Fast and Fair Training for Deep Learning in Heterogeneous GPU Clusters

Zizhao Mo
University of Macau
Macau, China
yc17461@connect.um.edu.mo

Huanle Xu*
University of Macau
Macau, China
huanlexu@um.edu.mo

Wing Cheong Lau
The Chinese University of Hong Kong
Hong Kong, China
wclau@ie.cuhk.edu.hk

Abstract

The GPU device heterogeneity in accelerating deep learning training workloads poses significant challenges for job scheduling in datacenters. Existing heterogeneity-aware job schedulers, however, cannot effectively reduce the overall job completion time (JCT) or provide fairness guarantees due to their coarse-grained resource allocation and poor integration of conflicting objectives.

This paper presents FFT, a novel scheduling system designed for Fast and Fair deep learning Training in heterogeneous GPU clusters. FFT incorporates two key designs. First, it incorporates a resource allocation scheme in each round to enable fine-grained control over resource utilization. Second, it seamlessly integrates a fairness compensation mechanism that dynamically evaluates fairness in real-time. Building upon these designs, FFT formulates a cost minimization problem to determine the optimal schedule, striking a delicate balance between efficiency and fairness. Extensive experiments conducted in physical clusters as well as large-scale testbed demonstrate that FFT can significantly accelerate the overall JCT by up to 5.2× while improving job finish-time-fairness by more than 2.2× compared to state-of-the-art heterogeneity-aware solutions.

CCS Concepts

• **Computer systems organization → Cloud computing**.

Keywords

Heterogeneous GPU cluster, efficiency, fairness

ACM Reference Format:
Zizhao Mo, Huanle Xu, and Wing Cheong Lau. 2025. Fast and Fair Training for Deep Learning in Heterogeneous GPU Clusters. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3721145.3728488

1 Introduction

In recent years, deep learning (DL) has played a pivotal role in driving advancements across various domains, including computer vision, natural language processing, and video processing [18, 43, 50]. The computational demands of DL training necessitate high-performance hardware, and GPUs and TPUs have emerged as the preferred choices due to their superior parallel processing capabilities compared to CPUs.

In response to the strong market demand, GPU manufacturers have been accelerating their efforts to release new hardwares. Consequently, technology giants and research labs have accumulated a diverse array of accelerators boasting varying computation capabilities over time. For instance, the Alibaba machine learning platform offers more than eight types of GPU devices, spanning from older generations to the latest equipment [1]. Recent works have extensively shown that these heterogeneous GPU devices can significantly differ in training throughput across various types of DL workloads [7, 23, 30, 33, 46, 53].

However, this fast GPU technology evolution presents significant challenges to the scheduling of deep learning training workloads. Motivated by the scaling law, AI researchers build larger and larger models over time to break state-of-the-art performance. For instance, the Llama-2 language model, one of the most popular giant language models, consists of 70B parameters [48]. Transferring the training state to new hosts incurs substantial transmission overhead. Additionally, the operation overhead to preempt an ongoing training process, including checkpoint saving and loading as well as communication group initialization, is also non-negligible. To this end, inappropriate preemption operation can nullify the throughput advantages of training on high-end GPUs.

Unfortunately, all existing heterogeneity-aware schedulers overlook the impact of migration overhead in heterogeneous environments. This oversight leads to two main issues: significant wastage of GPU time during training and data preparation stages [33], or a failure to effectively leverage

*Corresponding author

ICS '25, Salt Lake City, UT, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3728488

migration for accelerated training on high-end devices [23]. We have identified the root causes of these inefficiencies - these schedulers only characterize resource allocation in a coarse-grained manner based on effective throughput. By relying solely on throughput-based allocation, which represents average allocation over the long term, these schedulers fail to accurately capture the scheduling order among jobs and lack the necessary expressiveness to account for migration overhead. Consequently, the scheduling efficiency is significantly compromised, often leading to JCT performance degradation by over two times in many cases.

Furthermore, existing DL training schedulers that aim to minimize JCT in heterogeneous clusters [23, 33] often struggle to incorporate another vital metric at the same time - job fairness, which is crucial for incentivizing resource sharing among users in large clusters [13, 27]. As jobs arrive in the DL cluster continuously over time [32, 40], schedulers that focus on fast completion [23] tend to prioritize jobs with fewer GPU requirements and training iterations. This may lead to the neglect of giant training jobs [41, 49] requiring substantial GPU quotas, resulting in unfair resource allocation. Conversely, schedulers that focus on fairness often lead to significant JCT degradation because of poor integration between fairness guarantees and JCT optimization [7]. This highlights the challenge of finding the optimal balance between these conflicting objectives in heterogeneous clusters.

In this paper, we introduce FFT, a new system designed for Fast and Fair Training of distributed DL workloads in heterogeneous GPU clusters. FFT encompasses two key goals: 1) leveraging fine-grained resource allocation and facilitate opportunistic migration to maximize the utilization of heterogeneous resources for JCT reduction, 2) maintaining a dedicated balance between reducing JCT and ensuring long-term job fairness. To achieve these goals, FFT incorporates two essential designs. First, FFT leverages a per-round resource allocation vector that captures the dynamic nature of the cluster, accounting for churn events of jobs and machine changes, while mitigating the problem of excessive switching caused by overly frequent allocation changes. Second, FFT integrates seamless fairness compensation by promptly identifying instances where a job's allocation deviates significantly from the most fair scheduler. On top of these designs, FFT tackles a cost minimization problem to identify the globally optimal schedule with minimal overhead. Moreover, FFT employs a hierarchical mechanism to strategically determine the placement of DL workloads across various hosts, effectively mitigating communication overhead.

We implement a prototype of FFT on top of Kubernetes, providing support for diverse DL training jobs. FFT delivers seamless and efficient profiling and switching functionality, ensuring a nearly non-intrusive experience for users. Our extensive evaluations of FFT cover both a physical cluster and a large-scale simulation testbed, involving comprehensive DL training workloads derived from enterprise traces [32]. In summary, we make the following contributions:

- We identify the significance of fine-grained resource allocation in optimizing JCT. With our novel allocation scheme, opportunistic migration can be achieved, allowing for maximum utilization of heterogeneous accelerators.
- We make a delicate balance between reducing JCT and ensuring long-term fairness, by integrating seamless fairness compensation into a global optimization objective. By our design, the fairness control is transparent to users.
- We implement a scalable system to support our design, and experimental results highlight that our system outperforms state-of-the-art schedulers by up to 5.2× in terms of JCT and nearly 2.2× in fairness.

2 Background and Motivation

2.1 Scheduling Co-located DL Jobs

Cluster-level scheduling optimizes JCT through dynamic resource orchestration among co-located jobs, by strategically reallocating accelerators (e.g., prioritizing shorter jobs via GPU redistribution from long-running workloads) to combat JCT inflation and alleviate high-end GPU queueing bottlenecks [40]. To maximize systemic efficiency, heterogeneity-aware schedulers must (1) mitigate contention-driven JCT inflation by aligning accelerator capabilities (e.g., GPU generations, memory bandwidth) with the specific workload demands of co-located jobs [19], and (2) ensure resource-efficient GPU time utilization across co-located DL jobs in real time. This necessitates intelligent coordination across heterogeneous devices to balance workload requirements of various DL jobs with hardware capacities.

2.2 Opportunistic Migration

Previous studies [7, 23, 33, 46] extensively showcase the significantly superior performance of high-end GPUs for DL jobs. As illustrated in Fig. 1(a), we observe varying acceleration effects ranging from 1.6× to 2.71×. To hasten the training process for earlier completion, it is desirable for schedulers to migrate DL jobs to high-end workers whenever possible. However, this migration can introduce substantial overhead due to the following characteristics:

• *Large-volume training states.* Resuming DL training jobs on a new host necessitates the availability of previously updated parameters. Large language models, in particular, experience substantial parameter growth. Compounding this, modern optimizers like Adam [22] expedite the gradient descent process by incorporating additional information, such as exponentially weighted averages and exponential moving averages. Moreover, if the gradient accumulation technique is used, the gradients also need to be stored. Consequently,

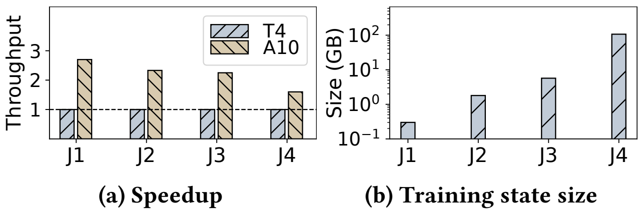


Figure 1: Benefits and overhead of job migration across diverse DL workloads over heterogeneous devices. J_1 (ResNet) and J_2 (VGG) were trained on the Imagenet-1k and CiFAR-10 datasets. Additionally, J_3 (GPT-neo-350M) and J_4 (OPT-6.7B) were trained on the Wikitext-103 and pile-law datasets.

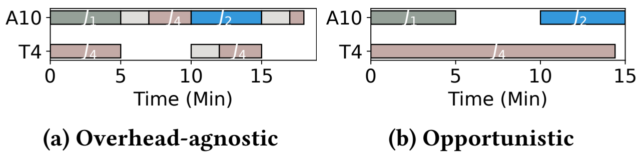


Figure 2: Illustration of opportunistic migration. Jobs J_1, J_2, and J_4 submitted at time 0, 10, and 0 respectively. Migrating J_4 costs two minutes. J_4 lasts for 14.4 minutes in T4 GPU while the execution on each GPU follows the SRTF (i.e., shortest-remaining-time-first) order. Grey spaces denote the time wasted by migration.

the training state of a DL training job can become significantly large. As depicted in Fig. 1(b), the J_4 with gradient accumulation and Adam techniques enabled generates state files as large as 107GB.

• *Extra operation overhead.* On the other hand, the operation overhead of stopping and restarting training process is non-negligible. Specifically, writing and loading checkpoint files are critical components of the migration operation, as training must always resume from the latest state. Furthermore, when training processes are restarted elsewhere, user containers must be restarted, and the NCCL communication group for multi-GPU training must be reestablished, inevitably wasting valuable GPU time.

The non-negligible migration overhead can impede the performance gains on high-end GPUs. In Fig. 2, we construct an example to prove the importance of taking the migration overhead into scheduling decision. We have three jobs in the cluster, i.e., J_1, J_2, and J_4, where the throughput gap among different GPUs and the training state sizes abide by the data listed in Fig. 1. The training time of J_1 and J_2 on A10 are 5 minutes, while it is 14.4 minutes for J_4 on T4. Under the overhead-agnostic approach (Fig. 2(a)) to schedule training jobs like SRTF policy, i.e., shortest-remaining-time-first, the J_4 is frequently migrated once the cluster condition changes, e.g., short jobs enter and complete in the cluster (i.e., events at time 0, 5, and 10), leading to significant GPU time waste. Moreover, the JCT for J_4 is improved by 25%. In extreme case where the length of J_4 is prolonged and the new jobs with short jobs continuously enter the cluster, nearly 40% of its training time will be wasted on migration. A more effective policy (Fig. 2(b)) is to continue the training of J_4 at time 5 on T4 with the knowledge of its training time and the speedup on high-end GPUs, avoiding severe migration overhead meticulously. We refer it as *opportunistic migration.*

Table 1: Throughput and allocation matrix of jobs where each allocation rate denotes the proportion of time the job spends on the corresponding GPU

	Throughput (epochs/min)		Allocation rate	
	GPU$_0$	GPU$_1$	GPU$_0$	GPU$_1$
I_0	8	1	m_0	m_1
I_1	8	1	n_0	n_1

2.3 Limitations of Current Schedulers in Optimizing JCT

Despite the existence of various heterogeneous DL schedulers, such as Allox and Gavel [23, 33] that are designed to minimize JCT, we found that they encounter specific shortcomings and challenges in achieving effective performance when dealing with switching overhead in heterogeneous environments.

• *Disabling switching can significantly prolong JCT.* AlloX adopts a direct approach by relying solely on execution time and completely disabling job switching during scheduling. When new jobs arrive, AlloX assigns them to the tail of queue with shortest waiting time, and the training process cannot be interrupted. This solution results in significantly degraded efficiency due to the head-of-the-line (HOL) blocking problem, where smaller jobs experience prolonged waiting times [38]. Additionally, the absence of job switching capability prevents many jobs from leveraging the acceleration offered by higher-end GPUs, resulting in resource waste and degraded JCT.

• *Throughput-based allocation helps little to reduce JCT.* Gavel determines a schedule based on effective throughput. However, it is worth noting that throughput-based optimization does not yield optimal JCT performance. To illustrate, we construct a simple scenario in Table 1 where two identical jobs arrive at time zero and require 16 epochs to complete.

Gavel finds an allocation matrix $(m_0, m_1; n_0, n_1)$ where m_k and n_k represent the fraction of time that I_0 and I_1 is assigned to the k-th GPU, respectively. The effective throughput is $(8m_0+m_1)$ for I_0 and $(8n_0+n_1)$ for I_1. To minimize the overall JCT, the optimization problem becomes:

$$\min \frac{16}{8m_0+m_1} + \frac{16}{8n_0+n_1} \geq \frac{\left(\sqrt{16}+\sqrt{16}\right)^2}{8(m_0+n_0)+(m_1+n_1)} \geq \frac{64}{9},$$

where the inequality is due to Cauchy-Schwarz Inequality, and $m_0 + n_0 \leq 1$, $m_1 + n_1 \leq 1$. The equality is attained when $m_0 = m_1 = n_0 = n_1 = 0.5$, which is equivalent to

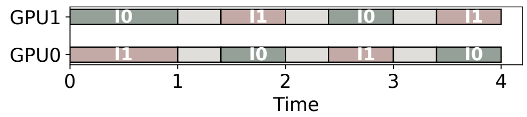


Figure 3: Coarse-grained throughput-based allocation involves approximation within multiple rounds, leading to frequent migrations that are colored in grey.

fair allocation. By contrast, an optimal policy is to assign I_0 to GPU$_0$ and I_1 to GPU$_1$ first, and migrate I_1 to GPU$_0$ after I_0 finishes. This results in an overall JCT of 4 + (16-2)/8 = 5.75 minutes, which is 20% less than that under Gavel. If we proceed to increase the number of jobs, we can demonstrate that this gap can be up to twice as large by repeating this analysis.

When the objective of Gavel is to minimize the makespan, i.e., the maximum JCT among jobs, the optimization formulation will change as follows:

$$\min\max\left\{\frac{16}{8m_0+m_1},\frac{16}{8n_0+n_1}\right\}\geq\frac{1}{2}\left(\frac{16}{8m_0+m_1}+\frac{16}{8n_0+n_1}\right).$$

This results in the same allocation policy as above. As such, The optimization framework of Gavel, which adopts effective throughput, is more relevant to job fairness than JCT minimization.

Furthermore, Gavel triggers frequent migrations in response to an allocation matrix characterized by numerous fractional values, aiming to synchronize job scheduling with the allocation outcome. To illustrate, consider the scheduling scenario delineated in Table 1, where migration accounts for 40% of the time in each scheduling round for both jobs. With $m_0 = m_1 = n_0 = n_1 = 0.5$, each job alternates their training on both GPU types within each scheduling round, as visually depicted in Figure 3, resulting in a utilization as low as 60%.

***Takeaway I*: Coarse-grained allocation significantly degrades scheduling efficiency**. Upon examination, Gavel and Allox are impeded by their coarse-grained allocation strategies. Specifically, they either disable switching altogether or rely on throughput-based allocation, which contributes minimally to reducing JCT and fails to effectively coordinate migration benefits and overhead among jobs. Consequently, the scheduling efficiency is compromised, resulting in suboptimal performance.

2.4 Limitations of Current Schedulers in Optimizing Fairness

Ensuring fairness is crucial to prevent starvation and mitigate the imbalance of benefits among users, thereby promoting resource sharing within the cluster. In the domain of DL scheduling, Finish Time Fairness stands out as a vital criterion for long-term job-level fairness [10]. It quantifies the fairness level of jobs by comparing JCT in a shared cluster with that in an exclusive cluster where each job has access to $1/N$ of the available resources, with N being the number of DL jobs. However, existing fair schedulers including Gavel and Gandiva$_{\text{fair}}$ [7] still perform unsatisfactorily within heterogeneous clusters.

• *Throughput-based allocation fails to guarantee finish time fairness in dynamic environments.* Gavel also includes a fairness-oriented policy based on training throughput. Specifically, it establishes a fairness target for allocation matrix and employs round-based schedules to align with this target. However, this policy assumes that the allocation matrix across heterogeneous devices undergoes minimal changes, enabling it to progressively allocate permissions over time and approximate the fractional share for each job. Unfortunately, in dynamic environments with frequent job arrivals, new jobs are prioritized over older ones in scheduling. As a result, the allocated share for older jobs cannot be preserved, and in extreme cases, they may not be scheduled at all.

• *Poor integration degrades JCT.* In addition to fairness violations, current fair schedulers also result in degraded JCT performance. Specifically, Gandiva$_{\text{fair}}$ allocates heterogeneous GPU resources equally to tenants based on max-min fairness and then focuses on maximizing overall training throughput through greedy trading among users. Unfortunately, these optimizations ignore essential job resource demand including requested workers and job length, leading to substantial JCT degradation consequently.

***Takeaway II*: Balancing JCT reduction and fairness is challenging**. While it is important to prioritize both JCT reduction and fairness, an excessive focus on fairness alone can negatively impact JCT performance. This is because there are inherent conflicts between these two objectives [14], making it essential to achieve an optimal balance. However, striking this delicate balance poses significant challenges in the presence of highly dynamic job submission and heterogeneous environments.

2.5 Our Goal

Based on the analysis conducted above, it is clear that heterogeneity presents substantial difficulties in efficiently scheduling DL workloads, with existing solutions falling short in minimizing JCT or ensuring fairness. In order to overcome these limitations, we aim to design a new heterogeneity-ware scheduler for DL workloads with two primary goals:

- The scheduler needs to capture resource allocation in a more fine-grained manner, aiming to enhance resource efficiency and facilitate opportunistic migration. These improvements are pivotal for reducing JCT and improving fairness performance.
- The scheduler should ensure a long-term fairness guarantee among DL jobs in highly dynamic environments, necessitating a systematic and flexible combination of fairness and JCT reduction.

3 FFT System Overview

To achieve our design goals, we propose FFT, a new system that dynamically distributes DL training workloads across heterogeneous accelerators. In this section, we provide a high-level overview of the FFT system to highlight its key design ideas and architectural features.

3.1 Key Ideas

Incorporating per-round optimization vector. In pursuit of fine granularity, FFT integrates a per-round resource allocation vector as the sole optimization variable, capturing the instantaneous scheduling of heterogeneous GPU devices for all jobs within a cluster. This approach stands in stark contrast to the throughput-based allocation in Gavel [33]. The inclusion of this vector enables FFT to dynamically quantify the training progress of each job and assess migration overhead in real-time. As a result, it facilitates global optimization across all DL training jobs and addresses the issue of excessive switching caused by frequent allocation changes. Additionally, this vector empowers FFT to accurately capture the dynamic nature of the cluster, encompassing events such as job churn and machine changes.

Seamless Fairness compensation. Long-term fairness is quantified by cumulative workload allocations relative to an equal-share policy—a baseline where cluster resources are evenly partitioned across all jobs. FFT ensures fairness by dynamically benchmarking its GPU allocations against a fairness-optimized reference scheduler, tracking and quantifying deviations from this policy over time. This approach enables FFT to promptly identify instances in each round where a job's allocation significantly deviates from what is considered fair. In such cases, FFT can make appropriate compensations during the relevant rounds, while still ensuring the overall JCT optimization.

3.2 Architecture & Workflow

Fig. 4 illustrates an architecture overview of FFT, along with its scheduling workflow. FFT is a comprehensive system that takes control of distributed DL training jobs throughout their entire process. Upon submission of jobs to the cluster using the API ❶, FFT automatically initiates the profiling process and records key information such as the required number of workers, the required training iterations, and the specified dataset. We store all above information in the Recorder.

Since it is impractical for users to provide throughput information for all types of accelerators and specify switching overhead in clusters, the profiler module ❷ in FFT profiles the runtime throughput and the size of the training state for jobs on the fly. Armed with the above information and the historical execution status, the FFT scheduler ❸ then outputs a preliminary GPU type-level result for the incoming round.

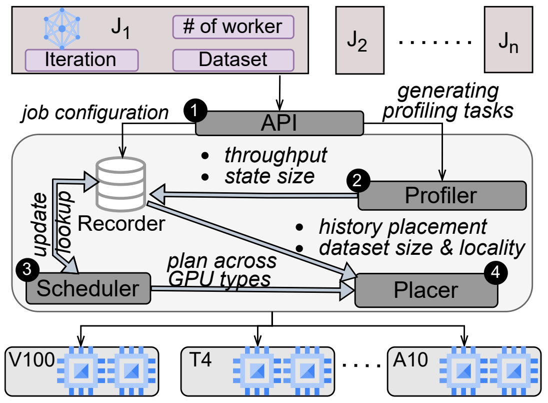


Figure 4: The system architecture of FFT.

This output strives to coordinate jobs for better utilization of heterogeneous GPUs in clusters. To guarantee fairness in the long term for each job, FFT traces the execution of jobs round by round, which enables FFT to prioritize both the starved and small jobs simultaneously. Subsequently, the placer module ❹ offers a more refined placement scheme for DL training jobs at the host level, based on the history of placement schemes and dataset information such as size and locality. This optimization minimizes futile preemption and avoids unreasonable waste of time spent on data transfer.

4 FFT System Design

In this section, we will delve into the design details of the key components in FFT.

4.1 Fine-grained Allocation Vector

We first formally define the heterogeneous DL scheduling problem as follows. The cluster contains M different types of accelerators, each having Y_i GPU workers. DL job j arrives at the cluster at time a_j. Upon the arrival of job j, the scheduler knows the requested number of workers d_j and the number of epochs, denoted as W_j, required for job j to execute[1], Furthermore, the scheduler is informed of the throughput θ_j^i after profiling. The throughput represents the number of epochs that can be completed on GPU type i during each scheduling round. Consequently, the completion time of job j on device of type i, without any preemption, can be estimated as W_j/θ_j^i.

The scheduler incorporates a fine-grained resource allocation vector for each job j to capture a more specific per-round allocation across heterogeneous devices, $\vec{x_j}(t) = \{x_j^i(t)\}$. Here, $x_j^i(t) \in \{0, 1\}$ is a binary variable that represents if job j is executed on type i device during round t. When considering the scheduling at a given time, only the jobs

[1]The FFT scheduler does not depend on precise workload estimations and exhibit resilience to significant estimation errors as shown in Fig. 13.

that have not yet completed by that time t, i.e., $A(t)$, the resource allocation vector $\vec{x}_j(t)$ must satisfy the following constraints:

$$\sum_{i=1}^{M} x_j^i(t) \le 1, \forall j, t; \sum_{j \in A(t)} d_j x_j^i(t) \le Y_i, \forall i, t. \qquad (1)$$

These constraints state that job j can only be executed on one device type and each device type cannot be oversubscribed. In addition to this, the schedule generated by $\vec{x}_j(t)$ must be work-conserving, i.e., always keeping the GPU devices busy if there are jobs applicable to be scheduled. The scheduler determines $\vec{x}_j(t)$ for each job j at the beginning of each round with the objective to minimize the overall JCT and ensure fairness among users.

4.2 FFT Scheduler

The fine-grained resource allocation vector $\vec{x}_j(t)$ makes it possible for us to evaluate JCT, fairness, and migration cost on-the-fly. Due to this, the scheduler efficiently combines all these factors together within one single objective to design policies that can well balance the trade-off between JCT and long-term fairness.

4.2.1 Minimizing JCT.. The job completion time C_j is defined as the earliest time that all W_j epochs of job j are executed: $C_j = \min \left\{ L \in \mathbf{N} : \sum_{t=a_j}^{L} \sum_{i=1}^{M} x_j^i(t)\theta_j^i \ge W_j \right\} - a_j$. However, this formulation is in general intractable since it is in the form of a combinatorial set. In the scheduling literature [3, 20, 26], a standard linear approximation is provided for C_j to analyze the theoretical performance of widely adopted schedulers SRPT and Round-Robin, more specifically, we have:

$$C_j \approx \sum_{t=a_j}^{\infty} (t - a_j) \cdot \sum_{i=1}^{M} x_j^i(t)\theta_j^i / W_j. \qquad (2)$$

The approximation can be interpreted as a fractional completion time where $\sum_{i=1}^{M} x_j^i(t)\theta_j^i/W_j$ fraction of job j spends $(t - a_j)$ rounds before it finishes. As such, this approximation serves as a lower bound of the true JCT.

Directly minimizing the overall JCT using the above approximation is computationally inefficient, as it requires optimizing over a resource allocation space with dimensions of $M \cdot N \cdot T$ where N represents the total number of DL jobs submitted to the cluster and T indicates the total number of rounds for scheduling. When jobs keep arriving, T can go to infinity and the optimization problem becomes intractable. To provide a scalable solution that can be practically used, the scheduler chooses to optimize job scheduling in each round based on the fraction $(t - a_j) \cdot \sum_{i=1}^{M} x_j^i(t)\theta_j^i/W_j$. However, simply solving this per-round minimization problem results in a poor schedule that always gives scheduling priority to jobs with large completion time W_j/θ_j^i. To avoid such

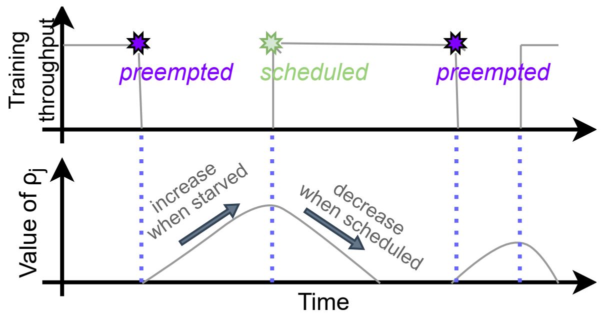


Figure 5: An illustrative example of fairness compensation mechanism under FFT using a specific training job. The value of ρ_j serves as an indicator of the GPU resources allocated to the job, increasing continuously during periods of starvation and decreasing when the desired resources are obtained. As ρ_j grows, so do the job's chances of being scheduled.

situations, the scheduler also incorporates the volume of job demand, i.e., the number of GPUs required multiplied by the total processing time of job j on type i, into the objective:

$$\min \sum_{j \in A(t)} \sum_{i=1}^{M} \Big((t - a_j)\theta_j^i / W_j + \big(d_j W_j / \theta_j^i\big) \Big) \cdot x_j^i(t). \qquad (3)$$

This formulation forces the scheduler to prioritize jobs with shorter processing time (i.e., W_j/θ_j^i) and fewer GPUs resources (i.e., d_j), making the scheduler behaves similarly to SRSF (shortest remaining-service-time first).

4.2.2 Incorporating fairness. Minimizing the overall JCT alone can sometimes lead to unfairness, where larger jobs are starved for a significant duration under SRSF-like schedulers. To incorporate fairness in job scheduling, we introduce a compensation factor $\rho_j(t)\theta_j^i$ to the JCT objective in Eq. (3), resulting in a balanced optimization problem:

$$\min \sum_{j \in A(t)} \sum_{i=1}^{M} \Big((t - a_j)\theta_j^i / W_j + \big(d_j W_j / \theta_j^i\big) - \rho_j(t)\theta_j^i \Big) \cdot x_j^i(t). \qquad (4)$$

The purpose of the compensation factor is to compensate job j when its allocation is far way from its fair share. Specifically, $\rho_j(t)$ captures the amount by which job j falls behind with respect to what it would have received under a max-min fair scheduler up until time t since its arrival. If $\rho_j(t)$ is large, the scheduler prioritizes scheduling job j on a higher-end type i to achieve a greater throughput θ_j^i in subsequent scheduling rounds. We exemplify the variantion of $\rho_j(t)$ across different conditions in Fig. 5. To ensure easy updates on-the-fly and maintain positivity, the scheduler recursively designs $\rho_j(t)$ as shown below:

$$\rho_j(t+1) = \max \left\{ 0,\ \rho_j(t)\ + \mu_j(t) \cdot \left(\frac{W_j}{\tau_j - a_j} - \sum_{i=1}^{M} \theta_j^i x_j^i(t) \right) \right\}. \qquad (5)$$

When job j enters the cluster, no compensation is given initially, resulting in $\rho_j(a_j) = 0$.

The interpretation of this design is as follows: τ_j represents the time at which job j would complete under the max-min fair scheduler. Consequently, $W_j/(\tau_j - a_j)$ quantifies the average workload that job j should complete in each round under the fair scheduler. As FFT completes $\sum_{i=1}^{\mathsf{M}} \theta_j^i x_j^i(t)$ workload in round t, the amount of workload FFT falls behind the progress supposed to be achieved under a fair scheduler in round t can be computed as as $W_j/(\tau_j - a_j) - \sum_{i=1}^{\mathsf{M}} \theta_j^i x_j^i(t)$. Moreover, the scheduler dynamically updates the estimation of τ_j for all j when a job arrives or completes. Specifically, it estimates τ_j as the completion time of job j, when it is trained in a dedicated cluster with $1/|A(t)|$ of all types of resources. [2]

In the design, $\mu_j(t)$ serves a fairness coefficient that regulates the update of $\rho_j(t)$, aiming to achieve a dedicated balance between JCT and fairness. It is important to note that if the fairness coefficient remains constant over time, larger jobs still have limited opportunities for scheduling when continuously incoming jobs with higher scheduling priority are present. To this end, the scheduler adopts a dynamic fairness coefficient that increases with the duration a job has stayed in the cluster. Specifically, $\mu_j(t)$ is defined as $(t - a_j)\big/(\tau_j - a_j)$. When a job has been starved for a long time, $\mu_j(t)$ can become quite large due to its increasing trend over time. This grants job j more chances for scheduling, thereby ensuring better long-term fairness guarantees.

4.2.3 Switching control.

To facilitate opportunistic migration, the scheduler integrates a meticulously crafted penalty term, denoted as $s_j^i(t)$. The design principle aims at discouraging over-frequent switching, which leads to substantial migration overhead between hosts of diverse device type.

The penalty factor $s_j^i(t)$ is determined by the expected training state transfer time. Specifically, if the training of job j at time t is going to be conducted on a type i different from the type at time $t-1$, the scheduler assigns it as the value of transfer duration, as migration is inevitable in this situation. However, if job j is executed consecutively on the same GPU type during time slots t and $t-1$, we hold an optimistic assumption that the job can be placed on the same host at time t and no switching overhead is incurred. This assumption is maintained by utilizing the placement strategy explained in section 4.3.

4.2.4 Scheduling based on cost minimization.

Combining all the design components, the scheduler formulates a cost minimization problem as follows:

$$\min \sum_{j\in A(t)} \sum_{i=1}^{\mathsf{M}} \left(\varphi_j^i(t) + s_j^i(t) - \rho_j(t)\theta_j^i\right) \cdot x_j^i(t), \tag{6}$$

along with a work-conserving constraint:

$$\sum_{i=1}^{\mathsf{M}} \sum_{j\in A(t)} d_j \cdot x_j^i(t) \geq \mathsf{K}(t), \forall t. \tag{7}$$

In the objective, the scheduling cost is represented by the sum of three terms and and is associated with a pair of a job and device type. The first term $\varphi_j^i(t) = (t - a_j)\theta_j^i/W_j + (d_j W_j/\theta_j^i)$ is used to optimize JCT, the second term $s_j^i(t)$ is to control switching, and the third term $\rho_j(t)\theta_j^i$ is for ensuring fairness. The constraint reflects the goal of utilizing the GPUs in the clusters as much as possible. The quantity $\mathsf{K}(t)$ represents the minimum value between the number of available GPUs and the maximum GPU demand of the available jobs, considering the per-type resource constraint at time t.

This optimization problem can be formulated as an integer linear optimization problem. Since the variables are related to the number of GPU types rather than the number of GPUs, this scheduling problem can be efficiently solved even in a large-scale cluster by leveraging standard optimization solvers without incurring significant overhead. As depicted in Fig. 13, FFT can solve this problem, which involves 4000 jobs, in just two seconds.

4.2.5 Theoretical foundations.

As many optimization problems with constraints, e.g., $\min_x f(x)$, s.t., $g(x) \leq 0$, the scheduling algorithm can be interpreted as the first step of dual approach [4], i.e., $\min_x f(x) + \lambda g(x)$ where λ is the Lagrangian multiplier. Specifically, $f(x)$ corresponds to the JCT objective in Eq. (3) and $g(x)$ corresponds to the fairness term that measures the difference between the completed workload in each round under fair scheduler and that under FFT's scheduler. More interestingly, the compensation factor $\rho_j(t)$ can be viewed as the multiplier λ whose update takes a recursive form similar to Eq. (5). Similarly, we can interpret $\mu_j(t)$ as the learning rate in the dual update. When $f(x)$ is a time-varying function, recent advancements have extended the dual approach to deal with online optimization problems [8, 28, 57]. By applying the analytical techniques developed in these studies, we can substantiate the optimality of both efficiency and fairness achieved by FFT, as corroborated by the ensuing theorems.

THEOREM 4.1. *The number of unfinished jobs under FFT is at most $O\left(\sqrt{N}\right)$, and the achieved fractional JCT under FFT is upper bounded by a constant factor times the optimal JCT.*

THEOREM 4.2. *Under FFT, each job has been executed in at least one round within a constant time from its arrival.*

[2]Computing the completion time for jobs with different number of workers in heterogeneous DL clusters is an open problem, we resort to an approximation, i.e., evaluating the τ_j on the condition that preemption is disabled.

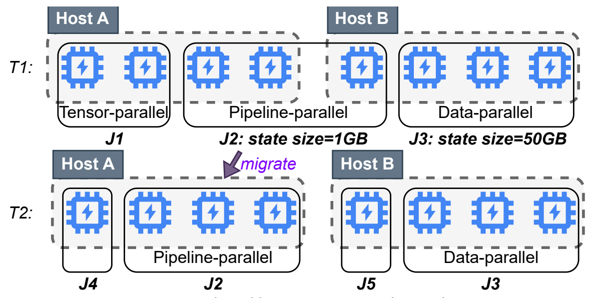


Figure 6: An example illustrating the placement mechanism in FFT. At time T_1, intra-host GPU workers are prioritized for jobs with data- and tensor-parallelism, specifically J_1 and J_3. Later, at time T_2, job J_2 is selected for migration due to a churn event, as it incurs minimal migration overhead and is likely to benefit the most from the migration.

The first theorem establishes that the schedule generated by FFT ensures that the majority of jobs are completed before they would finish under the optimal schedule. Additionally, the overall fractional JCT achieved by FFT's schedule is highly close to the actual JCT attained under the optimal schedule. This indicates that FFT is near-optimal for minimizing JCT. The second result guarantees job fairness by ensuring that a job is not left waiting for an extended period of time after its arrival. This is particularly significant considering that existing JCT-optimized schedulers often neglect larger jobs, causing them to be delayed excessively.

We prove these two theorems by utilizing the widely recognized Lyapunov-drift analysis [35, 57]. The pivotal step involves quantifying the drift of the compensation factor $\rho_j(t)$. By substituting this drift into the objective function in Eq. (6), we can compare the differences between the max-min fair schedule and FFT's schedule. Through this process, we can establish an upper bound for $\rho_j(t)$, which directly reflects the amount of workload completed under FFT. Based on this bound, we compare FFT with an optimal schedule to evaluate its overall JCT.

4.3 Optimizing DL Workload Placement

In addition to GPU resource allocation, job placement also plays a critical role in improving training throughput. The first issue related to placement is that transferring the training states and datasets of DL training jobs can incur heavy switching overhead. While the FFT scheduler can provide jobs with accelerator GPU type-level migration control, jobs within each GPU type still compete for workers, aiming to minimize switching overhead from their individual perspectives. The second issue arises from the advantage of packing workers of DL training jobs onto as few hosts as possible to minimize cross-network communication overhead [15, 19, 24, 39, 58], as the bandwidth of LAN is typically lower than intra-host interconnections such as PCIe and NVLink. In the current landscape, the significance of network packing is increasing, driven by the exponential growth in parameter size [34]. This growth necessitates a larger volume of data synchronization, particularly under hybrid communication patterns.

Simultaneously optimizing the two aforementioned issues in a large-scale cluster is generally intractable. Therefore, FFT introduces a straightforward yet efficient placement heuristic, employing a hierarchical strategy among jobs allocated to each GPU type. It takes into account both parallelism and migration overhead successively, as illustrated in Fig. 6. Given the long training process, FFT first strives to reduce communication overhead existed in each training iteration. Specifically, it prioritizes packing intra-operator DL training jobs [59] that involve data/tensor parallelism into as few hosts as possible. This limits intensive inter-worker collective communication (i.e., All-Reduce operation) within high-speed intra-host connections and prevents inter-host networks from heavy contention. In contrast, inter-operator jobs that only leverage pipeline parallelism, are less susceptible to network contention due to their point-to-point communication pattern and small scale of intermediate results. In this sense, we can minimize the training throughput degradation in the cluster level. In addition, at each churn event, FFT strives to pack training jobs requiring multiple GPU workers into a single host if possible for enhancing the overall training throughput, where the priority is determined by their migration overhead in an ascending order.

4.4 General Settings

4.4.1 Compatibility to large models. Large DL models, particularly popular large language models (LLMs) to date [2, 42, 48, 49], demand a substantial GPU memory capacity, rendering them unsuitable for training on low-end GPUs with limited memory. Fortunately, FFT offers supports to limit the GPU type selection algorithmically. To be specific, FFT employs placement constraints that explicitly restrict certain GPU types assigned to a specific job by the equation $x_j^i(t) = 0$, if their GPU memory is not enough to host the memory usage in the training process, including activations, gradients, model parameters, and optimization states, etc.

4.4.2 Dataset fetching. To ensure that DL training jobs can be executed properly in a local storage-based cluster, FFT transfers the associated datasets from the repository to the host before initiating the training process if they do not already exist there, enabling the overlapping of I/O with computational tasks. In contrast, when dealing with jobs that have their training datasets stored in high-bandwidth NFS-based environments, the network overhead of dataset transfer can be significantly reduced by pre-fetching samples during the computation phase [9, 36].

Table 2: The representative DL training workloads used in experiments

model	variants	dataset	workers
VGG	16,19	ImageNet CIFAR wikiart	1,2,3,4
ResNet	50,152		1,2,3,4
DenseNet	121,201		1,2,3,4
Bark	normal,small	librispeech	1
Bert	base,large	Wikipedia	1
GPT-neo	1.3B,2.7B	pile	4
GPT-2	medium,large		4,8
OPT	2.7B,6.7B		4,8

4.4.3 Awareness of training stall. In distributed DL training, it is observed that jobs spanning multiple GPU types often experience reduced throughput due to training stalls in lower-end workers [9]. In data-parallelism, while tuning the training configuration parameters such as *batch size* and *learning rate* across different workers within the same job can help mitigate this stall, it also leads to significant performance loss [39]. Moreover, pipeline parallelism also faces significant bubble effects in the presence of straggler GPUs, which can only be balanced via adjusting the model layers in each stage. Worse still, these solutions require framework-specific adaptation to amortize the performance gap within the GPU workers. To offer a uniform scheduling framework for training jobs of various parallisms, FFT avoids this dilemma via enforcing that only a single type of GPUs can be allocated to each job at the same time.

5 Implementation

FFT is built on top of Kubernetes, providing users with a platform for training various models using PyTorch and different training paradigms, including data parallelism and model parallelism. Control messages are exchanged between modules through gRPC, while training state files among training jobs are transferred over the network using rSync.

Submission API. Users submit DL training jobs to FFT via a command-line API that is registered as a Python entry point. Users only specify the training script, the requested worker number, and the number of training iterations in API. Once the API is involved, it submits user-specified information to the Job Recorder. After that, it generates and distributes profiling tasks to different hosts.

Job recorder. The job recorder is developed on Redis, an efficient Key-Value store, to serve the update requests and offer fast lookup for the job information. The data is encoded as $\{\ job\_id, attributes\}$.

Profiler. In FFT, profiling tasks are generated for each job to evaluate the training throughput and the file size of training states. Notably, different from elastic training setting [15, 19, 24, 46], we only profile job runtime using the user-specified worker number. These profiling tasks have top scheduling priority as they are crucial for making accurate scheduling decisions based on the runtime configurations of jobs. Consequently, ongoing training jobs are temporarily suspended to allow the profiling tasks to be executed. Once the profiling tasks are completed, the suspended training jobs resume. We replace the default dataloader in user code by our customized one, to minimize the effort for profiling. Our customized dataloader supplies only a limited number of data batches to the training loop in the profiling mode, while functioning normally in the training mode. To minimize the impact of profiling on the cluster, we kick off profiling tasks for jobs on GPUs with greatest memory first. As such, we can measure the memory consumption and omit the profiling process on device types with insufficient memory.

Scheduler. In each scheduling round, the scheduler leverages the cvxpy solver to generate an allocation that satisfies the basic constraints. To accomplish this, it formulates the objective using a variable matrix and a parameter matrix of size $\mathsf{M} \times A(t)$. To evaluate parameter matrix and formulate the necessary constraints, the scheduler queries job information from the job recorder. After making scheduling decisions, τ_j values are updated accordingly.

Placer. The placer component utilizes the interfaces provided by Kubernetes to initiate or terminate containers that are used for training on specific workers. It employs the rSync command to migrate training state files over the network before resuming the training process.

6 Evaluation

6.1 Experiment Setup

6.1.1 Cluster setup. Our experimental cluster was developed on the Alibaba Cloud Platform and consisted of twelve T4 GPUs, twelve A10 GPUs, and twelve V100 GPUs. All hosts are equipped with the Ubuntu 20.04 LTS operating system, Python 3.8 interpreter, PyTorch v2.1.2, and CUDA 12.4 environment. Each server contained four GPU workers, and physical hosts were connected via 10 Gbps LAN.

6.1.2 Workloads. We leveraged a diverse range of DL training jobs as our workload, encompassing popular DL models from various domains. These models exhibit distinct runtime profiles, checkpoint sizes, and associated datasets, as outlined in Table. 2. We also assigned different batch sizes to these training models to enhance the diversity of the workload. Additionally, all training jobs apply Adam optimizer and gradient accumulation techniques to accelerate their training processes.

6.1.3 Traces. To showcase the comprehensive advantages of FFT, we conducted evaluations using nine different traces that adhere to the arrival time and processing time distributions in Philly [32]. Specifically, we performed experiments in the physical cluster under the first two traces, with the remaining traces being evaluated in the simulation testbed. For cluster evaluation, we designed Trace 1, which comprised of four-worker training jobs only, to enable a fair comparison with AlloX (which requires each job in a cluster to have the same GPU demand), and Trace 2, which consists of jobs with varying amounts of worker demand. Both Trace 1 and Trace 2 comprise 800 jobs that continuously enter the cluster over a period of two days. By default, we adopted trace-1 in the cluster testbed. In the simulation testbed, we used the remaining traces, each consisting of up to 20,000 jobs of different models and variants submitted in diverse frequency.

6.1.4 *Baselines.* We compared FFT with existing representative schedulers that are designed to optimize JCT or ensure fairness. Although Tiresias [16] is not initially designed for heterogeneous DL clusters, we made adaptations to ensure its compatibility in heterogeneous GPU clusters, i.e., randomly assigning each incoming DL training job to queues for different GPU types. We achieved this by maintaining a queue for each GPU type, and upon job arrival, they were randomly placed in one of the queues.

- **AlloX [23]**. AlloX offers an efficient scheduling algorithm to single-worker jobs. Particularly, it allocates newly arrived jobs to the GPU queue with the lowest JCT contribution.
- **Gavel [33]**. Gavel includes several scheduling policies and the SJF-like policy turns out to be the best policy for minimizing JCT. It adopts a multi-round approximation scheme to preserve its throughput-based allocation in the long term, which is oblivious to switching overhead.
- **Tiresias [16]**. Tiresias employs a scheduling policy based on Shortest Remaining Service Time-First (SRSTF). Under this approach, incoming jobs are directed to the queue of the GPU type with the smallest cumulative remaining service time. Moreover, inter-GPU type migration is not permitted due to the absence of a heterogeneity-aware control mechanism.
- **PAL [21]**. PAL aims to mitigate the performance variability effects of homogeneous GPUs on training processes by leveraging locality-aware job placement—a strategy analogous to scheduling in heterogeneous GPU clusters. However, it lacks the ability to incorporating opportunistic migration when updating the placement of DL jobs.

6.1.5 *Fairness quantification.* We utilized two metrics to evaluate fairness. The first one is the Finish-Time-Fairness (FTF) introduced in Themis [27]. Specifically, the FTF value of each job is defined as $\rho = T_{sh}/T_{id}$, where T_{id} is the completion time in an exclusive cluster, and T_{sh} is the completion time in a shared cluster. It accounts for the slowdown due to placement and any queuing delays that the training jobs experience in the shared cluster. As per the definition, FTF

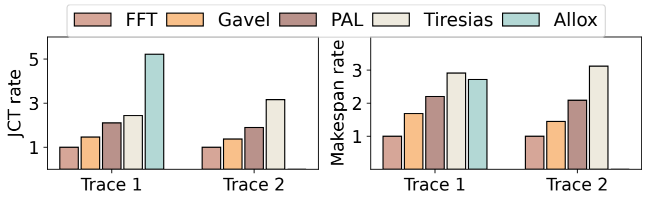


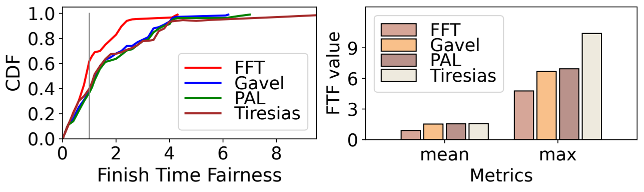
Figure 7: The efficiency advantages of FFT.

underscores the incentive to share resources within a cluster, with a tighter distribution of ρ signifying improved fairness performance. The second metric is the starvation time of jobs, which refers to the duration between job submission and the first time it is executed.

6.1.6 *Simulation setup.* Although there exist several simulators designed for evaluating scheduling strategies in large-scale clusters [5, 6, 52, 54–56], they do not ultimately meet our requirements. Specifically, they cannot capture the significant overhead of migrating DL training jobs, which involves large-volume states including optimizer states and model parameters. Additionally, they also fail to characterize the diverse sensitivity of DL training jobs to inter-GPU bandwidth under varying topologies, making it difficult to simulate the real-world collective-communication overhead involved in DL training. To this end, we developed a simulator to validate the effectiveness of FFT in a large-scale DL cluster. The training data in our simulator, including training throughput and migration overhead, is collected from the real-world experiments. These information helps mimic a precise long-term execution result in cluster testbed. To validate the fidelity, we compared the scheduling results generated by our simulator with those obtained from the cluster testbed. The maximum JCT deviation is 4.9%.

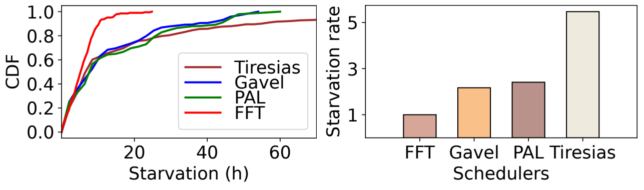
6.2 Performance in Physical Cluster

6.2.1 *Fast job completion.* Fig. 7 presents the comparison of FFT with other schedulers in terms of JCT and makespan performance. FFT employs fine-grained allocation to enable precise switching control, achieving significant JCT reductions of 1.46× and 1.37× compared to Gavel across both experimental traces. In contrast, PAL's reliance on coarse-grained cluster-level resource management and its lack of opportunistic migration lead to a 2.11× increase in JCT. Tiresias performs notably worse, exhibiting slowdowns of up to 3.15× relative to FFT due to its inability to dynamically leverage GPU migration capabilities. Furthermore, when compared to AlloX under Trace 1 where each job requested the same amount of GPU workers, FFT still outperforms AlloX by achieving a remarkable JCT reduction of up to 5.22×. This illustrates that although AlloX is aware of the heavy switching overhead of DL jobs and schedules jobs in a heterogeneity-aware manner, its failure to prioritize short jobs and leverage migration benefits hampers JCT optimization.



(a) The distribution of Finish Time Fairness **(b) Mean and Max value of Finish Time Fairness**

Figure 8: The advantage of Finish Time Fairness performance under FFT over other baselines.

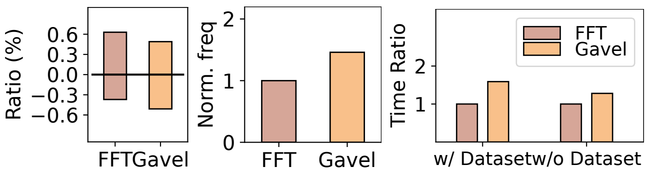


(a) Overall starvation **(b) Maximum starvation time**

Figure 9: The distribution of the starvation time and the maximum starvation time of DL training jobs.

We conducted further analysis on the makespan, which represents the time required to complete all training jobs in Fig. 7. It is worth noting that AlloX consistently minimizes the waiting time for each incoming job, resulting in a relatively shorter makespan. However, thanks to FFT's migration capability and delicate migration control, long-waiting jobs could utilize idle resources or benefit from the acceleration of high-end GPU workers, leading to a significant reduction in makespan. Specifically, FFT outperforms baselines by up to 3.12×.

6.2.2 Long-term fairness. We conducted an investigation into long-term fairness performance among jobs. As AlloX lacks the switching control that can provide jobs with fairness, we removed it from the baselines for comparison. Gavel, PAL, and Tiresias prioritize short jobs over time, resulting in more short jobs with small JCT and more long jobs with greater JCT. As a result, both of these schedulers generated relatively slacker FTF distributions than FFT, as depicted in Fig. 8(a), which indicates inferior fairness performance. Moreover, PAL performs similarly to Gavel due to their similar preemptive policies. With the exception of a few short jobs where Tiresias has a slight advantage over FFT, FFT overwhelmingly outperforms both baselines in offering smaller FTF values. Specifically, 70% of jobs could yield a FTF value smaller than 1 under FFT, whereas only 38% and 49% of such jobs exists under Gavel and Tiresias, respectively. More importantly, as shown in Fig. 8(b), FFT reduces the mean FTF value by 1.64× and 1.67×, and cut down the maximum FTF value by 1.4× and 2.18× against Gavel and Tiresias, respectively. This indicates that FFT grants more scheduling opportunities to jobs requesting more service quantum, enabling them to complete earlier than baselines.



(a) Beneficial migration **(b) Switching frequency** **(c) Dataset transfer and overall overhead**

Figure 10: An ablation study on migration, where the beneficial migration ratio, job switching frequency, and non-training overhead under FFT are presented.

In addition to FTF, we also investigated the first starvation time of jobs across baselines to examine the impact of fairness from a microscopic perspective. As shown in Fig. 9(a), FFT consistently achieves the shortest starvation times compared to the other schedulers. This advantage can be attributed to the design of the fairness coefficient in FFT, which allows effective fairness compensation for both short and long jobs, effectively reducing the overall starvation time for all jobs. On the other hand, due to the lack of fairness considerations in their design, both Gavel and Tiresias are insufficient in mitigating severe starvation for long jobs. As depicted in Figure 9(b), they increase the maximum starvation time by up to 2.03× and 6.35× respectively compared to FFT.

6.2.3 Switching control performance. We have also explored the performance of FFT's switching control. Since Gavel is the only scheduler that can take advantage of higher-end GPU resources on-the-fly, we selected Gavel as the baseline for comparison.

In Fig. 10(a), we observe significant migration benefits under FFT compared to Gavel. Specifically, 63% of migrations were towards higher-end devices under FFT, while this number was only 49% under Gavel. This indicates that FFT can effectively leverage switching control in the cluster level to fully utilize heterogeneous resources and avoid unnecessary migrations.

Furthermore, FFT achieves a 1.46× reduction in switching frequency as depicted in Fig. 10(b). This reduction is mainly attributed to the optimization of workload placement in the host level, which minimizes time spent on dataset transfer and switching events. Specifically, in Fig. 10(c), FFT reduces task-switching latency by 1.59× through proactive eviction of training datasets when host-local storage reaches capacity. Even without dataset migration, it still achieves a 1.28× reduction in switching overhead. These improvements significantly contribute to the JCT advantage of FFT.

6.2.4 Benefit of FFT's placement. We performed an ablation experiment to demonstrate the benefits of FFT's workload placement scheme in mitigating communication overhead. To control variables, we conducted scheduling using Trace 2 on A10 hosts and subsequently compared the training

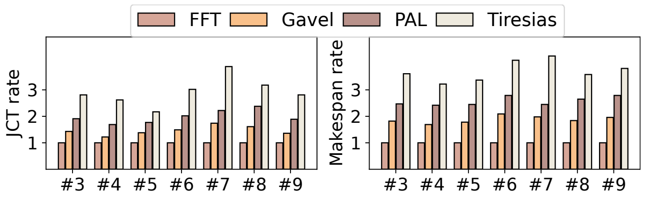


Figure 11: Efficiency performance of large-scale simulation on different traces.

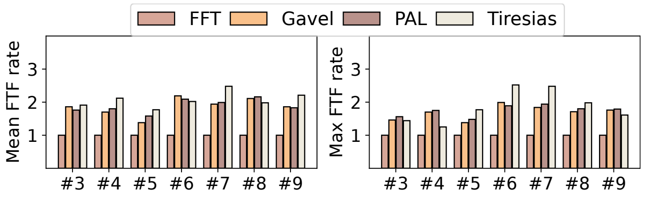


Figure 12: Fairness performance of large-scale simulation on different traces.

throughput under FFT with that of the baselines. Among the baseline schedulers, Tiresias distinguishes itself as the sole scheduler that takes placement impact into account. Nevertheless, its focus solely on the distribution of tensor size, while overlooking the effects of the communication paradigm. Consequently, it consistently diminishes training throughput by an average of 5.2%. Given Gavel and Allox's lack of special consideration for placement, the throughput under their placement schemes lags behind FFT by 12.4%.

6.2.5 Profiling overhead. We also conducted measurements on the overhead of FFT's profiling process, specifically focusing on preemption overhead and profiling time. The results indicate that these two components contribute to only a small fraction of the total training time, i.e., up to 0.8%.

6.3 Performance in Large-scale Testbed

6.3.1 Efficiency performance. To examine how FFT performs under different settings, we utilized a wide range of traces. In particular, we compared the performance across seven different traces using our simulator, namely Trace 3 to Trace 9. Among these traces, Trace 6 stands out as it consists of a significant number of large models. Approximately 50% of the models in Trace 6 are GPT-neo, GPT-2, and OPT, which require at least four workers and generated large files for training state. Additionally, Trace 7 experiences the highest intensive arrival rate, while Trace 9 has the smallest job length variance, with a factor of 3× between the shortest and longest jobs.

As depicted in Fig. 11, FFT showcases superior performance compared to the baselines across all scenarios. Notably, both Trace 6 and Trace 7 exhibit the most remarkable results, highlighting the significant advantages of FFT in handling intensive cluster contention and high switching overhead through more precise switching control. Specifically, FFT achieves a JCT reduction of 1.74×, 2.38×, and

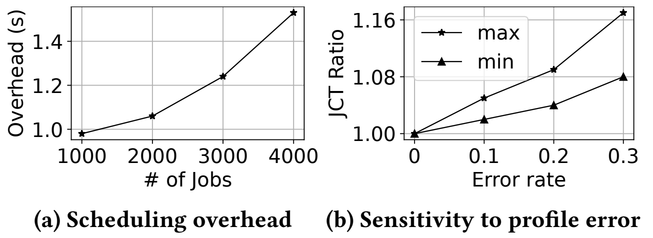


Figure 13: The sensitivity analysis result of FFT, where both scheduling overhead and the prolongment of JCT given profiling errors are in a moderate range.

3.88× compared to Gavel, PAL, and Tiresias, respectively. A similar trend was observed in terms of makespan, where FFT achieves the highest reductions under Trace 6 and 7. Furthermore, when comparing Trace 9 with other normal traces (e.g., Trace 3-5, Trace 8), FFT proves minimal performance differences in JCT, indicating robustness to highly variable job length.

6.3.2 Fairness performance. Moreover, we conducted experiments to highlight the superior long-term fairness of FFT under various scenarios. Specifically, FFT demonstrates the greatest performance margin over the baselines in the most resource-demanding traces, namely Trace 6 and Trace 7, as illustrated in Fig. 12. This highlights FFT's ability to effectively balance fairness and efficiency even in highly intensive contention scenarios, thanks to its fairness compensation mechanism. Specifically, compared to Gavel and Tiresias, FFT can reduce the mean FTF value by up to 2.19× and 2.48×, and the maximum FTF value by up to 1.99× and 2.52×.

6.4 Sensitivity Analysis

In this part, we also evaluated the performance of FFT concerning two important considerations in scheduling, i.e., scalability and tolerance to profiling errors, both of which are critical in the large-scale training background.

6.4.1 Scalability. Although FFT relies on Integer Linear Programming (ILP) to find out allocation schemes, it only incurs minimal computational overhead due to its well-designed formulation. Specifically, a limited number of GPU variables are involved since we consider GPU types instead of all GPU devices in the objective function and constraints, thereby significantly reducing the optimization overhead. As illustrated in Fig. 13(a), when scheduling a cluster of 1000 GPUs with eight types for 4000 jobs, the ILP-based solver introduces a mere overhead of 1.53 seconds. This overhead is significantly smaller than the duration of a scheduling round, i.e., 5 minutes in our system by default.

6.4.2 Tolerance to profiling errors. We conducted an analysis to evaluate the ability of FFT to handle workload estimation errors, as users often struggle to provide precise values for the total training iterations of their jobs. We focused on assessing the performance of JCT under varying degrees of estimation error rates. Results in Figure 13(b) highlight the robustness of FFT in managing inaccuracies in job size estimation. Specifically, when the number of epochs specified by the user deviated by up to 30% from the actual value, FFT exhibits only a modest increase in the overall JCT, reaching a maximum of 17%. Moreover, using the trace-6 and trace-7, we observed a JCT prolongment as low as 8% under the same error rate setting.

7 Related Works

DL schedulers for JCT minimization. Recently, a variety of schedulers have been developed to minimize JCT [16, 21, 30, 37, 46, 47, 51, 53]. For instance, Gandiva and Tiresias effectively reduces JCT by implementing an efficient packing policy and adapting heuristic policies (e.g., SRSF and LAS) in the homogeneous GPU cluster, while TopoOpt optimizes communication topology to accelerate training. Optimus allocates resources based on the model convergence curve. Beyond conventional schedulers, recent elastic resource schedulers dynamically provision heterogeneous GPUs for distributed deep learning (DL) training. For instance, Sia [46] enhances scheduling efficiency by dynamically adjusting batch sizes and GPU assignments during training. In contrast, FFT intentionally avoids modifying training configurations (e.g., batch size, model architecture) to preserve model accuracy, demonstrating strong orthogonality to Sia's approach. Similarly, Hadar [47] leverages parallelism by decomposing jobs into independent, single-GPU subtasks and strategically allocating heterogeneous resources across them. However, this design inherently restricts subtasks to single-GPU execution, precluding multi-GPU allocations for further JCT reduction—a key divergence from FFT, which optimizes distributed training workflows requiring coordinated multi-GPU execution.

DL schedulers for fairness. Themis [27] introduces an auction mechanism to provide Finish-Time Fairness for DL jobs in homogeneous DL clusters, which ensures sharing incentives across jobs within the cluster. Shockwave [60], on the other hand, dynamically adjusts hyperparameters of jobs to achieve fairness. Gandiva$_{\text{fair}}$ [7] and OEF [29] strive to provide tenant-wise fairness in heterogenous DL clusters, where the overall training throughput improvement is balanced among users each has multiple identical training jobs. Therefore, their scheduling policies are orthogonal to job-wise JCT minimization.

Unrelated Machine Scheduling. Scheduling DL workloads in heterogeneous clusters is analogous to unrelated machine scheduling problem, where a body of works have proposed randomized policies [11, 44, 45]. However, these solution are generally with high complexity and cannot be practically used in production clusters. Another research direction is to bound the performance of fair sharing policy using linear approximations [3, 11, 20]. Although theoretically solid, they fails to balance between fairness and JCT in real-world scheduling scenarios.

Checkpointing techniques. With the continuous growth of model size, mitigating the overhead of checkpointing for deep learning training has gained widespread attention [12, 17, 25, 31]. However, these existing works mainly focus on providing transparent and non-interruptible checkpointing function to the training workflow, which is orthogonal to our claim for opportunistic migration. To be specific, resuming the training process elsewhere always requires the up-to-date training states to be in place, leaving the checkpointing overhead, i.e., the transmission overhead over network and PCIe, unable to be hidden.

8 Conclusion

Existing heterogeneity-aware schedulers often struggle to incorporate both JCT reduction and fairness guarantees due to their coarse-grained resource allocation mechanism and poor integration between fairness and efficiency. Motivated by these limitations, this paper introduces FFT, a cluster-level scheduler designed for fast and fair DL training in a heterogeneous GPU cluster. FFT incorporates two key designs to simultaneously address efficiency and fairness. The first design is a fine-grained resource allocation vector proposed to enable efficient resource utilization, and the second one is a flexible fairness integration scheme that evaluates fairness in real-time. By doing so, FFT can strike a delicate balance between JCT and fairness without incurring excessive migration or starvation.

FFT is further distinguished by its placement scheme. Through a straightforward placement heuristic, FFT effectively mitigates network bottlenecks that could otherwise degrade large-scale training. Another advantage of FFT is its low complexity, rendering it a practical choice for implementation in large-scale production clusters. These features position FFT as an effective solution for scheduling DL training workloads in diverse and complex computing environments.

Acknowledgments

We thank the anonymous reviewers and our shepherd for their helpful comments. This work is supported in part by the Science and Technology Development Fund of Macau (0071/2023/ITP2 and 0024/2022/A1), as well as the Multi-Year Research Grant of University of Macau (MYRG-GRG2024-00255-FST-UMDF and MYRG-GRG2023-00019-FST-UMDF). This work is also supported in part by the CUHK MobiTeC fund.

References

[1] [n. d.]. https://www.alibabacloud.com/product/heterogeneous_computing.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[3] Anand et al. 2012. Resource augmentation for weighted flow-time explained by dual fitting. In *Proceedings of SODA*. SIAM.

[4] Dimitri P. Bertsekas et al. 2003. *Convex Analysis and Optimization*. Athena Scientific.

[5] Henri Casanova. 2001. Simgrid: A toolkit for the simulation of application scheduling. In *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, 430–437.

[6] Henri Casanova, Rafael Ferreira da Silva, Ryan Tanaka, Suraj Pandey, Gautam Jethwani, William Koch, Spencer Albrecht, James Oeth, and Frédéric Suter. 2020. Developing Accurate and Scalable Simulators of Production Workflow Management Systems with WRENCH. *Future Generation Computer Systems* 112 (2020), 162–175. doi:10.1016/j.future.2020.05.030

[7] Shubham Chaudhary et al. 2020. Balancing Efficiency and Fairness in Heterogeneous GPU Clusters for Deep Learning. In *Proceedings of Eurosys*.

[8] Chen et al. 2017. An online convex optimization approach to proactive network resource allocation. *IEEE Transactions on Signal Processing* 65, 24 (2017), 6350–6364.

[9] Chen Chen et al. 2020. Semi-dynamic load balancing: Efficient distributed learning in non-dedicated environments. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 431–446.

[10] A. Demers et al. 1989. Analysis and simulation of a fair queueing algorithm. In *Proceedings of ACM Sigcomm*.

[11] Nikhil R. Devanur et al. 2018. A Unified Rounding Algorithm For Unrelated Machines Scheduling Problems. In *Proceedings of ACM SPAA*.

[12] Assaf Eisenman, Kiran Kumar Matam, Steven Ingram, Dheevatsa Mudigere, Raghuraman Krishnamoorthi, Krishnakumar Nair, Misha Smelyanskiy, and Murali Annavaram. 2022. {Check-N-Run}: A checkpointing system for training deep learning recommendation models. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 929–943.

[13] Ali Ghodsi et al. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proceedings of NSDI*.

[14] Robert Grandl et al. 2014. Multi-resource Packing for Cluster Schedulers. In *Proceedings of ACM SIGCOMM*.

[15] Diandian Gu et al. 2023. ElasticFlow: An Elastic Serverless Training Platform for Distributed Deep Learning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 266–280.

[16] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU cluster manager for distributed deep learning. In *Proceedings of NSDI*.

[17] Tanmaey Gupta, Sanjeev Krishnan, Rituraj Kumar, Abhishek Vijeev, Bhargav Gulavani, Nipun Kwatra, Ramachandran Ramjee, and Muthian Sivathanu. 2024. Just-In-Time Checkpointing: Low Cost Error Recovery from Deep Learning Training Failures. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 1110–1125.

[18] He et al. 2017. Mask r-cnn. In *Proceedings of CVPR*.

[19] Changho Hwang et al. 2021. Elastic Resource Sharing for Distributed Deep Learning. In *Proceedings of NSDI*.

[20] Im et al. 2015. Temporal fairness of round robin: Competitive analysis for lk-norms of flow time. In *Proceedings of ACM SPAA*.

[21] Rutwik Jain, Brandon Tran, Keting Chen, Matthew D Sinclair, and Shivaram Venkataraman. 2024. PAL: A Variability-Aware Policy for Scheduling ML Workloads in GPU Clusters. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–18.

[22] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[23] Le et al. 2020. AlloX: compute allocation in hybrid clusters. In *Proceedings of Eurosys*.

[24] Jiamin Li et al. 2023. Lyra: Elastic Scheduling for Deep Learning Clusters. In *Proceedings of Eurosys*.

[25] Yuanhao Li, Tianyuan Wu, Guancheng Li, Yanjie Song, and Shu Yin. 2024. Portus: Efficient dnn checkpointing to persistent memory with zero-copy. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 59–70.

[26] Giorgio Lucarelli et al. 2018. Online Non-preemptive Scheduling on Unrelated Machines with Rejections. In *Proceedings of ACM SPAA*.

[27] Mahajan et al. 2020. Themis: Fair and efficient GPU cluster scheduling. In *Proceedings of NSDI*.

[28] Mahdavi et al. 2012. Trading regret for efficiency: online convex optimization with long term constraints. 13, 1 (2012), 2503–2528.

[29] Zizhao Mo, Huanle Xu, and Wing Cheong Lau. 2024. Optimal Resource Efficiency with Fairness in Heterogeneous GPU Clusters. In *Proceedings of the 25th International Middleware Conference*. 36–48.

[30] Zizhao Mo, Huanle Xu, and Chengzhong Xu. 2024. Heet: Accelerating Elastic Training in Heterogeneous Deep Learning Clusters. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 499–513.

[31] Jayashree Mohan, Amar Phanishayee, and Vijay Chidambaram. 2021. {CheckFreq}: Frequent,{Fine-Grained}{DNN} Checkpointing. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. 203–216.

[32] Myeongjae et al. 2019. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *Proceedings of ATC*.

[33] Narayanan et al. 2020. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *Proceedings of OSDI*. 481–498.

[34] Deepak Narayanan et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[35] Michael J. Neely et al. 2017. Online convex optimization with time-varying constraints. arXiv preprint:1701.03974.

[36] Paszke et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* (2019).

[37] Yanghua Peng et al. 2018. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *Proceedings of Eurosys*.

[38] Michael L Pinedo. 2012. *Scheduling*. Vol. 29. Springer.

[39] Aurick Qiao et al. 2021. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th USENIX Symposium on Operating Systems Design and Implementation OSDI 21*.

[40] Qizhen et al. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *Proceedings of NSDI*.

[41] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[42] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020),

5485–5551.

[43] Shafiee et al. 2017. Fast YOLO: A fast you only look once system for real-time embedded object detection in video. *arXiv preprint arXiv:1709.05943* (2017).

[44] René Sitters. 2017. Approximability of average completion time scheduling on unrelated machines. *Math. Program.* 161 (2017), 135–158.

[45] Martin Skutella et al. 2016. Unrelated Machine Scheduling with Stochastic Processing Times. *Mathematics of Operation Research* 41, 3 (2016).

[46] Jayaram Subramanya et al. 2023. Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 642–657.

[47] Abeda Sultana, Fei Xu, Xu Yuan, Li Chen, and Nian-Feng Tzeng. 2024. Hadar: Heterogeneity-Aware Optimization-Based Online Scheduling for Deep Learning Cluster. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 681–691.

[48] Hugo Touvron et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[49] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[50] Vaswani et al. 2017. Attention is all you need. In *Advances in neural information processing systems*.

[51] Weiyang Wang et al. 2023. TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 739–767.

[52] Jingfeng Wu, Minxian Xu, Yiyuan He, Kejiang Ye, and Chengzhong Xu. 2024. Cloudnativesim: A Toolkit for Modeling and Simulation of Cloud-Native Applications. *Software: Practice and Experience* (2024).

[53] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. 2018. Gandiva: Introspective cluster scheduling for deep learning. In *Proceedings of OSDI*.

[54] Minxian Xu, Guozhong Li, Wutong Yang, and Wenhong Tian. 2015. Flexcloud: A flexible and extendible simulator for performance evaluation of virtual machine allocation. In *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. IEEE, 649–655.

[55] Wutong Yang, Minxian Xu, Guozhong Li, and Wenhong Tian. 2015. CloudSimNFV: modeling and simulation of energy-efficient NFV in cloud data centers. *arXiv preprint arXiv:1509.05875* (2015).

[56] Xu Yang, Zhou Zhou, Sean Wallace, Zhiling Lan, Wei Tang, Susan Coghlan, and Michael E Papka. 2013. Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–11.

[57] Hao Yu et al. 2020. A Low Complexity Algorithm with $O(\sqrt{T})$ Regret and $O(1)$ Constraint Violations for Online Convex Optimization with Long Term Constraints. *The Journal of Machine Learning Research* (2020).

[58] Hanyu Zhao et al. 2020. {HiveD}: Sharing a {GPU} cluster for deep learning with guarantees. In *14th USENIX symposium on operating systems design and implementation (OSDI 20)*. 515–532.

[59] Lianmin Zheng et al. 2022. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 559–578.

[60] Pengfei Zheng et al. 2023. Shockwave: Fair and Efficient Cluster Scheduling for Dynamic Adaptation in Machine Learning. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 703–723.