Hadar: Heterogeneity-Aware Optimization-Based Online Scheduling for Deep Learning Cluster

Abeda Sultana[1] Fei Xu[2] Xu Yuan[3] Li Chen[1] Nian-Feng Tzeng[1]
[1]School of Computing and Informatics, University of Louisiana at Lafayette, USA
[2]School of Computer Science and Technology, East China Normal University, China
[3]Department of Computer and Information Sciences, University of Delaware, USA

***Abstract*—With the wide adoption of deep neural network (DNN) models for various applications, enterprises, and cloud providers have built deep learning clusters and increasingly deployed specialized accelerators, such as GPUs and TPUs, for DNN training jobs. To arbitrate cluster resources among multi-user jobs, existing schedulers fall short, either lacking fine-grained heterogeneity awareness or hardly generalizable to various scheduling policies. To fill this gap, we propose a novel design of a task-level heterogeneity-aware scheduler, *Hadar*, based on an online optimization framework that can express other scheduling algorithms. *Hadar* leverages the performance traits of DNN jobs on a heterogeneous cluster, characterizes the task-level performance heterogeneity in the optimization problem, and makes scheduling decisions across both spatial and temporal dimensions. The primal-dual framework is employed, with our design of a dual subroutine, to solve the optimization problem and guide the scheduling design. Extensive trace-driven simulations with representative DNN models have been conducted to demonstrate that *Hadar* improves the average job completion time (JCT) by $3\times$ over an Apache YARN-based resource manager used in production. Moreover, *Hadar* outperforms Gavel[1], the state-of-the-art heterogeneity-aware scheduler, by $2.5\times$ for the average JCT, and shortens the queuing delay by 13% and improve FTF (Finish-Time-Fairness) by 1.5%.**

***Index Terms*—distributed deep learning, scheduling, optimization**

I. Introduction

The application of deep learning has become ubiquitous across various domains, including but not limited to speech recognition, natural language processing [2], supercomputing, and social media [3]. To facilitate the ever-increasing demand for deep neural network (DNN) training [4], large enterprises and cloud providers [5], [6], [7] have constructed dedicated deep learning clusters. These clusters have increasingly deployed specialized accelerators, such as GPUs, TPUs, and FPGAs, to accelerate DNN model training with intricate architectures. Expensive resources in such clusters require efficient scheduling among multiple user jobs, considering overall deep learning job performance, cluster-wide resource utilization, fairness, *etc.*

The research is supported in part by the National Science Foundation under grants OIA-2019511, OIA-2327452, 2348452 and 2315613, in part by the Louisiana Board of Regents under Contract LEQSF(2019-22)-RD-A-21, in part by the NSFC under Grant 62372184, in part by the Science and Technology Commission of Shanghai Municipality under Grant 22DZ2229004.
Corresponding author: Li Chen. Email: li.chen@louisiana.edu

To this end, existing efforts have proposed a number of GPU cluster schedulers (*e.g.,* [8], [4], [9]) for deep learning. However, they either lack the awareness of job and resource heterogeneity, leading to suboptimal job performance, or are tightly coupled with specific objectives, hardly generalized to other operational goals. It has been observed in [1] that DNN 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 $10\times$ speedup when trained on an NVIDIA V100 GPU compared with a K80 GPU, while an A3C Deep Reinforcement Learning model only exhibits $2\times$ acceleration. In light of such observations, Gavel [1] has been proposed as a heterogeneity-aware cluster scheduler, which is the first to address the aforementioned performance heterogeneity of DNN training jobs across multi-type accelerators in a cluster. It utilizes an optimization-based scheduling framework to explicitly account for job placement and performance heterogeneity, able to be generalized to express other scheduling policies. However, it does not explicitly characterize performance heterogeneity at a finer-grained task level. This means that parallel tasks of a DNN training job cannot be effectively scheduled on heterogeneous devices. 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 instance. This limitation highlights the need for a more sophisticated and flexible scheduler that can make the best use of the available cluster resources while accommodating task-level performance heterogeneity.

To bridge this gap, in this paper, we present a new fine-grained heterogeneity-aware online scheduler, named *Hadar*, for a deep learning cluster shared by DNN training jobs. The essence of our scheduler relies on our problem formulation and optimization framework for task-level resource allocation across both temporal and spatial dimensions, as opposed to the state-of-the-art counterpart, Gavel [1], whose optimization framework only characterizes the spatial resource allocation with just job-level heterogeneity awareness. Based on our optimization framework, we design an online scheduling algorithm, which addresses the task-level performance heterogeneity of DNN jobs on different accelerators and makes the best task-level decisions upon a scheduling event. More specifically, we begin with the goal of maximizing the overall

Accepted for presentation at 38th IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2024

job utility, a metric defined according to the job throughput and epoch number, aiming at minimizing the average job completion time of the cluster. To tackle the combinatorial online optimization problem, our scheduler *Hadar* utilizes an online primal-dual framework combined with a dual subroutine, to approximate optimal solutions within proven constant bounds. It computes the dual resource price according to time-dependent resource (number and type of accelerators, bandwidth) consumption levels to decide job admission and resource allocation accordingly. *Hadar* can express other scheduling policies in our online optimization framework as well, maintaining the favorable property of generality as in Gavel. Finally, to evaluate the performance of *Hadar*, we conduct extensive experiments both in a physical cluster and by simulation, comparing it with those of two state-of-the-art counterparts, namely Gavel [1] and Tiresias [4], as well as the YARN-CS [6] scheduler commonly used in production. Our extensive simulation studies, conducted using an event-based trace-driven simulator under various settings, demonstrate that *Hadar* reduces the average job completion time (JCT) by 1.5-1.8× when compared to Gavel, 2.1-2.5× over Tiresias, and 7-15× over YARN-CS scheduler. Meanwhile, *Hadar* improves the finish-time fairness [10] by 1.5× compared to Gavel, and also improves the makespan of jobs by 1.5× over Gavel. Furthermore, real-world experiments in a physical cluster also exhibit superior performance of *Hadar* over its counterparts. More specifically, it improves the JCT by 2.3× compared to Gavel and 3× over Tiresias. With respect to the makespan, *Hadar* achieves an improvement of 1.9× and 2.9× over the two baselines, respectively.

Overall, the main contribution of this work can be summarized as follows.

- We propose an efficient online scheduler for deep learning training jobs in a GPU cluster, addressing the performance heterogeneity of multi-type accelerators at the task-level granularity.
- We present an online optimization algorithm that employs the primal-dual framework coupled with a dual subroutine to analyze and tackle the scheduling problem on multiple heterogeneous accelerators. Our optimization framework is general enough, able to express diverse scheduling objectives.
- We prove the polynomial runtime complexity of our algorithm and also perform a competitive analysis to provide a long-term performance guarantee that approximates optimal solutions within proven constant bounds.
- We conduct extensive real-world experiments and trace-driven simulations, with results consistently demonstrating the advantages of *Hadar* on different metrics compared to the state-of-the-arts.

II. Related Work and Motivation

With the prevailing data parallel training model [11], a deep neural network (DNN) training job typically has a large number of epochs and spans multiple devices, to process voluminous input data in an iterative and distributed manner. With data parallelism, the complete set of training data is partitioned across multiple machines in a cluster, and the deep learning model is trained with the parallel implementation of stochastic gradient descent (SGD). Each worker machine maintains a local copy of the DNN model, computes the updates based on the local dataset, and synchronizes among workers periodically. Specifically, the training data is divided into equal-sized data chunks that are trained by different workers, with each data chunk further divided into equal-sized mini-batches. When a worker starts, it fetches a data chunk, processes the first mini-batch, and sends gradients to the parameter servers [11] for parameter updates. Training one mini-batch is called an iteration. Upon receiving updated parameters from all parameter servers, the worker continues computing gradients using the next mini-batch, and so on. Once an entire data chunk is processed, the worker continues training the next data chunk assigned to it. In a deep learning training job, input data chunks can be repeatedly trained for multiple rounds or epochs [12]. Each epoch refers to a complete pass through all data chunks. A training job stops after a pre-specified number of epochs.

To accommodate multi-user DNN training jobs, traditional CPU-based cluster schedulers ([13], [14], [15], [16], [17], [18], [19], [20], *etc.*) fall short, due to the lack of consideration on the unique characteristics of distributed DNN training. Recent production-scale workload analyses [21], [22] (*e.g.*, a two-month trace from a production cluster with over 6000 GPUs in Alibaba [21]) confirm such characteristics in the temporal (job runtime) and spatial (resource request) patterns around in a deep learning cluster. In addition, the heavy-tailed nature of resource requests, along with the wide range of queuing delay and job runtime, has increasingly drawn research attention to deep learning cluster scheduling (such as [8], [4], [10], [23], [24]). These works focus on designing cluster schedulers that are customized for DNN training jobs, to improve the overall job performance and resource utilization in various ways. However, they fail to effectively address the adverse impact of resource heterogeneity.

Some recent schedulers [25], [26], [1], [27] have paid attention to device and model heterogeneity to some extent, but not at a fine granularity. AlloX [25] only considers two types of devices (CPU and GPU) and the workloads are not limited to deep learning jobs. Gandiva$_{fair}$ [26] focuses merely on fairness among jobs while Hydra [27] aims at meeting deadlines. Gavel [1] accounts for the heterogeneity of both DNN training workloads and hardware devices when allocating resources among jobs. It presents a general optimization framework to characterize a number of scheduling policies. In sharp contrast, in this paper, we design a new cluster scheduler for DNN training jobs, effectively incorporating heterogeneity awareness in our online resource scheduling at the task level of finer granularity, across both temporal and spatial dimensions. Compared to Gavel [1], which is the closest counterpart of our work among the state-of-the-art, our scheduler is aware of task-level performance heterogeneity and allocates resources at finer granularity across an additional temporal dimension,

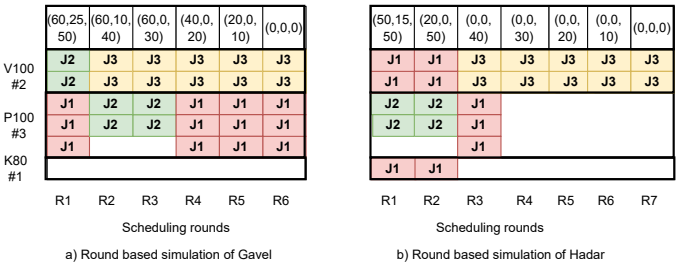


Fig. 1: An example to compare the scheduling outcomes of three jobs in a cluster with 2 V100, 3 P100, and 1 K80 GPUs, achieved by Gavel [1] and *Hadar*.

yielding marked overall performance improvement. Moreover, similar to Gavel [1], our optimization framework underlying the scheduler is also able to express other scheduling policies. We next present a toy example to highlight our insights, further strengthening our motivation.

A. Motivation Example

We consider a cluster with two V100, three P100, and one K80 GPUs. 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. These jobs exhibit different training speedup performances on different GPUs, expressed as the following throughput matrix X:

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

According to Gavel [1], the optimal allocation matrix Y^{Gavel} is calculated as follows, under the assumption that the cluster has the capacity to meet the requirements of all queued jobs:

$$Y^{Gavel} = \begin{array}{c} \\ J1 \\ J2 \\ J3 \end{array} \begin{array}{c} \begin{array}{ccc} V100 & P100 & K80 \end{array} \\ \begin{pmatrix} 0.6 & 0.4 & 0.0 \\ 0.2 & 0.6 & 0.2 \\ 0.2 & 0 & 0.8 \end{pmatrix} \end{array}$$

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 GPU is defined as the corresponding element of Y^{Gavel} divided by the number of rounds received (*i.e.*, resource allocation received). Fig. 1a illustrates the scheduling outcome over 6 rounds according to Gavel, where the first row represents the number of remaining 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. 1a.

Hadar employs a preemptive, round-based approach to allocate jobs across various GPU types. Calculating allocations in each round aims to guide the cluster toward optimal allocation. When a job is suspended, the latest model parameter would be checkpointed to stable storage to prevent loss of training progress. Asynchronous computation of allocations and worker assignments minimizes synchronous overhead, with the loading and saving of checkpoints being the only synchronous operation, dependent on the model size. The experimental section will provide insights into the overhead of the preemption algorithm. Within each round, Gavel schedules all tasks of a job on the same type of GPUs. In contrast, we exploit the flexibility of task-level allocation, to maximize the overall performance of the cluster. As shown in Fig. 1b, *Hadar* strategically assigns the tasks of job $J1$ to two $V100$ GPUs and one $K80$ GPU in round 1, resulting in a throughput of $min(40, 30) = 30$, while job $J2$ attains a throughput of 15. In comparison, Gavel's policy adopts the homogeneous allocation of tasks strictly, causing jobs $J1$ and $J2$ to achieve suboptimal throughputs in the long run. The average per round throughputs achieved by *Hadar* (or Gavel) for jobs $J1, J2$ and $J3$ are: 26.27 (or 20), 15 (or 10) and 10 (or 10). As observed, the finer-grained allocation policy employed by *Hadar* results in marked reduction in the average job completion time (JCT). Specifically, $J1$ and $J2$ in this example finish faster under *Hadar* than under Gavel, leading to 20% improvement in the average JCT.

III. Design of *Hadar*

Our overall objective is to design an online scheduler for distributed deep learning training jobs with the awareness of resource heterogeneity. The *Hadar* scheduler features a task-level scheduling granularity, which means it operates by making decisions at the level of individual tasks. A task corresponds to a configurable number of epochs over a subset of input data, offering flexibility in defining the granularity. This granularity allows *Hadar* to more effectively manage resources and schedule tasks in a fine-tuned manner, taking into account the specific requirements and characteristics of each task within deep learning workloads. Such a task-level approach is crucial for optimizing performance and resource utilization in heterogeneous computing environments, like those commonly found in deep learning clusters. 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 deep learning 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, characterized 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 online 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

TABLE I: 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
up 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

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, $\mathcal{U}_j(\cdot)$, is a general non-negative function that characterizes the value or desirability of a job's execution based on certain criteria. It specifies a job's value in terms of user-defined metric such as the job's completion time, fairness, effective throughput, *etc*. As a special case of job utility, the effective throughput, defined as the average number of iterations completed per second over the job's lifetime, is expressed as E_jN_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_jN_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 [21]. A brief notation summary is presented in Table I. **Expressing other scheduling policies.** Note that our optimization-based scheduling framework can express other scheduling objectives. For example, minimizing the average job completion time is denoted as $\min \sum_j (f_j - a_j)/J$, minimizing the makespan is represented as $\min \max_j f_j$), and achieving fairness across users can be $\min \max_j (f_j - a_j)/(f_j^{isolated} - a_j)$, considering the finish-time fairness metric [10], where $f_j^{isolated}$ is the job finish time when using $1/J$ of the cluster.

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:

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

$$s.t. \quad \mu_j \geq \mathcal{U}_j(f_{js} - a_j) - \sum_{t\in s} \sum_{h\in(t,s)} \sum_r k_h^r(t) w_{jh}^{rs}(t) \tag{3a}$$

$$k_h^r(t) \geq 0, \forall h, \forall r, \forall j, \forall t, \qquad \mu_j \geq 0, \forall j$$

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:

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

where

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

$$U_{min}^r = \frac{1}{4\eta}\min_j \frac{\mathcal{U}_j(T-a_j)}{t_j^{max}\sum_{r\in[R]} w_j^r}, \quad \forall r \tag{7}$$

$$t_j^{min} = \frac{N_j E_j}{M_j \max_r(X_j^r)}, \quad t_j^{max} = \frac{N_j E_j}{M_j \min_r(X_j^r)} \tag{8}$$

U_{max}^r and U_{min}^r imply the maximum and the minimum per-unit-resource job utility values for type-r accelerator 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 ensure the initial value of the dual objective is bounded. 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 online algorithm, to be presented in the next section. The values of U_{max}^r and U_{min}^r are calculated based on the current workload of the cluster in the online algorithm.

C. Algorithm Design

Based on the resource price and job payoff interpretations, we next present our online 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:

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

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 `line(14-15)`. The set of jobs and the allocations with minimum cost is returned from the DP function call `line(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 (`line 26-27`). The allocation with minimum cost is selected and μ_j is calculated to determine the feasibility of the allocation (`line 28-32`). According to the selected allocation, the amount of allocated resource $\gamma_h^{rc}(t)$, price function $k_h^{rc}(t)$ and server state are updated (`line 10-12`).

Algorithm 1 Online Scheduling in *Hadar*.

Input: $c_h^r, \forall h \in [H], r \in [R]$
1: **Initialize:** $w_{jh}^r(t) = 0, \gamma_h^r(t) = 0, k_h^r(t) = k_h^r(0), \ \forall j \in [J], t \in [T], h \in [H]$
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**

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 the 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 ($all\_alloc_{\neg packed}$) 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 $\alpha = \max_{r\in[R]}(1, ln\frac{U_{max}^r}{U_{min}^r})$ and U_{max}^r, U_{min}^r are defined in Eqs. (6), (7).

The concept of 2α competitiveness, as introduced in [28] and applied in *Hadar*, is a metric for assessing the performance of online algorithms in comparison to the ideal offline solution. This competitiveness ratio quantifies the maximum deviation

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 (desc. order of $x^r_j,\ \forall h \in H$)
  $\{allocs_{packed}\} \leftarrow$allocs. within a consolidated setting.
  $\{allocs_{!packed}\} \leftarrow$ allocs. independent of consolidation.
  $\{cost_{packed}\} \leftarrow \sum_h \sum_r k^r_h(t) w^r_{jh}(t), \forall allocs_{packed}$
  $\{cost_{!packed}\} \leftarrow \sum_h \sum_r k^r_h(t) w^r_{jh}(t) + comm.\ cost, \forall allocs_{!packed}$
  $alloc \leftarrow$ alloc. 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

of an online algorithm's performance from an optimal offline algorithm that has complete information about future events. In the case of *Hadar*, a 2α competitive rating indicates that the worst-case performance of *Hadar* is at most double the performance of the optimal offline solution, when adjusted for a factor of α. This factor represents the degree of variability or uncertainty inherent in the online decision-making process. Therefore, a lower 2α competitiveness ratio signifies a closer approximation to the ideal offline performance, underscoring the efficiency of *Hadar* in managing resources in deep learning clusters despite the lack of foresight that an online scheduler inherently faces. We prove the theorem in what follows.

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. (6, 7) 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 can be used to derive:

$$\begin{aligned} D_0 &= \sum_t \sum_h \sum_r U^r_{min} c^r_h \\ &= \sum_t \sum_h \sum_r \frac{1}{4\eta} \min_{j,s} \frac{\mathcal{U}_j(f_{js} - a_j)}{t^{max}_j \sum_r w^r_j} c^r_h \\ &= \frac{\sum_t \sum_h \sum_r c^r_h}{4\eta} \min_{j,s} \frac{\mathcal{U}_j(f_{js} - a_j)}{t^{max}_j \sum_r w^r_j} \\ &\leq \frac{1}{4} t^{max}_j \sum_r w^r_j \min_{j,s} \frac{\mathcal{U}_j(f_{js} - a_j)}{t^{max}_j \sum_r w^r_j}, \forall j \end{aligned} \tag{9}$$

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

$$\begin{aligned} (9) &\leq \frac{1}{4} t^{max}_j \sum_{r \in [R]} w^r_j \min_{j,s} \frac{\mathcal{U}_j(f_{js} - a_j)}{t^{max}_j \sum_r w^r_j}, \forall j \\ &\leq \frac{1}{2} \mathcal{U}_j(f_{js} - a_j) \quad \leq \frac{1}{2} OPT \end{aligned}$$

The last inequality holds because we assume the offline opti-

mal solution accepts at least one job, which is reasonable in the real-world cluster. Then we have $OPT \geq \min_{j,s} \mathcal{U}_j(f_{js} - a_j)$. Based on Lemma 1 and Lemma 2, we conclude the proof. ■

E. Implementation of Hadar

Guided by the theoretical investigation, we implement our fine-grained heterogeneity-aware scheduler, *Hadar*, as illustrated in Fig. 2. Given a set of queued jobs, the online scheduler dispatches all jobs onto different types of accelerators from different servers 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* obtains performance measurements for each runnable job on each available accelerator type either from user input or by profiling during the first few rounds of execution. For a given input, the scheduling algorithm in the allocator calculates the number and types of GPUs assigned to each job on particular servers in a given round. It considers task-level heterogeneity, straggler performance, and job packing decisions to maximize overall cluster utility.

IV. Performance Evaluation

In this section, we conduct extensive experiments to evaluate the performance of *Hadar* in terms of the job completion time (JCT), GPU utilization, makespan, and fairness, compared to the state-of-the-art schedulers.

A. Trace-driven Simulations

We have developed a discrete-time simulator to evaluate *Hadar* using a real-world trace [9]. Following the setup of the simulation experiments in Gavel [1], our simulated cluster consists of 15 nodes and a total of 20 GPUs of each type (V100, P100, and K80). The workloads are based on a Microsoft trace, summarized in Table II to be elaborated in the next paragraph. For each job (workload) in Table II, we leverage its throughput measurements from Gavel as our scheduling input. We simulate the job events such as job arrival, completion, and preemption. The overhead of checkpoint-restarts is simulated by enforcing a 10-second delay for each job that has received a new allocation which is based on the result shown in Table IV. The duration of a scheduling round is set as 6 minutes.

Synthetic Workloads and Datasets. In our experiments, 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 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 II. We considered two arrival patterns in our experiments: *static* and *continuous*. In the static pattern, all jobs were available at the beginning of the trace, and no jobs were added subsequently. In the continuous pattern, jobs were continuously submitted to the cluster, and the job arrival times were generated according to a Poisson arrival process with an interarrival rate λ.

TABLE II: Evaluation workloads: model, dataset, and relative size for each deep learning task

Task	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], [36]	Monet2photo[36]	M
Language Translation	Transformer[37]	Multi30K[38] (de-en)	L

Baselines and Metrics. We conducted experiments to compare the performance of *Hadar* with state-of-the-art deep learning cluster scheduler proposals, Gavel [1] and Tiresias [4], as well as the default production-level cluster scheduler, Apache YARN's capacity scheduler (YARN-CS) [6]. While *Hadar* considers the task-level heterogeneity of DNN training jobs for scheduling decisions, Gavel only focuses on job-level heterogeneity, and Tiresias is not aware of the heterogeneity among accelerators. The metrics used for comparison include the average job completion time (JCT) and the makespan of jobs. We also evaluate the fairness of our algorithm using finish-time fairness [10], which is the ratio of the elapsed time to finish a job given an allocation and the elapsed time to finish a job when using $\frac{1}{n}$ of the cluster, with n being the number of jobs executed on the cluster. We configure Gavel similar to its testbed experiments while keeping the objective of its optimization problem similar to ours in the comparative evaluation. Tiresias is configured with two priority queues and its `PromoteKnob` disabled. YARN-CS, as a capacity scheduler, is principally geared towards optimizing resource allocation, ensuring equitable resource distribution across users and applications. Unlike *Hadar* which is specifically engineered to enhance job completion times and ensure fairness, YARN-CS's approach to finish time fairness and minimizing overall job completion duration is less advanced. Consequently, we did not include YARN-CS in the comparison for these two metrics, as its performance benchmarks in these areas do not align closely with the objectives and design principles of *Hadar*.

1) Job Completion Time: As demonstrated in Fig. 3, *Hadar* outperformed the other schedulers in both static and continuous trace settings. For a static trace, *Hadar* reduced the average JCT by 7× compared to YARN-CS, and 1.8× and 2.5× over Gavel and Tiresias, respectively. Additionally, the median JCT of *Hadar* was 15× better than YARN-CS, 2.1× better than Gavel, and 3× better than Tiresias. Furthermore, in the continuous trace setting, *Hadar* demonstrated improved performance due to its awareness of straggling tasks and the strategic task allocation policy of the online algorithm. The scheduler shortened JCT under the continuous trace by 1.5, 5, and 2.3× when compared to Gavel, YARN-CS, and Tiresias,

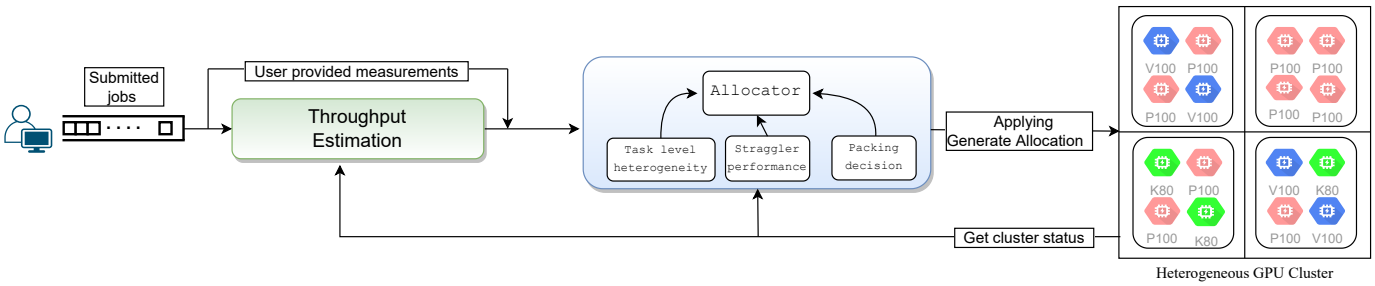
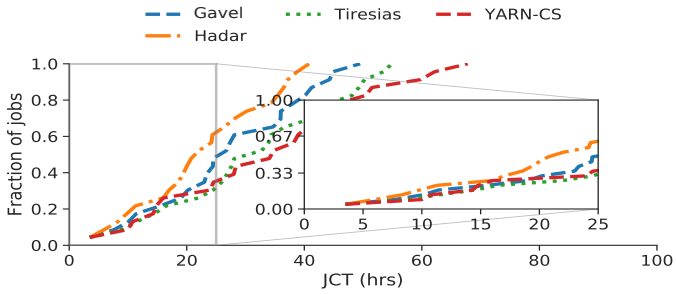
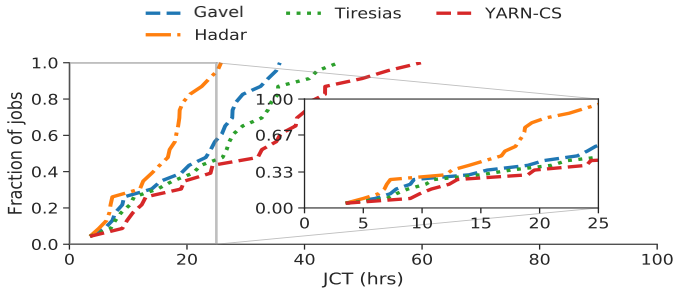


Fig. 2: The overview of *Hadar*, a fine-grained heterogeneity-aware scheduler for a GPU-based deep learning cluster.



a) Static trace result



b) Continuous trace result

Fig. 3: The accumulative faction of jobs completed along the timeline, when scheduled by Gavel [1], Tiresias [4], YARN-CS [6], and *Hadar*, respectively.

respectively. The scheduler achieved better performance in a static trace setting due to the availability of information about all jobs, which facilitated favorable allocation. However, being an online scheduler, *Hadar* can handle dynamic environments where the workload is constantly changing by taking strategic decisions. It can react to changes in workload and resource availability, thereby providing better performance in terms of the average JCT and the overall throughput. Additionally, *Hadar* handles straggling tasks more effectively by reallocating resources to other tasks that are progressing faster.

2) GPU Utilization: Fig. 4 shows a comparison of the four schedulers in terms of GPU utilization, which refers to the percentage of total job run-time during which the GPUs are utilized. The highest GPU utilization is achieved by YARN-CS due to its non-preemptive nature. However, this comes at the cost of long job completion times, as observed in the previous subsection. Gavel, on the other hand, does not enable fine-grained task-level scheduling, which leads to heterogeneous

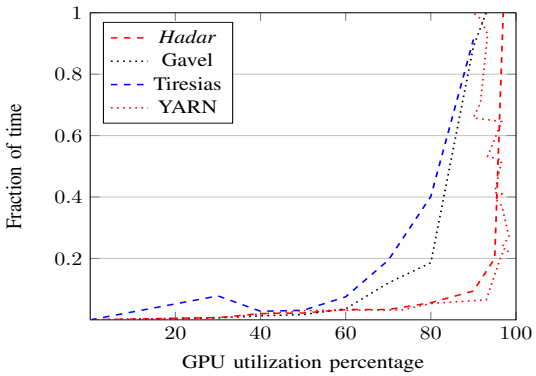


Fig. 4: Comparison of the cluster-wide GPU utilization among the four schedulers.

GPUs remaining unused even if the total number of them meets the requirement of a queued job. This results in a lower GPU utilization compared to YARN-CS. Tiresias also suffers from the same limitation as Gavel. In contrast, *Hadar* leverages the advantages of fine-grained scheduling and task-level heterogeneity consideration, which results in GPUs being utilized to a greater extent. More specifically, *Hadar* can allocate tasks to GPUs that are most suited for them, and possibly of different types when necessary, based on task characteristics and cluster resource availability. As a result, the number of GPUs that remain unused is minimized, leading to a higher GPU utilization compared to Gavel and Tiresias. Moreover, *Hadar* exhibits a similar utilization compared to YARN-CS, which indicates that it is capable of utilizing the GPUs effectively while also achieving better job completion times.

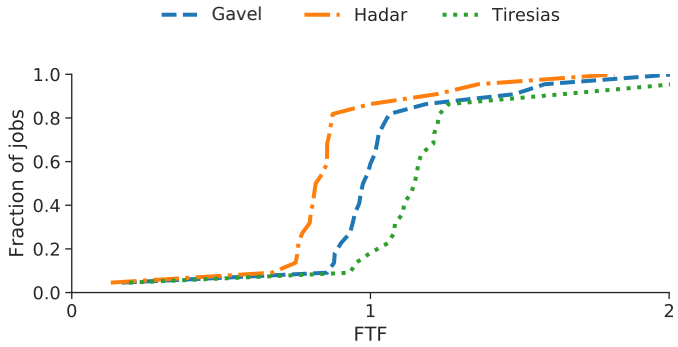


Fig. 5: Comparison of the finish-time-fairness (FTF) among Gavel [1], Tiresias [4], and *Hadar*.

3) Fairness: We conducted a comparison of the finish-time fairness (FTF) metric among Gavel, Tiresias, and *Hadar*, as presented in Fig. 5. The FTF metric measures the fairness of job completion time given an allocation. According to Fig. 5, our task-level heterogeneity-aware policy in *Hadar* outperformed Gavel and Tiresias in terms of the average FTF, with an improvement of 1.5× and 1.8× respectively. This result implies that *Hadar* is more effective in ensuring fairness in job completion time than the baseline schedulers.

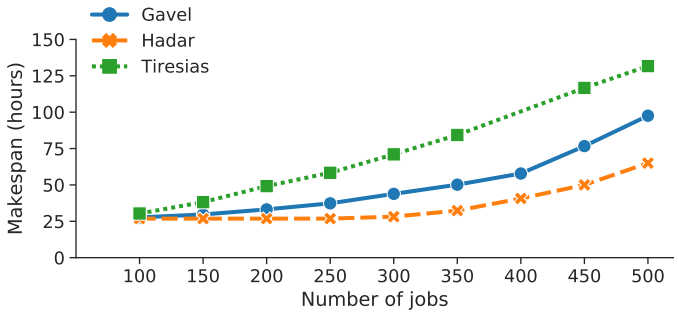


Fig. 6: Comparison of the makespan among Gavel [1], Tiresias [4], and *Hadar*.

4) Makespan: When we flexibly specify our scheduling policy towards makespan minimization, *Hadar* achieves a shorter makespan by 1.5× compared to Gavel and by 2× compared to Tiresias, as illustrated in Fig. 6. Again, *Hadar* benefits from leveraging the task-level assignment, based on possibly optimal allocation decisions from our online optimization framework, which leads to performance improvement of the makespan over the state-of-the-art baselines.

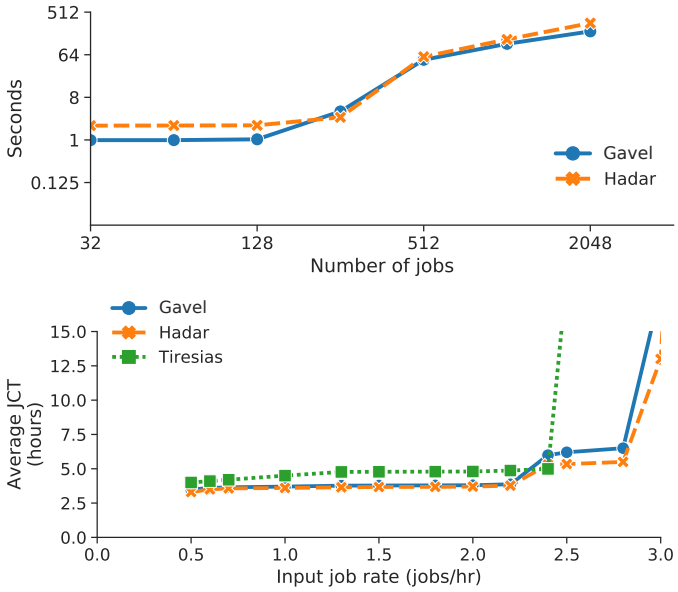


Fig. 7: Scaling of our algorithm compared to Gavel [1] with active jobs in a heterogeneous cluster. The cluster size increases as the number of jobs increases.

5) Scalability: In Fig. 7, we present the running time of our scheduling algorithm to generate decisions, in comparison with Gavel, when the number of jobs increases from 32 to 2048. As observed, the scaling performance of our algorithm is similar to Gavel. Even under the heavy workloads of 2000 jobs, our allocation algorithm can compute and apply allocations in a scheduling round in less than 7 minutes. Our scheduler also has an efficient allocation update policy. Rather than recomputing the allocation in every scheduling round, the scheduler computes the allocation with the new incoming job while the existing jobs in the cluster are still in the running state. If the allocation of the running job changes by computation, the job will be preempted and the new allocation will be in effect. We observe that only 30% of scheduling rounds require a change in allocation for an average job.

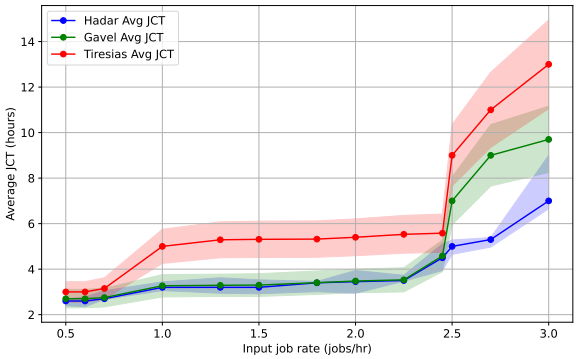


Fig. 8: The average Job Completion Time (JCT) for *Hadar*, Gavel[1], and Tiresias[4], under varying input job rates. The shaded regions denote the range between the minimum and maximum JCT estimates, illustrating the performance resilience and variability under different operational loads.

6) Min-Max Completion Time: In Fig. 8, the comparative analysis of the minimum and maximum job completion times (JCT) for the *Hadar*, Gavel, and Tiresias systems, the focus on the range between these values reveals critical insights into the robustness and reliability of each system under varying workloads. The *Hadar* system demonstrates a relatively tight range of JCT, suggesting consistent performance and a high degree of control over job processing times. This could be indicative of efficient resource management and load balancing within the system. Gavel, while maintaining a comparable average JCT to *Hadar*, exhibits a wider range as the input job rate increases, hinting at potential challenges in sustaining performance when faced with heavier workloads. In contrast, Tiresias shows the largest range in JCT, particularly at higher job rates, signaling a substantial variability that could affect dependability. This might be a result of complex job processing algorithms or less optimized resource allocation strategies. Understanding these ranges is vital for system selection in various operational scenarios, as they provide a window into how each system might behave in the face of real-world uncertainties and fluctuations in demand.

7) Impact of Round length: Fig. 9 presents the average completion time of jobs when scheduled by *Hadar* under different settings: with increasing workloads (x-axis) and with increasing length of a scheduling round (6 minutes to

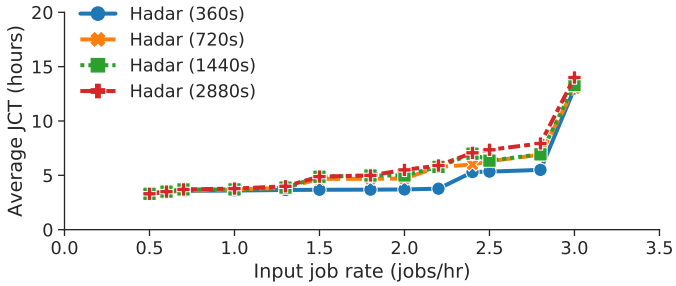


Fig. 9: Impact of round length on the average JCT.

TABLE III: Comparison of performance (JCT and makespan) among schedulers, in both physical and simulated clusters

	Metric	*Hadar*	Gavel [1]	Tiresias [4]
Physical Cluster	JCT	1.99 hrs	4.776 hrs	5.97 hrs
	Makespan	11.29 hrs	21.451 hrs	32.74 hrs
Simulated Cluster	JCT	2.21 hrs	4.97 hrs	5.52 hrs
	Makespan	12.4 hrs	18.6 hrs	24.92 hrs

48 minutes). Using smaller round lengths results in more optimal allocations, but it also incurs higher overhead due to frequent checkpointing. To balance this, a round length of 7 minutes and a checkpoint time of fewer than 6 seconds can provide a steady average JCT as the input job rate (*i.e.* arrival rate) increases. Larger round lengths lead to performance degradation due to both queuing delays for waiting jobs and allocation drifts from optimums with changing cluster states. In our observation, queuing delays contribute to roughly half of the performance degradation.

B. Prototype Experiments

Physical Cluster and Workloads. We further conduct prototype experiments in a cluster consisting of 4 servers and 8 GPUs on AWS. Specifically, the cluster has two `g4dn.xlarge` instances with NVIDIA T4 Tensor Cores, two `g2dn.2xlarge` with NVIDIA GRID K520 GPUs, two `p2.xlarge` with NVIDIA Tesla K80 GPUs, and two `p3.2xlarge` with NVIDIA Tesla V100 GPUs. Each instance has ∼100GB general purpose SSD used for logs and checkpoint files. The maximum read and write throughput is 1000 MiB/s. An open-source high-performance Remote Procedure Call (RPC) framework, `gRPC` is adopted to facilitate the exchange of

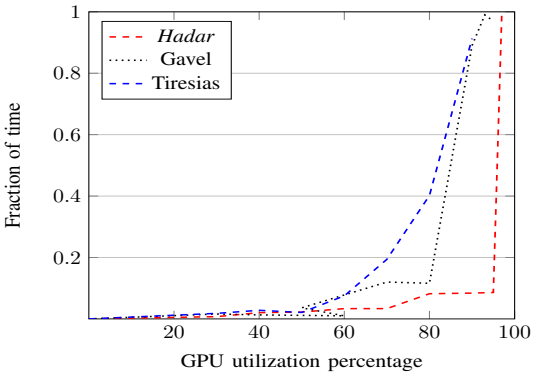


Fig. 10: Comparison of the GPU utilization for the physical cluster.

TABLE IV: Performance overhead of *Hadar*'s preemptive scheduler for various models, with and without resource reallocation, over a 6-minute round duration.

Model	Overhead w reallocation	Overhead w/o reallocation
Resnet-50	2.1%	0.33%
ResNet-18	1.29%	0.21%
LSTM	2.01%	0.87%
CycleGAN	0.68%	0.13%
Transformers	0.71%	0.17%

control messages between the scheduler and workers. Similar to our previous simulation experiments, we use models and datasets in Table II for the prototype experiments as well. One of the large datasets, ImageNet [31] is used by downscaling the data size so that the experiment can be finished within a reasonable amount of time. 100000 images of 200 classes (500 for each class) are included in the dataset. 10 jobs of different models and sizes (GPU demands) from Table II are submitted in the cluster. Each scheduling round is set as 6 minutes long.

Results and Analysis. We evaluate the performance of *Hadar*, with respect to the average job completion time (JCT) and the makespan metrics, in comparison to the baseline schedulers, Gavel and Tiresias. As shown in Table III, *Hadar* improves the average JCT by 2.3× compared to Gave and 3× over Tiresias. Table III also presents the results from another simulation experiment, following the same workload and cluster setting. The results and implications are consistent. Remarkably, the JCT differs within 10% between the simulation and prototype experiments. This further confirms the accuracy and reliability of our simulation experiments. Fig. 10 compares the three schedulers with respect to the resulting cluster-wide GPU utilization in the physical cluster. Again, the results consistently demonstrate superior cluster utilization achieved by *Hadar* over Gavel and Tiresias, due to its capability of allocating heterogeneous GPUs to concurrent tasks of a job in an optimal way and reacting to dynamics in the workload and resource availability.

Table IV presents the performance overhead associated with *Hadar*'s preemptive scheduler when preempting tasks every six minutes (round length). The table contrasts the overhead for different deep learning models without job reallocation (when the computed allocation and current allocation remain the same) and with reallocation. The overhead, mainly influenced by checkpoint loading and saving, varies according to the model size. For instance, the Resnet-50 model has an overhead of 2.1% with reallocation, which is substantially reduced to 0.33% when reallocation is not employed. Similarly, the LSTM model sees an overhead decrease from 2.01% to 0.87% with reallocation.

V. Conclusion

Our paper proposes a novel task-level heterogeneity-aware cluster scheduler called *Hadar*, which aims to optimize several performance metrics such as the average job completion time, fairness, and makespan. To achieve this, the scheduler is formulated into an optimization problem for its solution, utilizing the online primal-dual framework for task-level resource allocation across both temporal and spatial dimensions. We

also undertake the theoretical analysis of the proposed solution to show its polynomial runtime and a long-term performance guarantee in terms of a bounded competitive ratio in job utility, implying approximate optimal solutions within proven constant bounds. Leveraging the dynamic programming structure, our scheduler generates optimal scheduling decisions effectively. Our prototype experiments and trace-driven simulations show that *Hadar* outperforms its state-of-the-art counterparts.

REFERENCES

[1] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *proc. of USENIX Symposium on Operating Systems Design and Implementation*, 2020, pp. 481–498.

[2] Y. Wu, M. Schuster, Z. Chen, Q. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, u. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," in *arXiv preprint arXiv:1609.08144*, 2016.

[3] A. Xu, Z. Liu, Y. Guo, V. Sinha, and R. Akkiraju, "A new chatbot for customer service on social media," in *proc. of Conference on Human Factors in Computing Systems*, 2017, p. 3506–3510.

[4] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A gpu cluster manager for distributed deep learning," in *proc. of USENIX Symposium on Networked Systems Design and Implementation*, 2019, pp. 485–500.

[5] (2006) Amazon ec2 elastic gpus. [Online]. Available: https://aws.amazon.com/

[6] (2008) Gpu-accelerated microsoft azure. [Online]. Available: https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/microsoft-azure/

[7] (2008) Google cloud gpu. [Online]. Available: https://cloud.google.com/gpu/

[8] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *proc. of USENIX Symposium on Operating Systems Design and Implementation*, 2018, pp. 595–610.

[9] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant gpu clusters for dnn training workloads," in *proc. of USENIX Annual Technical Conference*, 2019, pp. 947–960.

[10] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and efficient gpu cluster scheduling," in *proc. of USENIX Symposium on Networked Systems Design and Implementation*, 2020, pp. 289–304.

[11] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *proc. of USENIX Symposium on Operating Systems Design and Implementation*, 2014, pp. 583–598.

[12] C. Trishul, S. Yutaka, A. Johnson, and K. Karthik, "Project adam: Building an efficient and scalable deep learning training system," in *proc. of USENIX Symposium on Operating Systems Design and Implementation*, 2014, pp. 571–582.

[13] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, A. Shenker, and I. Stoica, "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," in *proc. of USENIX Conference on Networked Systems Design and Implementation*, 2011.

[14] I. Gog, M. Schwarzkopf, A. Gleave, R. N. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale," in *proc. of Symposium on Operating Systems Design and Implementation*, 2016, pp. 99–115.

[15] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, "Altruistic scheduling in multi-resource clusters," in *proc. of USENIX symposium on operating systems design and implementation*, 2016, pp. 65–80.

[16] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "Graphene: Packing and dependency-aware scheduling for data-parallel clusters," in *proc. of USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 81–97.

[17] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *proc. of USENIX Symposium on Networked Systems Design and Implementation*, 2011, pp. 295–308.

[18] B. Huang, M. Boehm, Y. Tian, B. Reinwald, S. Tatikonda, and F. R. Reiss, "Resource elasticity for large-scale machine learning," in *proc. of SIGMOD International Conference on Management of Data*, 2015, pp. 137–152.

[19] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "Slaq: quality-driven scheduling for distributed machine learning," in *proc. of Symposium on Cloud Computing*, 2017, pp. 390–404.

[20] P. Thinakaran, J. R. Gunasekaran, B. Sharma, M. T. Kandemir, and C. R. Das, "Phoenix: A constraint-aware scheduler for heterogeneous datacenters," in *proc. of International Conference on Distributed Computing Systems*, 2017, pp. 977–987.

[21] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous gpu clusters," in *proc. of USENIX Symposium on Networked Systems Design and Implementation*, 2022, pp. 945–960.

[22] 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. of The International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[23] S. Wang, O. J. Gonzalez, X. Zhou, T. Williams, B. D. Friedman, M. Havemann, and T. Woo, "An efficient and non-intrusive gpu scheduling framework for deep learning training systems," in *proc. of International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–13.

[24] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, and X. Jin, "Multi-resource interleaving for deep learning training," in *proc. of the ACM Special Interest Group on Data Communication*, 2022, pp. 428–440.

[25] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "Allox: Compute allocation in hybrid clusters," in *proc. of European Conference on Computer Systems*, 2020, pp. 1–16.

[26] S. Chaudhary, R. Ramjee, M. Sivathanu, N. Kwatra, and S. Viswanatha, "Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning," in *proc. of European Conference on Computer Systems*, 2020, pp. 1–16.

[27] Z. Yang, H. Wu, Y. Xu, Y. Wu, H. Zhong, and W. Zhang, "Hydra: Deadline-aware and efficiency-oriented scheduling for deep learning jobs on heterogeneous gpus," *IEEE Transactions on Computers*, 2023.

[28] N. Buchbinder and J. S. Naor, "The design of competitive online algorithms via a primal–dual approach," *Foundations and Trends® in Theoretical Computer Science*, pp. 93–263, 2009.

[29] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *proc. of IEEE Conference on Computer Communications*, 2018, pp. 495–503.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *proc. of IEEE conference on computer vision and pattern recognition*, 2009, pp. 248–255.

[32] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 2012.

[33] (2013) Word-level language modeling rnn. [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. of Conference on Learning Representations*, 2017.

[35] (2020) Pytorch-gan. [Online]. Available: https://github.com/eriklindernoren/PyTorch-GAN#cyclegan

[36] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *proc. of International Conference on Computer Vision*, 2017, pp. 2242–2251.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, 2017.

[38] D. Elliott, S. Frank, K. Sima'an, and L. Specia, "Multi30k: Multilingual english-german image descriptions," in *proc. of the Workshop on Vision and Language*, 2016, pp. 70–74.