Specifically, $P_0 = 0$ and $D_0 = \sum_t \sum_h \sum_r k^r_h(0) c^r_h(0)$. Finally, P_f and D_f represent the final primal and dual objective values returned by Algorithm 1. The theorem is proved based on the following definitions and lemmas taken from [29], to ensure that solutions so derived are bounded within 2α from actual optimality. ■

Lemma 1: If there exists a constant $\alpha \geq 1$ such that $P_j - P_{j-1} \geq \frac{1}{\alpha}(D_j - D_{j-1})$ for all jobs $j \in [J]$, and if $P_0 = 0$ and $D_0 \leq \frac{1}{2}OPT$, then Algorithm 1 is 2α-competitive in total job utility.

Definition 1: The allocation-cost relationship for Algorithm 1 with $\alpha \geq 1$ is: $k^{r,j-1}_h(t)(\gamma^{r,j}_h(t) - \gamma^{r,j-1}_h(t)) \geq \frac{c^r_h}{\alpha}(k^{r,j}_h(t) - k^{r,j-1}_h(t))$.

Lemma 2: If the allocation-cost relationship holds for $\alpha \geq 1$, then Algorithm 1 ensures $P_j - P_{j-1} \geq \frac{1}{\alpha}(D_j - D_{j-1}), \forall j$.

Definition 2: The differential allocation-cost relationship for Algorithm 1 with $\alpha^r_h \geq 1$ is: $k^r_h(t) d\gamma^r_h(t) \geq \frac{c^r_h}{\alpha^r_h} dk^r_h(t), \forall t, h, r$.

Lemma 3: $\alpha^r_h = ln\frac{U^r_{max}}{U^r_{min}}$ and the price function defined in Eq. (5) satisfies the differential allocation-cost relationship.

Based on Lemma 3, the marginal cost function employed in Algorithm 1 meets the condition of differential allocation-cost relationship with $\alpha = \max_{r\in R}(1, ln\frac{U^r_{max}}{U^r_{min}})$. As the resource demand of a job j is expected to be less than the capacity, we can infer that:

$$d\gamma^r_h(t) = \gamma^{r,j}_h(t) - \gamma^{r,j-1}_h(t)$$
$$dk^r_h(t) = k^{r'}_h(\gamma^r_h(t))(\gamma^{r,j}_h(t) - \gamma^{r,j-1}_h(t)) = k^{r,j}_h(t) - k^{r,j-1}_h(t).$$

The allocation-cost relationship in Definition 1 holds for $\alpha = \max_r(1, ln\frac{U^r_{max}}{U^r_{min}})$, which is implied by the differential allocation-cost relationship in Definition 2. Considering Algorithm 1, we note that $\frac{1}{\eta} \leq \frac{t^{max}_j \sum_{r\in[R]} w^r_j}{\sum_h \sum_r c^r_h}$, which implies that $\frac{\sum_h \sum_r c^r_h}{\eta} \leq t^{max}_j \sum_r w^r_j$ for all jobs j. This minimum resource consumption across all tasks of job j equals:

$$D_0 = \sum_t \sum_h \sum_r U^r_{min} c^r_h = \sum_t \sum_h \sum_r \frac{1}{4\eta} \min_{j,s} \frac{\mathcal{U}_j(f_{js} - a_j)}{t^{max}_j \sum_r w^r_j} c^r_h$$

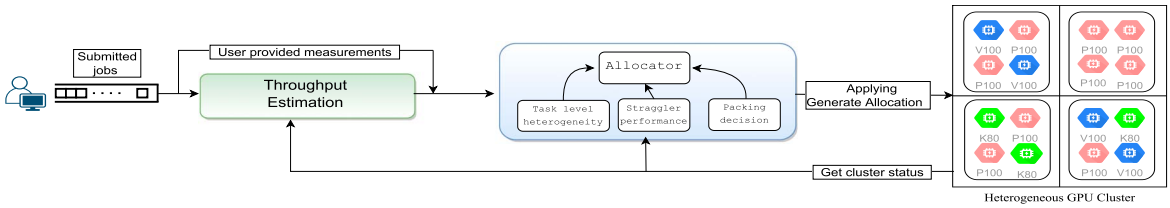


Fig. 2. The overview of *Hadar*, a fine-grained heterogeneity-aware scheduler for a GPU-based deep learning cluster.

$$= \frac{\sum_t \sum_h \sum_r c_h^r}{4\eta} \min_{j,s} \frac{\mathcal{U}_j(f_{js} - a_j)}{t_j^{max} \sum_r w_j^r}$$

$$\leq \frac{1}{4} t_j^{max} \sum_r w_j^r \min_{j,s} \frac{\mathcal{U}_j(f_{js} - a_j)}{t_j^{max} \sum_r w_j^r}, \forall j. \quad (9)$$

Selecting $(j, s) = arg\min_{j,s} \mathcal{U}_j(f_{js} - a_j)$ yields:

$$(9) \leq \frac{1}{4} t_j^{max} \sum_{r \in [R]} w_j^r \min_{j,s} \frac{\mathcal{U}_j(f_{js} - a_j)}{t_j^{max} \sum_r w_j^r}, \forall j$$

$$\leq \frac{1}{2} \mathcal{U}_j(f_{js} - a_j) \quad \leq \frac{1}{2} OPT.$$

The last inequality holds because we assume the offline optimal solution accepts at least one job, which is reasonable in the real-world cluster. Then we have $OPT \geq \min_{j,s} \mathcal{U}_j(f_{js} - a_j)$. According to Lemmas 1 and 2, we conclude the proof. ■

E. Hadar Overview

Guided by the theoretical investigation, our fine-grained heterogeneity-aware scheduler, *Hadar*, is illustrated in Fig. 2. Given a set of queued jobs, the scheduler dispatches all jobs onto different types of accelerators on different servers (i.e., machines or cluster nodes) towards maximizing the cluster-wide utility. Our scheduler takes the job's performance result (*i.e.*, iterations per second) on each accelerator type as its input. In particular, the throughput estimator in *Hadar* requires performance measurements for every runnable job on each available accelerator type, and such measurements can be provided either as input data (for trace-driven evaluation in Section IV) or by an estimation formula (to be detailed in Section VI). For a given input, the scheduling algorithm in the allocator calculates the number and types of GPUs assigned to each job on particular machines in a given round. It considers task-level heterogeneity and job packing decisions to maximize overall cluster utility.

IV. Trace-Driven Evaluation

Extensive trace-driven simulation is conducted by our discrete-time simulator using a real-world trace [9] to evaluate *Hadar* for comparison with its counterpart schedulers. Following the setup of the simulation experiments in Gavel [10], our simulated cluster consists of 15 nodes, which house 60 GPUs in total, with 20 GPUs each for V100, P100, and K80.

TABLE III
Evaluation Workloads: Model, Dataset, and Relative Size for Each Deep Learning Job

Training Job	Model	Dataset	Size
Image Classification	ResNet-50 [30]	ImageNet [31]	XL
Image Classification	ResNet-18 [30]	CIFAR-10 [32]	S
Language Modeling	LSTM [33]	Wikitext-2 [34]	L
Image-to-Image Translation	CycleGAN [35]	Monet2photo [35]	M
Language Translation	Transformer [36]	Multi30K [37] (de-en)	L

The workloads are based on a Microsoft trace [9], summarized in Table III and elaborated in the next paragraph. For each job (workload) in Table III, we leverage its throughput measurements from Gavel as our scheduling input to simulate the job events such as job arrival, completion, and preemption. The overhead of each checkpoint-restart is simulated by enforcing a 10-second delay when a job receives a new allocation. According to our evaluation, the duration of a scheduling round impacts the results of evaluation performance metrics, with the duration ranging from 6 minutes to 1.5 minutes to yield the best results, depending on workloads, available resources, and metrics of interest. The results in this section are obtained over a duration of 6 minutes.

A. Synthetic Workloads and Datasets

In our trace-driven evaluation, we randomly selected 480 jobs from the busiest hour range (hours 3-10) of the Microsoft trace [9]. The trace includes information such as the requested number of GPUs, submission time, and job duration, while details on model architectures and datasets are not provided. Therefore, we categorized the jobs based on their total GPU required time into four groups: Small (0-1 GPU-hours), Medium (1-10 GPU-hours), Large (10-50 GPU-hours), and XLarge (60-100 GPU-hours). For each training job in the trace, we uniformly sampled the job type from these categories and specified its model and dataset accordingly, as shown in Table III. In our evaluation, all jobs were available at the beginning of the trace.

B. Evaluation Results

We conducted an evaluation for comparing *Hadar* and its state-of-the-art DL cluster scheduler counterparts, Gavel [10] and Tiresias [4], as well as the default production-level cluster scheduler, Apache YARN's capacity scheduler (YARN-CS) [6].

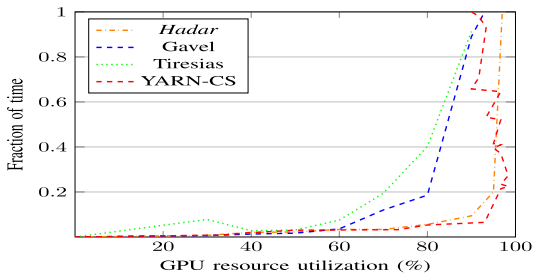


Fig. 3. Comparison of cluster-wide GPU resource utilization (GRU) among the four schedulers.

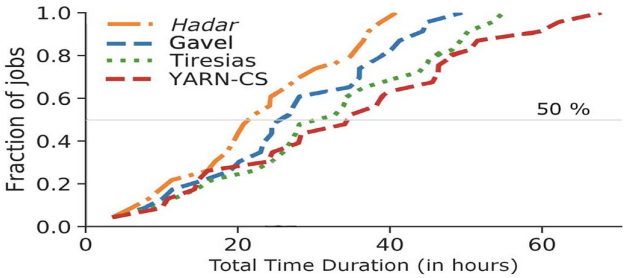


Fig. 4. Cumulative fractions of completed jobs over time, when scheduled by Gavel [10], Tiresias [4], YARN-CS [6], and *Hadar*, respectively.

While *Hadar* considers the task-level heterogeneity of DNN training jobs for scheduling decisions, Gavel only focuses on job-level heterogeneity, and Tiresias is heterogeneity-unaware among accelerators. For comparison, Tiresias is configured with two priority queues and its `PromoteKnob` disabled, whereas Gavel has its configuration similar to that under the previous experimental study [10]. The comparative performance metrics of interest include GPU resource utilization (GRU) and the total time duration (TTD), with scalability also compared, as detailed next.

GPU Resource Utilization. GPU resource utilization (GRU) refers to the percentage of the total job run-time during which GPUs are utilized. The comparative GRU results of four schedulers are shown in Fig. 3. The highest GRU is achieved by YARN-CS due to its non-preemptive nature. However, this comes at the cost of long total job completion duration, as to be seen in Fig. 4. Gavel, on the other hand, leaves heterogeneous GPUs unused even if the total number of them meets the requirement of a queued job. This results in lower GRU compared to that of YARN-CS. Tiresias also suffers from the same limitation as Gavel. In contrast, *Hadar* takes advantage of fine-grained scheduling and resource heterogeneity awareness to elevate its GRU. More specifically, *Hadar* can allocate tasks to GPUs that are most suited for them, possibly of different types when necessary, based on task characteristics and cluster resource availability. As a result, the number of GPUs left unused is minimized, leading to better GRU compared to those of Gavel and Tiresias. Moreover, *Hadar* exhibits similar GRU compared to YARN-CS, indicating that it can utilize GPUs effectively to lower job completion times.

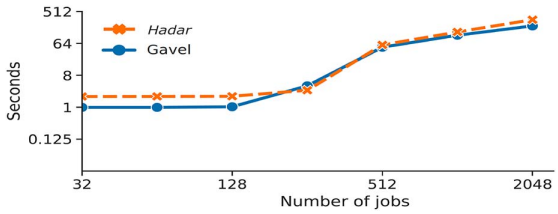


Fig. 5. Scalability comparison under *Hadar* and Gavel [10] versus active jobs in a heterogeneous cluster, whose size grows as the number of jobs increases.

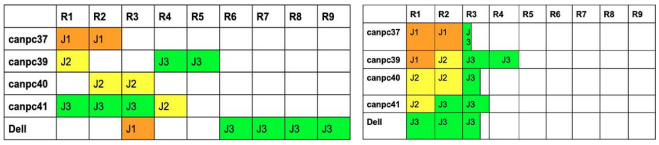
Total Time Duration. Cumulative fractions of completed jobs over time, when scheduled by Gavel [10], Tiresias [4], YARN-CS [6], and *Hadar*, are demonstrated in Fig. 4, where *Hadar* completes training all jobs in 40 hours, known as the total time duration (TTD). From the figure, *Hadar* is observed to outperform its counterparts, whose TTDs equal 68 hours by 1.67× compared to YARN-CS, and by 1.35× and 1.21× against Tiresias and Gavel, respectively. Additionally, the median time duration to complete 50% jobs (marked by the horizontal gray line in Fig. 4) under *Hadar* is 1.20× (or 1.40×) shorter than that under Gavel (or under Tiresias). Clearly, *Hadar* outperforms all its counterparts due to better resource utilization.

Scalability. The time (in seconds) taken by *Hadar* and by Gavel to generate their decisions versus the number of jobs are depicted in Fig. 5. When the job count increases from 32 to 2048, *Hadar* and Gavel are observed to have similar scaling performance in terms of the scheduling time.

Even under heavy workloads (say, with some 2000 jobs), *Hadar* can schedule, and allocate resources to, jobs in less than 7 minutes per scheduling round. Our scheduler achieves resource allocation updating efficiently by dealing with solely newly incoming jobs in each scheduling round, if no preemption exists. Rather than recomputing the allocations of all jobs in every scheduling round, *Hadar* just allocates resources to those newly incoming jobs progressively, without changing the allocated resources of running jobs present in the cluster. If the allocation of a running job is altered due to preemption, the affected job will get a new resource allocation, following a checkpoint/restart. We observe that only 30% of scheduling rounds require changes to job resource allocations on average.

V. Resource Utilization Enhancement

While *Hadar* is heterogeneous-aware to schedule DL training jobs run on a cluster across both spatial and temporal dimensions, each job is scheduled to run on at most one cluster node (i.e., machine or server) over training rounds. Hence, the cluster resource can often be under-utilized because a job cannot be executed concurrently on two or more nodes, even when some nodes are idle and available in any scheduling round. It is due to the fact that there is just one single copy of each job under training throughout all scheduling rounds, limiting *Hadar* to allocate just one node at any job. As can be seen in Fig. 6(a), *Hadar* schedules three training DL jobs (J1, J2, and J3) to run on just three of the five nodes constituting the cluster testbed available in our lab, leaving two nodes idle for the first three



(a) Under *Hadar* (b) Under *HadarE*

Fig. 6. Comparative illustration of scheduling rounds under (a) *Hadar* and (b) *HadarE*.

rounds, from R1 to R3. Since J1 completes its training in R3, three nodes become idle during R4, when only two training jobs are in progress. The last four rounds involve just one node (Dell) to continue training J3 until completion. Our solution to overcoming this limitation is to fork a job to multiple copies, which may then be scheduled onto separate nodes available for training the job concurrently, yielding *Hadar* Enhancement (or *HadarE* for short). This way enhances resource utilization by avoiding cluster nodes to stay idle in any scheduling round, when there is a training job copy left to be completed. If a job is forked to f copies, the job may be executed on up to f nodes, if available. When the training times of jobs in a batch are known *a priori*, it is desirable to fork these long-running jobs, each up to n copies, for the best resource utilization under *HadarE*. If each training job is forked to five copies when run on our 5-node testbed to keep all cluster nodes busy in all rounds but possibly the last one, as illustrated in Fig. 6(b), highest resource utilization is achieved to yield the shortest TTD (total time duration).

Each training job in a workload batch under *HadarE* is forked to n copies for execution on an n-node cluster, permitting multiple copies of a given job to run concurrently on up to n nodes, if available. Two issues are involved in realizing *HadarE* for the best performance: scheduling the copies of training jobs (plus initial throughput estimation) and aggregating and consolidating the training copies of each job.

A. Scheduling Copies of Training Jobs

Each training job starts with forking it into a proper number of copies, say n, for an n-node DN cluster. A Job Tracker is designed to track the progress of forked copies of all jobs, responsible for training copy aggregation and model parameter consolidation during the course of training, as shown in Fig. 7. All forked copies of a job are registered with Job Tracker, utilizing their unique job-IDs. Each job-ID is produced by job_ID $=$ max_job_count $\times\, i +$ parent_job_id, where max_job_count is the maximum number of jobs expected to co-exist in the n-node cluster and i ranges from 1 to the number of forked copies for the job, typically being n to maximize resource utilization. With their unique job_IDs obtained from Job Tracker, all copies of the job are sent to *Hadar* for scheduling to run on cluster nodes (see Fig. 7).

As demonstrated in Fig. 6(b), *HadarE* schedules the three jobs, with each of them forked to five copies (by Job Forker shown in Fig. 7) and registered with Job Tracker, for a total of 15 jobs to maximize resource utilization so that no node is left idle throughout the first two rounds. In the 3rd round, all nodes are assigned to train J3 (the longest running job) for various dispatched epoch counts, and in the last round, R4, one single node, canpc39, is assigned to finish up the remaining epochs of J3. In this figure, four nodes, besides canpc41, complete their dispatched numbers of epochs of J3 at various time points before the scheduled R3 time slot expires; the last round involves solely canpc39, which completed its previously assigned epochs first, thus best suited for handling J3's remaining epochs in R4. In general, the last round may involve β nodes (for $1 \leq \beta \leq n$) to concurrently finish up the remaining epochs, with participating nodes being those β fastest nodes that complete their assigned epochs in the immediately earlier round. The job copy(s) run on the very last round for a batch of training jobs is (are) likely to end ahead of its (theirs) scheduled time slot, exhibiting early finish of those training jobs. Clearly, *HadarE* enjoys higher performance than *Hadar*, boosting resource utilization to shorten the duration of training a batch of DL models and thus to lower the mean job completion time (JCT).

Relying on *Hadar*, *HadarE* schedules jobs in a round-based manner with a fixed time slot per round. During the time slot, each scheduled node trains its assigned job for the specified number of epochs. The node may complete the specified number of epochs before the time slot expires; in this case, the node waits to get its next allocated job at the beginning of the next round. On the other hand, the node may fail to complete the specified number of epochs when the time slot ends; in this case, the node informs Job Tracker of the number of epochs it has completed, since this information is needed in scheduling the next round. Hence, communications take place between nodes and the tracker at each round after they complete their assigned epochs or the time slot expires, for the tracker to (1) calculate the total number of epochs completed by all allocated nodes for every job, (2) aggregate all copies of every job, and (3) consolidate the model parameters of every job obtained by its concurrently executed copies, where (2) and (3) are elaborated in subsection V-B. As necessity, *HadarE* augments the *Hadar* Scheduler with initial throughput estimation (see Fig. 7), outlined below.

Initial Throughput Estimation. Following the primal-dual approach outlined in Section III-B for performance improvement, *Hadar* depends on the measured speedups (i.e., throughputs) of each model when trained on all cluster nodes (which are characterized mainly by their equipped GPUs and associated on-board Performance-Memory Index (PMI), PCIe communication capacities, etc.). Such throughput information can be either obtained by model execution profiling or given as the scheduling input. However, model execution profiling, even done on the fly, tends to take many rounds (so that every model has the chance to be assigned for execution on each node) before sound overall throughput information can be obtained. On the other hand, throughput information given as the input cannot be inclusive, so that any cluster containing nodes with different GPU and/or PCIe components absent in given throughput information would involve profiling in early rounds after scheduling starts. In both situations, unsatisfactory allocations happen from

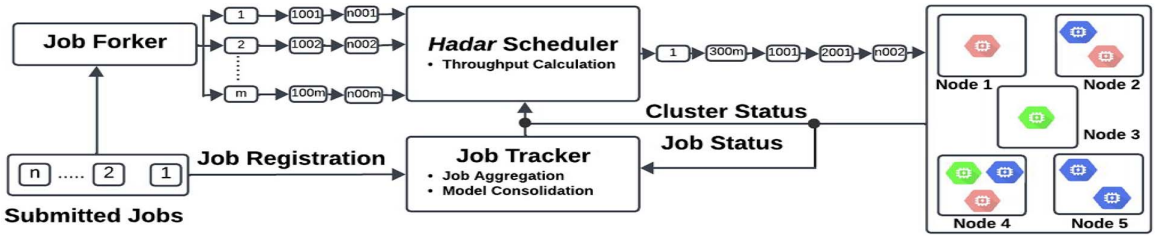


Fig. 7. An overview of *Hadar* Enhancement, *HadarE*.

the beginning until the throughput of every model executed on each node is all available. To address this shortcoming, we have derived an expression for initial throughput estimation, after numerous experiments on the heterogeneous GPU testbed in our lab, as follows:

$$\text{Throughput} = \frac{PMI \times \text{batch\_size} \times \text{pcie\_scaling}}{\text{model\_weight} \times \text{dataset\_size}}, \quad (10)$$

whose rationale is highlighted next.

Since DL model training is often computation- and memory-intensive, best performed on GPUs (with tensor cores), PMI (Performance-Memory Index) in the expression denotes the ratio of GPU's parallel processing ability aided by tensor cores, in terms of teraflops (tera floating point operations) per second and the GPU's VRAM capacity in a square root fashion. In the course of training on GPUs, the host machines of involved GPUs require frequent and heavy communications from host's DRAM and GPU's on-board VRAM through their Peripheral Component Interconnect Express (PCIe) channels, making PCIe capacities critical to GPU throughputs. The expression thus includes the term of pcie_scaling, which signifies different PCIe versions integrated on the machine's motherboard. Additionally, the mini-batch size during training on a GPU affects its throughput, with a larger size calling for less communication to net a larger throughput. On the other hand, a more complicated model or a larger dataset used for training on a GPU lowers its throughput, with model complexity approximated by a weight scale, from small, modest, high, to extra high; so is the dataset size to scale from S, M, L, to XL (see Table III).

Note that the aforementioned expression provides a reasonable estimate for *HadarE* to yield good scheduling decisions from the beginning. The quality of throughput information is improved progressively in the course of training, since every scheduled round let each involved node notify the Job Tracker of its actual throughput under its allocated job. As a result, the accurate throughputs of every model on all nodes are progressively available.

B. Aggregating and Consolidating Training Copies

Unlike *Hadar* and earlier schedulers (Gavel, Tirsias, YARN-CS, etc.) where every training job is run on one node at a time throughout all scheduling rounds, *HadarE* lets each training job be executed on multiple (available) nodes concurrently for resource utilization enhancement and thus training time reduction. It naturally needs to aggregate and consolidate results of copies of every job (i.e., DL model) trained on different nodes after each scheduled round to ensure the quality of models after training finishes, so that they are similar to, or even better than, what would have been obtained when trained without job forking. Both result aggregation and result consolidation are conducted by Job Tracker at the end of each scheduling round. Result aggregation simply sums completed training steps up, where one training step for a model means to train the model via ϕ mini-batches of training data, with $\phi =$ (training data size)/(mini-batch size). Note that in practice, model training progress is tracked at the step level, instead of the epoch level, especially when the scheduling time slot is short (i.e., a few minutes). When the total number of completed training steps of a job reaches the specified threshold, equal to $\phi \times$ (the training epoch count), all copies of the job in the scheduler are then discarded to avoid rescheduling of tasks of the completed job.

Upon completing its assigned training steps or ending the time slot, a node notifies Job Tracker of the number of steps it completed and the trained model parameters of its scheduled job. The total number of steps for the job completed by all nodes involved in training copies of the job is added (i.e., aggregated) by Job Tracker, with the job's model parameters consolidated by weight-averaging those of training copies, before passing them to the scheduler (*Hadar*, see Fig. 7), which then makes allocations for the next round. Given the number of training steps of a job left at the start of a scheduling round, *HadarE* for an n-node cluster divides that number into n portions according to their respective throughput values (that reflect nodes' training capabilities), for assigning to those n forked copies of the job. Every copy of the job, if scheduled to run on a node in the next round, will continue job training with its consolidated model parameters for its assigned number of steps.

C. Theoretical Analysis

Theorem 3 (Maximal Resource Utilization): HadarE achieves the maximal cluster resource utilization of an n-node cluster for training a batch of jobs forking its every job to n copies for training.

Proof: Suppose there are n nodes in a cluster running multiple jobs with the time slot per round of T_S which is shorter than the shortest job's training time. The metric of interest, cluster resource utilization (CRU)[1], for a single round, is the ratio of the total busy time spans during T_S for all n nodes to ($T_S \times n$), where the former is given by $\sum_{i=1}^{n} T_i^B$, with

[1]For clarity, CRU_k^x refers to the CRU of k total jobs (denoted by subscript), with each job forked to x copies (denoted by superscript).

T_i^B denoting the busy time span for node i, $1 \leq i \leq n$, CRU $= \frac{1}{T_S \times n} \sum_{i=1}^{n} T_i^B$.

Our proof first considers one job which takes R_1 rounds, with the following four cases to be examined:

Case I: No job forking. For only one job without job forking, *HadarE* will reduce to *Hadar* with only one of the nodes occupied in each round by the job. CRU_1^1 in this case is given by $\text{CRU}_1^1 = \frac{1}{R_1} \sum_{r=1}^{R_1} \text{CRU}(r)$, where r, $1 \leq r \leq R_1$, refers to the round number.

Case II: Job forking to x copies ($1 < x < n$). With the job forked into x copies, for $1 < x < n$, cluster resource utilization (denoted by CRU_1^x) then rises in a single round, as x nodes (out of n total nodes) will be running x forked jobs. As x copies of the job are trained simultaneously (before their obtained model parameters are consolidated, in preparation for the next training round), the total number of rounds to complete the job is reduced accordingly (to equal the ceiling function of the ratio $\frac{R_1}{x}$). The cluster resource utilization is therefore given by $\text{CRU}_1^x = \frac{1}{\lceil R_1/x \rceil} \sum_{r=1}^{\lceil R_1/x \rceil} \text{CRU}(r)$.

Case III: Job forking to n copies. This case forks the job to n copies, giving rise to $\text{CRU}_1^n = \frac{1}{\lceil R_1/n \rceil} \sum_{r=1}^{\lceil R_1/n \rceil} \text{CRU}(r)$.

Case IV: Job forking to (n + j) copies. Since only n copies will be assigned to the nodes in each round, the additional forked job will not reduce the number of rounds needed to complete model training, and thus cluster resource utilization (CRU_1^{n+j}) is the same as CRU_1^n. By observing equations above, we have

$$\text{CRU}_1^1 < \text{CRU}_1^x < \text{CRU}_1^n = \text{CRU}_1^{n+j}, \tag{11}$$

where j refers to additional copies of forked jobs beyond n copies. Eq. (11) signifies that cluster resource utilization under a single job is maximum when the job is forked into n copies, serving as the initial step of our proof.

Let's assume that Eq. (11) is true for $k\ (>1)$ jobs with R_k total rounds and the CRU of CRU_k. We have

$$\text{CRU}_k^1 < \text{CRU}_k^x < \text{CRU}_k^n = \text{CRU}_k^{n+j}. \tag{12}$$

Next, the derivation of CRU_{k+1} under $k+1$ jobs is provided by considering the additional job, as follows. For simplicity, let the additional job be the last one. The job can then be viewed as what we examined above for the single job situation, with R_1 rounds to complete without forking. If the last job for scheduling is not the additional one, the same conjectural process holds.

For $k+1$ jobs forked into x copies, CRU_{k+1}^n and CRU_{k+1}^x are similar when number of forked jobs is greater than or equal to number of nodes, $n \leq [(k+1) * x]$, but CRU_{k+1}^x decreases significantly than CRU_{k+1}^n for each additional round when $[(k+1) * x] < n$. Evidently, for the latter case, the total number of rounds to complete the remaining jobs when forked into x copies is higher than when forked into n copies, so the CRU_{k+1}^x over the entire process reduces along with each additional round of the process. For example, assume only one job is remaining when $(k * x) \leq n$, and it takes R_n^{re} rounds to complete the remaining job when forked into n copies, then the remaining job will complete in $\left\lceil \frac{R_n^{re} * n}{x} \right\rceil$ rounds when forked into x copies. The CRU for the remaining rounds (without considering the overheads) is $\frac{x*100}{n}\%$ (or 100%) when forked into x (or n) copies. Thus, CRU_{k+1}^n is higher than CRU_{k+1}^x. Similarly, for jobs without forking, when number of jobs is less than number of nodes ($(k+1) < n$), the CRU in each round (without considering overhead) is $\frac{1*100}{n}\%$. With the aforementioned fact about the CRUs of $k+1$ jobs under different cases (without forking, every job forked into x copies, every job forked into n copies, and every job forked into $x+j$ copies) and the facts of Eqs. (11) and (12), we infer that CRU is smaller without job forking than with every job forked into x copies and that the maximum CRU is obtained when each job is forked into n copies, arriving at

$$\text{CRU}_{k+1}^1 < \text{CRU}_{k+1}^x < \text{CRU}_{k+1}^n = \text{CRU}_{k+1}^{n+j}. \tag{13}$$

Since Eq. (11), the base case, is true, and Eq. (12) is assumed to be true for proving Eq. (13), we conclude that the maximum CRU, is attained when each job is forked to n copies, yielding

$$\text{CRU}_m^1 < \text{CRU}_m^x < \text{CRU}_m^n = \text{CRU}_m^{n+1}, \tag{14}$$

where m is any number of jobs in a batch. ■

Corollary: Under *HadarE* with every job forked to n copies in an n-nodes cluster, no idle node exists in any scheduling round but possibly the last one.

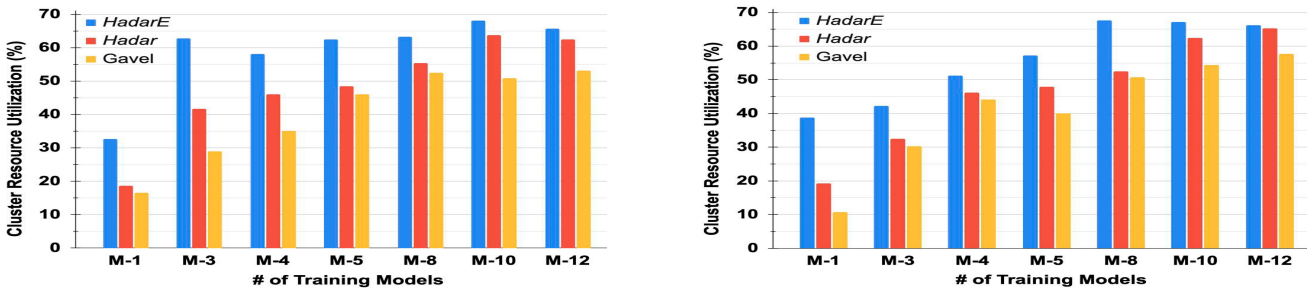
For every job, there are n forked jobs. Each of the n copies can be assigned to one of the nodes. So, every node will get a forked job to complete in each round. Nodes will never be idle if all nodes are needed to complete remaining jobs. Hence, only the last round of the entire training course may a node be idle, and any earlier round will have every node assigned with one remaining job.

VI. Evaluation on Physical Clusters

Extensive experiments are conducted to evaluate *HadarE* on physical GPU clusters either leased from the AWS (Amazon Web Services) Cloud (called the AWS cluster) or available at our research lab (called the testbed cluster), for comparison with *Hadar* and Gavel.

A. Experimental Setup

The AWS physical cluster comprises five nodes located in the same AWS region to keep the network latencies low, including one p3.2xlarge node equipped with a Tesla v100 GPU having 16 GB on-board VRAM, two p2.xlarge nodes each equipped with a Tesla K80 GPU having 12 GB on-board VRAM, and two g4dn.xlarge nodes each equipped with a Tesla T4 GPU having 16 GB on-board VRAM. Meanwhile, the heterogeneous testbed cluster available in our lab includes five nodes equipped respectively with Nvidia Titan RTX GPUs (24 GB VRAM), Tesla T4 GPU (16 GB VRAM), Nvidia T400 GPU (4 GB VRAM), GeForce RTX 3090 GPUs (24 GB VRAM), and Nvidia RTX A2000 GPU (6 GB VRAM). While some cluster nodes have two GPUs each, our evaluation always utilizes one GPU for every node.



(a) Cluster Resource Utilization (CRU) for AWS Cluster

(b) Cluster Resource Utilization (CRU) for testbed Cluster

Fig. 8. Comparison of cluster resource utilization among Gavel [10], *Hadar* [1], and *HadarE* for AWS and testbed clusters.

TABLE IV
DETAILS OF FIVE DL MODELS EMPLOYED TO CONSTRUCT WORKLOAD BATCHES RUN ON PHYSICAL CLUSTERS FOR EVALUATION

Training Job	Model	Dataset	Size
Image Classification (IC)	ResNet-18 [30]	CIFAR-10 [32]	S
Language Modeling (LM)	LSTM [33]	Wikitext-2 [34]	L
Language Translation (LT)	Transformer [36]	Multi30K [37] (de-en)	L
Recommendation System (RS)	Recorder [41]	ML-20M [42]	XL
MiMa Weather Predictions (MM)	Encoder-Decoder Transformer [39], [40], [43]	Mesonet [44] and WRF-HRRR [45]	M

B. Workloads and Datasets

Experiments are undertaken under five distinct DL models, with four of them present in a Microsoft trace [38] and the fifth for predicting weather parameters, called the Encoder-Decoder Transformer model [39] [40]. Those DL models are summarized in Table IV, and they cover different applications with their dataset sizes ranging from S (small) to XL (extra large), same as trace-driven evaluation. The experiments are performed under batches of seven workload mixes, which involve various numbers of DL models, ranging from 1 to 12. Specifically, workload mix-1 (M-1) involves just one MiMa weather prediction model, whereas workload mix-3 (M-3) contains one Language Translation model and two MiMa models, denoted by <LT, 2×MM>. Other five workload mixes are: M-4 = <IC, LM, LT, MM>, M-5 = <IC, LM, LT, RS, MM>, M-8 = <IC, LM, LT, RS, 4×MM>, M-10 = <IC, LM, LT, RS, 6×MM>, and M-12 = <IC, LM, LT, RS, 8×MM>.

C. Evaluation Results

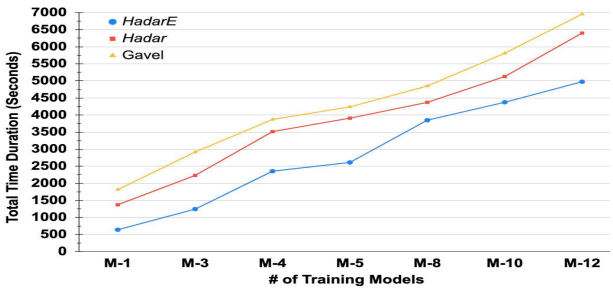
We conducted experiments to compare *Hadar* and *HadarE* against the best previous DL scheduler, Gavel [10], in terms of performance metrics of interest, including cluster resource utilization (CRU), the total time duration (TTD) taken to complete a batch of jobs, and the average job completion time (JCT) of all jobs.

Cluster Resource Utilization (CRU). CRU refers to the ratio of the total busy times of all cluster nodes over the allocated time slots of all nodes. Fig. 8 compares the resource utilization result of three schedulers, Gavel, *Hadar*, and *HadarE*. As evident by the Fig. 8 *Hadar* exhibits greater cluster resource utilization than Gavel, enjoying cluster utilization gain of 1.20× (or 1.21×) in comparison with Gavel on the AWS (or testbed) cluster, as can be observed in 8(a) (or 8(b)). *HadarE* achieves a 1.56× (or 1.62×) performance gain on average against Gavel on AWS (or testbed). Apparently, the CRU gain of *HadarE* is dictated by the difference between the number of training jobs left and the cluster node count. For example, if just one job is left to be trained on a 5-node cluster, four nodes are idle under all schedulers but *HadarE*, which lets five copies of the remaining job run concurrently on all nodes to get the highest CRU utilization possible.

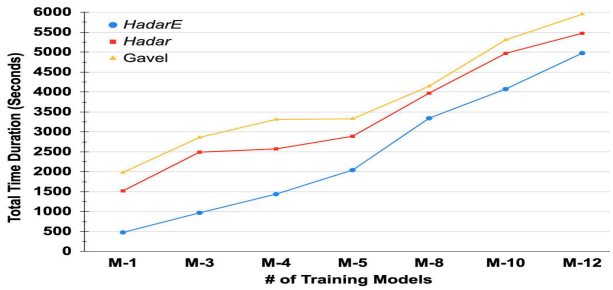
Total Time Duration (TTD). The TTD results of seven different mixes of workloads under AWS and testbed clusters are illustrated in Fig. 9. From the figure, it is clear that *Hadar* outperforms its counterpart, Gavel, in having smaller TTDs to complete all training job(s) under every workload mix on both physical clusters. For example, Gavel takes some 4200 seconds (or 3300 seconds) to finish the M-5 workload mix versus 3900 seconds (or 2800 seconds) required by *Hadar* on the AWS (or our testbed) luster, to enjoy the training speedup of 1.18× (or 1.15×), as shown in Fig. 9(a) (or Fig. 9(b)). For all the seven workload mixes, *Hadar* on average exhibits the speedup of 1.17× (or 1.16×) on the AWS (or testbed) cluster, in comparison to Gavel. Note that the performance gaps between *Hadar* and Gavel change insignificantly across all workload mixes, because if a given mix sees the cluster node(s) idle under Gavel, the mix is expected to idle the cluster node(s) under *Hadar* as well.

As expected, *HadarE* achieves further performance gains against *Hadar* (or Gavel), to yield the mean speedup of 1.79× (or 2.12×) over all seven workload mixes. It is due mainly to forking every training job to multiple copies so that a job can be trained on multiple nodes simultaneously to shorten the TTD. This way lets *HadarE* achieve more pronounced performance gains for workload mixes that cause Gavel and *Hadar* to idle cluster nodes.

Average Job Completion Time (JCT). As demonstrated in Fig. 10, *Hadar* consistently exhibits smaller JCT values than its previous counterpart, Gavel, for all seven workload mixes (with 1 to 12 training models) under both AWS and testbed clusters. Given the workload with 5 training models (M-5), for example, JCT is 1.23× (or 1.43×) smaller for *Hadar* than for Gavel under the AWS (or testbed) cluster. The maximal JCT and

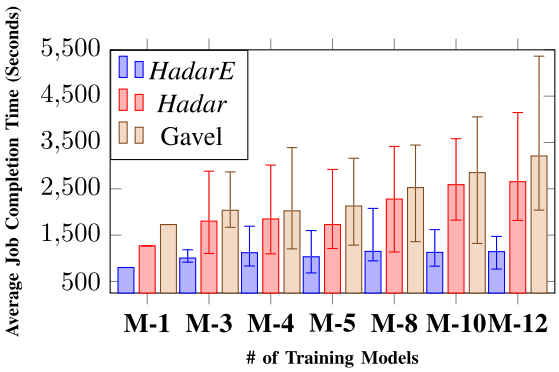


(a) Total Time Duration (TTD) for AWS cluster

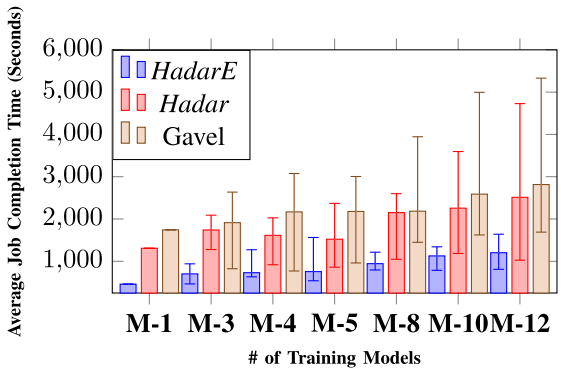


(b) Total Time Duration (TTD) for testbed cluster

Fig. 9. Comparison of TTD among Gavel [10], *Hadar* [1], and *HadarE* for AWS and testbed clusters.



(a) Average Job Completion Time for AWS cluster



(b) Average Job Completion Time for testbed cluster

Fig. 10. Comparison of JCT among Gavel [10], *Hadar* [1], and *HadarE* for AWS and testbed clusters.

TABLE V
COMPARATIVE INFERENCE QUALITY VALUES FOR MODELS TRAINED UNDER HADARE VERSUS UNDER *HADAR*

Training Job	Forking (under *HadarE*)	No Forking (under *Hadar*)	Metric
Language Translation [36]	54.690	52.410	ACC
Image Classification [30]	91.620	87.340	ACC
Recommendation System [41]	38.700	40.300	MSE
Language Modeling [33]	4.310	4.460	MSE
MiMa Weather Predictions [39], [40]	0.025	0.028	MSE

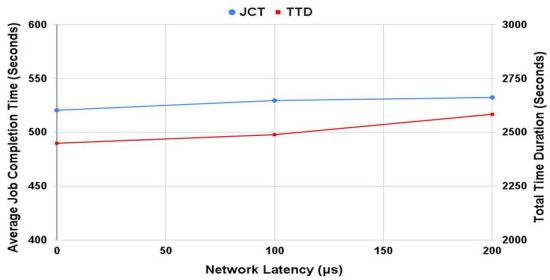
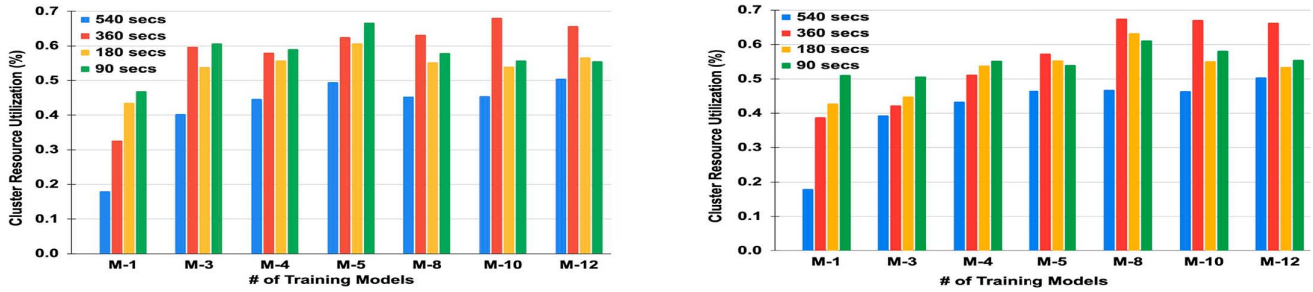


Fig. 11. Sensitivity analysis on the network latency of *HadarE* in testbed cluster.

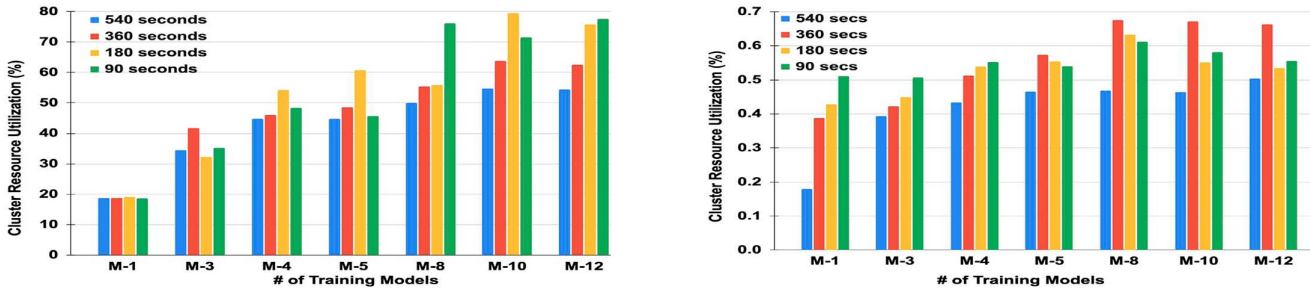
the minimal JCT for a given workload mix (with more than one training model) and a scheduler are indicated by a range mark on its average JCT bar. For the workload mix with five training models, as an example, the maximal JCT and the minimal JCT on the AWS cluster under *Hadar* equal 2918 seconds and 1210 seconds, respectively, whereas they under Gavel are 3159 seconds and 1283 seconds, according to Fig. 10(a). Across all seven workload mixes, *Hadar* enjoys 1.17× (or 1.23×) shorter JCT on average, when compared with Gavel, under the AWS (or testbed) cluster, according to Fig. 10(a) (or Fig. 10(b)). As expected, *HadarE* exhibits further JCT reduction in comparison to Gavel, yielding JCT reduction by 2.23× (or 2.76×) under the AWS (or testbed) cluster. It also enjoys substantial JCT reduction, in comparison with *Hadar* for both AWS and testbed clusters, due to its effective boost in resource utilization. In fact, the reduction degrees from *HadarE* to *Hadar* are observed to be consistently far larger than those from *Hadar* to Gavel, under both clusters. Interestingly, the JCT range of every workload mix (with more than one training model) is more confined under *HadarE* than under Gavel (and *Hadar* as well) for both clusters, due to the highest node resource utilization under *HadarE* to let any training job be executed on multiple nodes concurrently and to leave no node idle, as long as job copies still exist for training.

Model Quality. *HadarE* accelerates training workload mixes and thus shortens average JCT substantially, by forking every DL training job (model) so that multiple cluster nodes can train a sgiven model concurrently for the highest CRU values. Amid



(a) Results on AWS cluster (b) Results on testbed cluster

Fig. 12. CRU results of various slot time spans on AWS and testbed clusters under *HadarE*.



(a) Results on AWS cluster (b) Results on testbed cluster

Fig. 13. CRU results of various slot time spans on AWS and testbed clusters under *Hadar*.

training speedups, *HadarE* is found to obtain trained models able to exhibit higher inference quality levels than those trained by *Hadar*, as illustrated in Table V. All the five DL models trained for workload mix M-5 under the two schedulers on our testbed cluster are compared for their inference quality results, in terms of accuracy (ACC) and mean squared error (MSE).

The quality metric of ACC (or MSE) is for the language translation model [36] and the image classification model [30] (or the remaining three models); see Table V. It is found that the language translation model trained by *HadarE* achieves higher accuracy at 54.69%, in contrast to 52.41%. Likewise, *HadarE* trains the encoder-decoder transformer model to yield the smaller MSE of 0.025 versus 0.028 for the model trained by *Hadar*. Interestingly, *HadarE* trains all the five DL models to deliver higher quality than *Hadar* consistently, despite its training speedups, possibly resulting from its use of heterogeneous cluster nodes to train models with more powerful ones undertaking larger number of steps before model parameter aggregation and consolidation (described in Section V-B) for better model generality. Note that the DL models trained under other workload mixes (e.g., M-8, M-10, etc.) enjoy similar inference quality advantages by *HadarE* than by *Hadar*, so do the models trained on an AWS cluster.

D. Network Latency Sensitivity Analysis

HadarE, for distributed DL training, is inherently susceptible to network latency caused by traffic congestion or switching (within the same enterprise) delays. The experimental testbed clusters used in this study reside within the same local network, with negligible communication latencies; therefore, no significant latency effects were observed. However, in scenarios where working nodes are geographically distributed, network latency may have noticeable impacts on key metrics, such as TTD and JCT. To simulate such cases, various network delays (up to 200 microseconds) were introduced to pairs of nodes for evaluating *HadarE*'s sensitivity to latency. Fig. 11 illustrates the additional time incurred for JCT and TTD under the microsecond latency scales. It should be noted that a typical network switch has a latency of < 5 μs in practice, suggesting our experiments are in scenarios where working nodes are situated far apart. Introducing a 100 μs delay increased JCT and TTD time by approximately 2%, while a 200 μs delay led to their time rises of about 2.5% and 5.5%, respectively. In more extreme cases, with a latency of 100 ms (not shown in Fig. 11), JCT (or TTD) increased by around 4% (or 6%). These results demonstrate that *HadarE* remains largely resilient to the network latency, as performance degradation remains negligible even under unusually significant delays.

E. Impact of Time Slots

Models are trained under *HadarE* on rounds of a fixed slot time span, as under *Hadar*. Upon completing the assigned number of training steps for a model or when the slot time expires, a cluster node notifies the Job Tracker (depicted in Fig. 7) of its obtained model parameter values and training progress in terms of the finished training step count, for model aggregation and consolidation therein before scheduling the next round of

training job assignments, as detailed in Section V-B. Intuitively, the training performance of a given workload mix is expected to be impacted by slot time length, with a smaller length to have better performance because the workload is then better distributed across all cluster nodes. However, *HadarE* involves overhead due (1) mainly to communications between the Job Tracker and every assigned cluster node and (2) slightly to model aggregation and consolidation, making an excessively short slot time unfavorable. As a result, its best training performance (with the highest CRU) is expected to vary for different workload mixes, as demonstrated in Fig. 12. From Fig. 12(a) (or Fig. 12(b)), training performance peaks at the slot time of 360 seconds for large workload mixes, namely, M-8 to M-12 on the AWS cluster (or M-5 to M-12 on our testbed cluster), as somewhat expected. When the slot time shrinks below 360 seconds, the overhead amounts then dwarf the benefits due to better workload distribution among cluster nodes. For small workload mixes, on the other hand, a short slot time yields the highest CRU, as can be observed for M-1 to M-5 (or M-1 to M-4), to enjoy the best training performance under the slot time of 90 seconds on the AWS cluster (or our testbed cluster).

Like *HadarE*, *Hadar* also incurs communication overhead between its Scheduler and assigned cluster nodes (see Fig. 2), albeit at a lighter degree due to fewer jobs involved (without forking). Unlike *HadarE*, it is free from model aggregation and consolidation. The CRU results of various slot time spans for workload mixes trained on the AWS cluster (or our testbed cluster) under *Hadar* are depicted in Fig. 13(a) (or Fig. 13(b)). It is observed from Fig. 13(a) that CRU results on an AWS cluster are the largest under a short time slot of 90 or 180 seconds for all workload mixes, except M-3 (which peaks under the time slot of 360 seconds). This is expected because *Hadar* incurs a lighter overhead than *HadarE* to favor a shorter time slot. On our testbed cluster, whose three nodes (out of five) have relatively old motherboards with slow PCIe 3.0 to suffer from larger communication overhead; however, CRU results peak at a long slot time of 360 seconds for M-5 to M-12, as illustrated in Fig. 13(b). For small workload mixes (of M-1 to M-4), however, the best training performance is achieved under the shortest slot time of 90 seconds, partially because DL models can be trained mostly on cluster nodes with new motherboards to have PCIe 4.0 that curbs communication overhead, thus making better job workload distribution, which outweighs increased communication overhead.

VII. Conclusion

This paper has treated a novel task-level heterogeneity-aware cluster scheduler, *Hadar*, which aims to optimize such performance metrics as cluster resource utilization, total time duration, and average job completion time. The novel scheduler is formulated into an optimization problem for its solution, utilizing the primal-dual framework for task-level resource allocation across both temporal and spatial dimensions. A theoretical analysis on the proposed scheduler has been undertaken to show its polynomial runtime and a long-term performance guarantee with a bounded competitive ratio for job utility, implying approximate optimal solutions within proven constant bounds. Leveraging the dynamic programming structure, *Hadar* generates optimal scheduling decisions effectively. In addition, *Hadar* is enhanced by forking each job into multiple copies to let jobs trained concurrently on heterogeneous GPUs resided on separate cluster nodes to further boost CRU levels and thus shorten the total time duration, arriving at *HadarE*. Besides considerable training acceleration, *HadarE* is also shown to train DL models with high inference quality than *Hadar*.

References

[1] A. Sultana, F. Xu, X. Yuan, L. Chen, and N. Tzeng, "Hadar: Heterogeneity-aware optimization-based online scheduling for deep learning cluster," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS),* May 2024, pp. 681–691.

[2] Y. Wu et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*.

[3] A. Xu, Z. Liu, Y. Guo, V. Sinha, and R. Akkiraju, "A new chatbot for customer service on social media," in *Proc. Conf. Human Factors Comput. Syst.*, 2017, pp. 3506–3510.

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

[5] "API reference for amazon elastic compute cloud." AWS. Accessed: 2024. [Online]. Available: https://docs.aws.amazon.com/pdfs/AWSEC2/latest/APIReference/ec2-api.pdf#API_DescribeElasticGpus

[6] "GPU-accelerated Microsoft Azure." NVIDIA Corporation. Accessed: 2008. [Online]. Available: https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/microsoft-azure/

[7] "Google cloud GPU." Google. Accessed: 2008. [Online]. Available: https://cloud.google.com/gpu/

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

[9] M. Jeon, et al., "Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads," in *Proc. USENIX Annu. Tech. Conf* 2019, pp. 947–960.

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

[11] C. Trishul, S. Yutaka, A. Johnson, and K. Karthik, "Project Adam: Building an efficient and scalable deep learning training system," in *Proc. USENIX Symp. Oper. Syst. Des. Implementation*, 2014, pp. 571–582.

[12] Q. Weng et al., "MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 945–960.

[13] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale GPU datacenters," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2021, pp. 1–15.

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

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

[16] Z. Yang et al., "Hydra: Deadline-aware and efficiency-oriented scheduling for deep learning jobs on heterogeneous GPUs," *IEEE Trans. Comput.*, vol. 72, no. 8, pp. 2224–2236, Aug. 2023.

[17] S. Jayaram Subramanya, D. Arfeen, S. Lin, A. Qiao, Z. Jia, and G. R. Ganger, "Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling," in *Proc. 29th Symp. Oper. Syst. Princ.*, 2023, pp. 642–657.

[18] J. Mohan, A. Phanishayee, J. Kulkarni, and V. Chidambaram, "Looking beyond GPUs for DNN scheduling on multi-tenant clusters," in *Proc. 16th USENIX Symp. Oper. Syst. Des. Implementation (OSDI),* 2022, pp. 579–596.

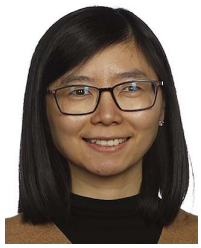
[19] M. Ghorbani, Y. Wang, Y. Xue, M. Pedram, and P. Bogdan, "Prediction and control of bursty cloud workloads: a fractal framework," in *Proc. Int. Conf. Hardware/Softw. CoDes. Syst. Synthesis*, 2014, pp. 1–9.
[20] Z. Ye et al., "Deep learning workload scheduling in GPU datacenters: A survey," *ACM Comput. Surv.*, vol. 56, no. 6, pp. 1–38, 2024.
[21] C. Li, Z. Lin, L. Tian, and B. Zhang, "A scheduling algorithm based on critical factors for heterogeneous multicore processors," *Concurrency Computation, Pract. Exp.*, vol. 36, no. 7, 2024, Art. no. e7969.
[22] Y. Xiao, S. Nazarian, and P. Bogdan, "Plasticity-on-chip design: Exploiting self-similarity for data communications," *IEEE Trans. Comput.*, vol. 70, no. 6, pp. 950–962, Jun. 2021.
[23] S. Chen, M. Ghorbani, Y. Wang, P. Bogdan, and M. Pedram, "Trace-based analysis and prediction of cloud computing user behavior using the fractal modeling technique," in *Proc. IEEE Int. Congr. Big Data*, 2014, pp. 733–739.
[24] S. Duan et al., "A structure-aware framework for learning device placements on computation graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, vol. 37, pp. 81748–81772.
[25] Z. Fan and M. Peng, "DGMF: A unified dynamic mapping framework for graph neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 18, no. 3, pp. 1–30, 2025.
[26] P. Zheng, R. Pan, T. Khan, S. Venkataraman, and A. Akella, "Shockwave: Fair and Efficient Cluster Scheduling for Dynamic Adaptation in Machine Learning," in *Proc. 20th USENIX Symp. Netw. Syst. Des. Implementation (NSDI 23),* Apr. 2023, pp. 703–723.
[27] K. Ma et al., "PPS: Fair and efficient black-box scheduling for multi-tenant GPU clusters," *Parallel Comput.*, vol. 120, no. C, 2024, Art. no. 103082.
[28] N. Buchbinder and J. S. Naor, "The design of competitive online algorithms via a primal–dual approach," *Found. Trends Theor. Comput. Sci.*, vol. 3, no. 2–3, pp. 93–263, 2009.
[29] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 495–503.
[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. (CVPR),* 2016, pp. 770–778.
[31] J. Deng, et al., "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2009, pp. 248–255.
[32] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. of Toronto, 2012.
[33] A. Shroyer. "Word-level language modeling RNN." GitHub. Accessed: 2008. [Online]. Available: https://github.com/pytorch/examples/tree/main/word_language_model
[34] M. Stephen, X. Caiming, B. James, and S. Richard, "Pointer sentinel mixture models," in *Proc. Conf. Learn. Representations*, 2017, pp. 1–15.
[35] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. Int. Conf. Comput. Vision*, 2017, pp. 2242–2251.
[36] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.,* 2017, pp. 1–11.
[37] D. Elliott, S. Frank, K. Sima'an, and L. Specia, "Multi30K: Multilingual English-German image descriptions," in *Proc. Workshop Vision Lang.*, 2016, pp. 70–74.
[38] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant GPU clusters for DNN training workloads," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2019, pp. 947–960.
[39] Y. Zhang et al., "Precise weather parameter predictions for target regions via neural networks," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discovery Databases (ECML-PKDD),* Sep. 2021, pp. 151–167.
[40] PREFER, "MiMa codes and pertinent information," 2023. [Online]. Available: https://prefer-nsf.org/publications.html
[41] A. Moussawi, "Towards large scale training of autoencoders for collaborative filtering," in *Proc. Late-Breaking Results Track Part 12th ACM Conf. Recommender Syst.*, 2018.
[42] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, pp. 1–19, Dec. 2015.
[43] Y. Zhang et al., "Regional weather variable predictions by machine learning with near-surface observational and atmospheric numerical data," *IEEE Trans. Geosci. Remote Sens.*, vol. 63, pp. 1–21, 2025.
[44] Kentucky Mesonet, "Kentucky Mesonet at WKU," 2021. [Online]. Available: https://www.kymesonet.org/
[45] National Oceanic and Atmospheric Administration, "The High-Resolution Rapid Refresh (HRRR)," 2022. [Online]. Available: https://rapidrefresh.noaa.gov/hrrr

Abeda Sultana received the B.S. degree from the Department of Computer Science and Engineering, University of Dhaka, Bangladesh, in 2018, and the Ph.D. degree from the School of Computing and Informatics, University of Louisiana, Lafayette. Her research interests include distributed machine learning, scheduling of distributed systems, resource allocation, and federated learning.

Nabin Pakka (Graduate Student Member, IEEE) received the B.S. degree in electronics and communication engineering from Tribhuvan University, Nepal, in 2021, and the M.S. degree in computer science from the University of Louisiana, Lafayette, where he is currently working toward the Ph.D. degree. His research interests include machine learning, computer vision, natural language processing, and distributed computing.

Fei Xu received the B.S., M.E., and Ph.D. degrees from Huazhong University of Science and Technology, Wuhan, China, in 2007, 2009, and 2014, respectively. He received the Outstanding Doctoral Dissertation Award in Hubei province and the ACM Wuhan and Hubei Computer Society Doctoral Dissertation Award in 2015. He is currently a Professor with the School of Computer Science and Technology, East China Normal University, China. His research interests include cloud computing and datacenter, virtualization technology, and distributed systems.

Xu Yuan received the Ph.D. degree from Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, in 2016. He is currently an Associate Professor with the Department of Computer and Information Sciences, University of Delaware, Newark. His research interests include artificial intelligence (AI), AI for science, cybersecurity, networking, and cyber-physical systems. He received the Best Paper Award and the Distinguished Paper Award from DSN 2023.



Li Chen (Senior Member, IEEE) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Toronto, in 2018. She is currently an Associate Professor with the School of Computing and Informatics, University of Louisiana, Lafayette, USA. Her research interests include big data analytics systems, machine learning systems, cloud computing, datacenter networking, and resource allocation.

Nian-Feng Tzeng received the Ph.D. degree in computer science from the University of Illinois, Urbana-Champaign, in 1986. He has been with the Center for Advanced Computer Studies, School of Computing and Informatics, University of Louisiana, Lafayette, since 1987. His research interests include high-performance computer systems, computer networks, and parallel and distributed processing.