Taming Flexible Job Packing in Deep Learning Training Clusters

PENGYU YANG[*], Shanghai Jiao Tong University, China
WEIHAO CUI[*], Shanghai Jiao Tong University, China and National University of Singapore, Singapore
CHUNYU XUE, HAN ZHAO, CHEN CHEN, QUAN CHEN, JING YANG, and MINYI GUO, Shanghai Jiao Tong University, China

Job packing is an effective technique to harvest the idle resources allocated to the deep learning (DL) training jobs but not fully utilized, especially when clusters may experience low utilization, and users may overestimate their resource needs. However, existing job packing techniques tend to be conservative due to the mismatch in scope and granularity between job packing and cluster scheduling. In particular, tapping the potential of job packing in the training cluster requires a local and fine-grained coordination mechanism. To this end, we propose a novel job-packing middleware named Gimbal, which operates between the cluster scheduler and the hardware resources. As middleware, Gimbal must not only facilitate coordination among the packed jobs but also support various scheduling objectives of different schedulers. Gimbal achieves dual functionality by introducing a set of worker calibration primitives designed to calibrate workers' execution status in a fine-grained manner. The primitives obscure the complexity of the underlying job and resource management mechanisms, thus offering the generality and extensibility for crafting coordination policies tailored to various scheduling objectives. We implement Gimbal on a real-world GPU cluster and evaluate it with a set of representative DL training jobs. The results show that Gimbal improves different scheduling objectives up to 1.32× compared with the state-of-the-art job packing techniques.

CCS Concepts: • **Computer systems organization → Cloud computing**.

Additional Key Words and Phrases: Job packing, DNN training, GPU cluster, Co-location

ACM Reference Format:
Pengyu Yang, Weihao Cui, Chunyu Xue, Han Zhao, Chen Chen, Quan Chen, Jing Yang, and Minyi Guo. 2025. Taming Flexible Job Packing in Deep Learning Training Clusters. *ACM Trans. Arch. Code Optim.* 37, 4, Article 1 (August 2025), 23 pages. https://doi.org/XXXXXXX.XXXXXXX

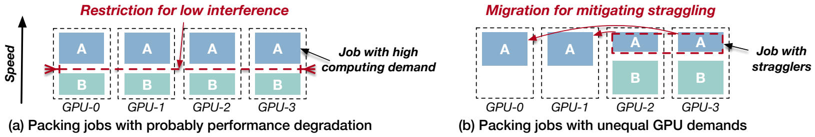
1 INTRODUCTION

GPU clusters are now fundamental infrastructures for training deep learning models, which usually host a vast number of training jobs submitted from multiple tennants [12, 20, 23, 34, 40, 50]. In such clusters, there usually exists a dedicated scheduler to manage the submitted jobs, and the scheduler design is crucial for cluster efficiency. In the literature, many scheduling techniques have been proposed to attain good performance in aspects like job completion time (JCT) [23], makespan [50], and deadline fulfillment [20].

[*]Both authors contributed equally to this research.

Authors' addresses: Pengyu Yang, yp_yuu@sjtu.edu.cn, Shanghai Jiao Tong University, China; Weihao Cui, weihao@sjtu.edu.cn, Shanghai Jiao Tong University, China and National University of Singapore, Singapore; Chunyu Xue, dicardo@sjtu.edu.cn; Han Zhao, zhaohan_miven@sjtu.edu.cn; Chen Chen, chen-chen@sjtu.edu.cn; Quan Chen, chen-quan@cs.sjtu.edu.cn; Jing Yang, jyang23@gzu.edu.cn; Minyi Guo, guo-my@cs.sjtu.edu.cn, Shanghai Jiao Tong University, China.

ACM 1544-3566/2025/8-ART1
https://doi.org/XXXXXXX.XXXXXXX


Restriction for low interference
Speed
A
B
Job with high computing demand
GPU-0
GPU-1
GPU-2
GPU-3
Migration for mitigating straggling
Job with stragglers

(a) Packing jobs with probably performance degradation (b) Packing jobs with unequal GPU demands

Fig. 1. Limitations of the conservative job packing in existing schedulers for training cluster.

In particular, given that GPU computing capability is growing exponentially [1, 3], job packing stands out as a key technique to maximize cluster resource utilization [23, 34, 50, 52]. According to the production traces [26], the allocated GPUs in typical training clusters are utilized at a very low level. By packing multiple jobs onto the same GPUs, cluster schedulers can harvest idle GPU resources, thus boosting cluster throughput and reducing job queuing delay. However, the packing mechanism of existing schedulers [23, 34, 50, 52] is over-conservative, which limits its capacity to improve cluster efficiency. Figure 1 illustrates the two primary limitations inherent in the conservative nature.

L1. Firstly, to avoid potential dramatic performance slow-down, job packing is limited to jobs with low interference. The jobs packed by existing schedulers [23, 34, 50] freely compete for resources, either in a time-sharing or a spatial-sharing manner. When packed, jobs with high computational demands are likely to interfere with each other significantly, as illustrated in Figure 1-(a). It can lead to worse performance than executing those jobs sequentially. However, our experimental results in §2.2 show that if we restrict the execution of one job's workers using a technique like CUDA Multi-Process Service (MPS) [35] in such cases, the interference between packed jobs can be minimized, thus bringing performance improvement.

L2. Secondly, to avoid stragglers in distributed training, job packing is limited to jobs with identical GPU number demands. As shown in Figure 1-(b), packing jobs with different numbers of GPUs would yield inconsistent processing speeds across different job workers. With no workload redistribution mechanism in existing schedulers, the slow workers would become stragglers and degrade the end-to-end training performance. However, if a fast worker can somehow "steal" certain workloads from the slower ones, it would be possible to mitigate those stragglers. That is, with intra-job workload rebalancing enabled, arbitrarily packing two jobs demanding different numbers of GPUs may still attain better overall performance than that without packing.

With the increasing prevalence of large language models, the deficiency of conservative packing is much more obvious. Large language models [9, 46, 47] have outstanding capabilities on general-purpose language generation and have substantially revolutionalized various application fields. Yet, pretraining or fine-tuning a large language model is usually very compute-intensive. Meanwhile, with the persistent adoption of classical CNN or RNN-based neural network models, production clusters often face a mixture of (traditional) "small" and (new) "large" model training jobs. In that sense, it is an urgent need to allow more flexible job packing to fully unleash the potential of resource utilization enhancement.

We find that the conservative job packing used in existing schedulers stems from the mismatch in scope and granularity between cluster scheduling and job packing. Schedulers are designed as job-level global managers, which allocate resources for training jobs in the cluster. Based on the profiled or predicted job performance, they are naturally suited for selecting appropriate jobs from the submitted ones for packing. However, restricting the execution of one job's workers and rebalancing the workload among workers are worker-level coordination mechanisms. They function locally within the packed jobs that share the same hardware resources. Moreover, such coordination is a runtime adjustment that needs to be conducted incrementally according to our investigation in §2.2. It is critical to have a local manager that coordinates the workers of each job packing combination in a fine-grained manner for tapping the potential of job packing.

To this end, we propose Gimbal, a novel job-packing middleware between cluster scheduler and hardware resources. As for each scheduled job packing combination, Gimbal locally coordinates the involved workers in a fine-grained way to relax the conservative nature of job packing. Meanwhile, as a middleware, Gimbal must be general and extensible to accommodate various scheduling objectives since schedulers are designed for different goals [20, 23]. At the core of Gimbal, we propose worker calibration primitives, `worker.squeeze() and worker.swell()`. The two primitives expose the capability for fine-grained worker coordination in a generic way, thus unifying the construction of tailored coordination policies for various schedulers.

Specifically, when a job worker is swelled, the worker's execution speed is adjusted to be faster, while the worker is squeezed when the execution speed needs to be slowed down (a process we refer to as *calibration*). Behind the calibration primitives, Gimbal incorporates two calibration mechanisms, low-level GPU resource restriction (addressing **L1**) and lossless input sample stealing (addressing **L2**), to achieve the adjustment of worker execution speed accordingly without modifying user code. At runtime, Gimbal maps primitives to different combinations of calibration mechanisms for worker coordination.

Since the primitives obscure the complexities of calibration mechanisms, cluster maintainers only need to use the primitives to implement the coordination policy that aligns with the scheduler's objectives. Instructed by the coordination policy, Gimbal further coordinates the involved workers for each scheduled job packing combination. We have extended state-of-the-art schedulers [23, 34, 50] using Gimbal's primitives for various scheduling objectives, including job-completion time, deadline awareness, and makespan. As for other scheduling objectives, the cluster maintainer could also implement the corresponding coordination policy with the two primitives.

We extensively evaluate Gimbal with production trace [48, 49] using representative benchmarks, including large language models and traditional deep learning models. Our experimental results show that Gimbal achieves average JCT and queuing delay improvement by 1.28× and 1.87× respectively, job completion ratio improvement by 1.22×, and makespan improvement by 1.12×. In this paper, we make the following contributions:

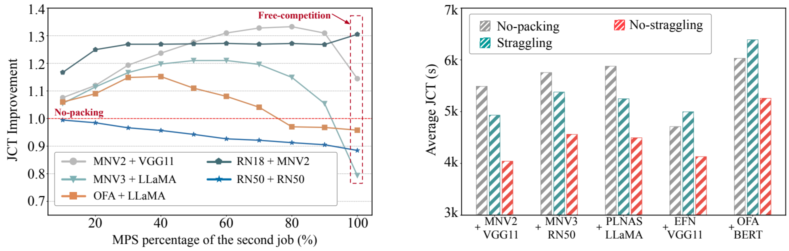
- We reveal that the root cause of the conservative nature of job packing in existing works is the scope and granularity mismatch between cluster scheduling and job packing.
- We relax the conservative nature of job packing with fine-grained coordination, thus tapping the potential of job packing for enhancing cluster efficiency.
- We propose general calibration primitives to locally and incrementally coordinate the workers of packed jobs, which are generic and extensible for supporting various scheduling objectives.
- We implement the state-of-the-art scheduling policies in Gimbal, showing significant improvement in job completion time, deadline awareness, and makespan.

2 BACKGROUND AND MOTIVATIOIN

In this section, we begin with providing an overview of cluster scheduling in deep learning clusters and the way of using job packing to improve cluster utilization. Next, we discuss the untapped potential in the current job packing with experimental results. Finally, we delve deeper into the root cause of these inefficiencies to motivate the design of Gimbal.

2.1 Scheduling In Cluster for Deep Learning

Large-scale deep learning clusters are widely used in industry and academia to train deep learning models [31, 39], which consist of a large number of GPUs and are multi-tenant. The same as



(a) Packing with different resource restriction schemes.

(b) Packing with different GPU numbers.

Fig. 2. Inefficiencies of the conservative job packing: (a) JCT Improvement for each packing pair with different MPS schemes, normalized to no-packing (launching jobs in serial). While the first job occupies at most 100% resources, the at-most resource of the second job varies from 0 to 100 percent. All jobs require 2 GPUs. (b) Average JCT for each scheduling job pair under no-packing, packing with straggling, and packing without straggling. Jobs are packed under free competition without resource restriction. While the first job requires 4 GPUs, the second job requires 2 GPUs.

traditional HPC clusters, deep learning clusters also need to allocate resources to different jobs and schedule them to maximize cluster utilization. Various dedicated schedulers have been proposed to tackle the unique scheduling challenges in deep learning clusters, including fulfilling different scheduling objectives [12, 20, 50], handling cluster heterogeneity [32, 34, 54, 55], supporting training jobs with adaptive resource requirements [25, 40], etc.

To enhance the scheduling efficiency, many schedulers employ job packing mechanism [23, 34, 50, 51], as it can harvest the resources allocated to jobs but not fully utilized. In deep learning clusters, user-specified resource requirements can often be overestimated to ensure the job can be finished within a reasonable time. Job packing enhances cluster-level hardware utilization by carefully selecting the inefficient jobs of complementary resource requirements for co-location [50].

2.2 Untapped Potential in Conservative Job Packing

Although job packing has proven effective [23, 34, 50], the methods employed by existing schedulers are over-conservative to prevent compromising scheduler objectives. They adhere to certain conservative practices: 1) only packing jobs with negligible interference under free-contest sharing; 2) only packing jobs with identical resource requirements, particularly GPUs. We conduct experiments using representative DL training jobs to explore inefficiencies of conservative job packing and identify potential improvements. The benchmarked training models are selected following the previous works and current trends. Model details and their abbreviations are presented in Table 1.

Minimizing slowdown in free-competition packing. Job packing under free-competition sharing can lead to dramatic slow-downs and performance instability, especially when the packed jobs are compute-intensive. New advanced GPU-sharing technologies, such as MPS, Multi-Instance GPU (MIG) [16, 35], have shown the ability to restrict resources among co-located programs. In this case, we examine job performance slow-down under resource-restricted co-location to find more beneficial co-location options than free-competition packing.

Figure 2-a shows the JCT of five packing pairs of two jobs with different resource restriction schemes. The scheduling objective in the experiment is JCT, and we use MPS for resource restriction. MPS does not ensure strict resource isolation but imposes a limit on the maximum resources a process can utilize. As shown in the figure, only two of the five co-location options outperform no packing under free competition (x = 100), whereas four perform better when resource restriction is enabled (such as x = 60). The two new additions feature Llama [47] for packing, a widely-used large language model. Therefore, sharing resources with restrictions can increase the opportunities for beneficial job packing.

Meanwhile, different co-location options have different performance curves, which depend on the jobs' characteristics. Since the training jobs arrive randomly at runtime, it is hard to prepare an accurate performance model before scheduling. Under these circumstances, we need to search for the optimal packing setup incrementally.

Mitigating stragglers in unequal packing. Unequal packing refers to the practice of packing jobs with inconsistent GPU requirements, which introduces stragglers. For instance, in Figure 1-(b), Job A (requiring 4 GPUs) and Job B (requiring 2 GPUs) are packed together. In this experiment, we manually mitigate stragglers caused by unequal packing and evaluate the impact of stragglers with or without straggling. Specifically, we adjust the batch sizes of a training job's workers for the load distribution. Based on this, all workers could complete the computation at a similar time.

Figure 2-b shows the average JCT of each scheduling job pair under no-packing, packing with straggling and packing without straggling. We observe that the packing jobs with straggling perform better in 3 out of the 5 benchmarked job pairs. This indicates that while stragglers exist in unequal packing, packing can be still beneficial in some cases. Meanwhile, after mitigating stragglers, job packing outperforms no-packing and packing with straggling in all cases. Therefore, it is also urgent to enable the automatic straggler mitigation mechanism in existing schedulers to maximize the potential of job packing.

In addition, the execution speed of different jobs has different relationships with the batch size, and the interference to the straggler depends on the co-running job. This also means that we cannot adjust the batch size of the workers in advance. Therefore, we also need to balance the workload in a fine-grained way.

2.3 Scope and Granularity Mismatch between Scheduling and Packing

Although the opportunities mentioned above alleviate the conservative nature, integrating them into existing schedulers is challenging. The core issue is the scope and granularity mismatch between scheduling and packing. A scheduler for a training cluster aims to enhance cluster efficiency at the job level by assigning the right jobs with suitable resource allocations. The scheduler selects jobs for packing because co-locating them aligns with the cluster's scheduling objectives. However, to enable the new potential improvement in §2.2, it is crucial to locally adjust each worker of the packed jobs in a fine-grained way. These mismatches between scheduling and packing hinder a training cluster from maximizing the potential of job packing. Therefore, this paper designs Gimbal, a generic middleware for automatically coordinating workers to bridge the gap between scheduling and packing. Upward, Gimbal adapts to various scheduler targets. Downward, it seamlessly integrates with the worker coordinating mechanisms to optimize job packing.

3 ARCHITECTURE OVERVIEW

Gimbal proposes to perform *fine-grained worker coordination* to fully tap the potential of job packing in DL training clusters. The core of Gimbal is a set of calibration primitives that speed up or slow down the workers of packed jobs. The calibration primitives facilitate the expression of various coordination policies towards different scheduling objectives. During runtime, Gimbal utilizes the primitives to calibrate workers' execution status based on the coordination policy, which instructs the job packing to maximize the performance related to the scheduling objective.

Architecture. Figure 3 illustrates the architecture of Gimbal. Gimbal works as a middleware between the scheduler and the hardware resources, enabling flexible and efficient job packing for various scheduling objectives. It consists of three modules: a *runtime monitor*, a *worker calibrator*, and

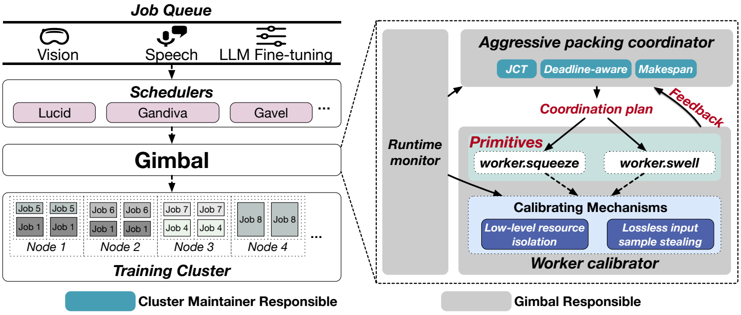


Fig. 3. Overview of Gimbal's system architecture.

an *aggressive packing coordinator*. In Gimbal, the runtime monitor and the worker calibrator work on their own, while the aggressive packing coordinator requires coordination policy customization for different scheduling objectives. Inside the aggressive packing coordinator, we have implemented the policies for minimizing JCT, makespan, and fulfillment of job deadlines. As for other scheduling objectives, the cluster maintainer is responsible for extending the coordinator by adding new policies with the proposed primitives.

Workflow. Gimbal is designed to be generic and extensible, which means it can be easily integrated into existing DL training clusters without modifying the scheduler. The schedulers in DL training clusters are responsible for selecting jobs for packing. With the packing decisions made, Gimbal then coordinates the workers of packed jobs to achieve the best packing efficiency.

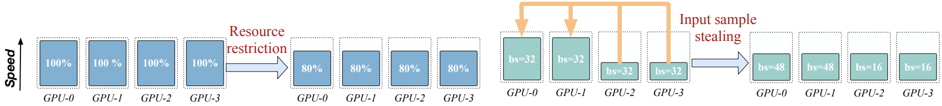
In a typical workflow, the scheduler (like Lucid [23]) selects jobs for packing and passes the scheduling decisions to Gimbal. Gimbal then launches the packed jobs on the allocated GPUs. The runtime monitor collects runtime information of workers, such as GPU utilization, memory usage, and training speed. Based on the runtime information, the aggressive packing coordinator finds an efficient coordination plan using its predefined coordination policy (JCT for Lucid as Lucid is a JCT-oriented scheduler). A coordination plan is a set of invocations of calibration primitives (more details in §4.2). Behind these primitives, Gimbal automatically calibrates workers' execution status using a combination of supporting mechanisms, such as low-level GPU resource restriction (LGRR) and lossless input sample stealing (LISS). The coordinator and calibrator work iteratively, which locate the optimal worker coordination status for the packing jobs in an incremental way.

4 CALIBRATION OF JOB PACKING

In this section, we present Gimbal's worker calibrator, which allows for fine-grained coordination of packed jobs. We start with calibration primitives and calibration mechanisms. Then we demonstrate the mapping between primitives and mechanisms for a single job. Finally, we will show how a coordination plan is applied through calibration primitives.

4.1 Calibration Primitives and Mechanisms

Calibration Primitives. Optimal fine-grained worker coordination for job packing performance involves joint use of the worker calibration opportunities outlined in §2.2. It is challenging for the schedulers to directly determine how to use these opportunities, as the coordination heavily relies on job placement and requires incremental adjustments. However, it is straightforward to determine whether the calibration effect, speeding up or slowing down a specific worker, benefits the scheduling objectives. For instance, Job A and Job B are packed together. Under free competition, Job B requires 5 hours to complete, but due to significant performance interference, Job A's completion time is extended to 10 hours. By squeezing resources from Job B, its completion time increases to 7



(a) Low-level GPU resource restriction mechanism. (b) Lossless input sample stealing mechanism.

Fig. 4. (a) LGRR controls the training speed of jobs by changing resource limits, while (b) LISS allows the load to flow between workers within a job.

hours; however, Job A gains more resources and improves its performance, reducing its completion time to 5 hours. As a result, by slowing down Job B and speeding up Job A, the average JCT decreases from 7.5 hours to 6 hours, leading to an overall improvement. In this case, Gimbal proposes two simple and intuitive calibration primitives for the scheduler to use for worker coordination.

- **`worker.squeeze()`**: The `squeeze` primitive is used to indicate that a worker's training speed needs to be slowed down.
- **`worker.swell()`**: The `swell` primitive is used to indicate that a worker's training speed needs to be sped up.

Mechanisms for incremental calibration. To support primitives and utilize opportunities in §2.2, we implement two underlying mechanisms that incrementally calibrate workers' execution status. These mechanisms do not directly correspond to primitives, but their activation ultimately achieves the effect of accelerating or decelerating the worker's speed as dictated by the primitives.

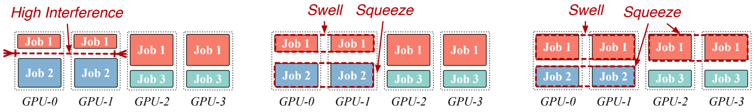
Low-level GPU resource restriction **(LGRR)**. In job packing, if a training job is computationally intensive or has a large batch size, it may fill the GPU task queue with numerous CUDA kernels, thereby monopolizing the majority of GPU resources. This causes packed jobs to be forced to wait, resulting in severe and unstable performance degradation. Gimbal devises a mechanism named low-level GPU resource restriction to alleviate these issues based on MPS [35].

Figure 4-(a) shows an example of restricting job workers with LGRR. Restricting a worker's allocated resources through MPS slows down its corresponding speed, and the co-located one can launch its kernels with more resources, thus achieving a stable speedup. In Gimbal's calibrator, we adjust the resource restriction at a fixed granularity percent of the total GPU resources. Through empirical evaluation, we found that granularity of 10% strikes a good balance between coordination duration and the quality of results, as detailed in §7.4. Notably, the resource allocated by MPS to all packed jobs can exceed 100% (200% represents free-competition). In this case, kernels from co-located jobs still can be interleaved.

Lossless input sample stealing **(LISS)**. In unequal packing, where a distributed job is co-located with several different jobs, the varying degrees of interference lead to stragglers among the workers. Inspired by LB-BSP, a method designed to mitigate stragglers in deep learning training on heterogeneous GPUs [13], we introduce the lossless input sample stealing mechanism to address this issue. We consider the input sample as a load and allow faster workers to steal the sample from slower workers, thereby reducing the workload of slower workers and enhancing their training speed, as shown in Figure 4-(b). The convergence and correctness of the imbalanced distribution of input samples among workers have been proven by LB-BSP. Although Gimbal does not originate the concept of modifying workers' sample numbers for straggler mitigation, it is the first to apply this method to mitigate stragglers in scenarios involving unequal packing.

4.2 Applying Coordination Plan through Calibration

Gimbal calibrates workers according to its received coordination plan. Therefore, Gimbal is required to map primitives to mechanisms automatically. In this subsection, we first introduce the mappings and then demonstrate the application of the coordination plan with a typical example.



(a) Unequal packing jobs (b) LGRR mechanism (c) LISS mechanism

Fig. 5. Stragglers appear in unequal packing jobs, reducing overall training efficiency. In (a), GPU 0 and 1 cause significant interference, slowing down Job 1 training. This issue can be mitigated using the low-level GPU resource restriction (LGRR) mechanism and the lossless input sample stealing (LISS) mechanism, as shown in (b) and (c), respectively.

Mapping Primitives to Mechanisms. For each job, depending on the combination of worker primitives, the worker calibrator will use different underlying mechanisms, as shown in **??**. Based on the different workers squeezed or swelled, Gimbal derives the following three possibilities.

- **Squeeze/Swell only the specified N workers**: Only slowing down/speeding up the workers in the current job. Gimbal's calibrator first activates LGRR to set a smaller/larger resource limit for these workers. Since LGRR may lead to the emergence of stragglers, Gimbal then utilizes LISS adaptively to eliminate the newly generated stragglers within the job in runtime.
- **Squeeze N workers, Swell M workers**: Slowing down N workers while speeding up M others. This strategy is primarily used when there are significant performance disparities among workers within a job. LISS will be applied to redistribute the input samples among these workers while keeping the total batch size for job convergence.

Coordination example. Different combinations of primitives are mapped to different underlying support mechanisms, but this does not imply that the two mechanisms work independently. In fact, during the coordinating process, these two mechanisms are closely integrated and work together. While one job receives a combination of primitives, the other packed jobs may receive a different set of primitives, leading to the employment of different underlying mechanisms.

Figure 5-(a) shows a typical example after relaxing the limitations of conservative packing: three jobs are packed together on 4 GPUs. In this case, we assume that the interference between Jobs 1 and 2 is significant, leading to Job 1's training performance being severely affected. The coordinator needs to accelerate Job 1 for better cluster efficiency. The coordinator may have two options to coordinate the packed jobs to achieve the best performance.

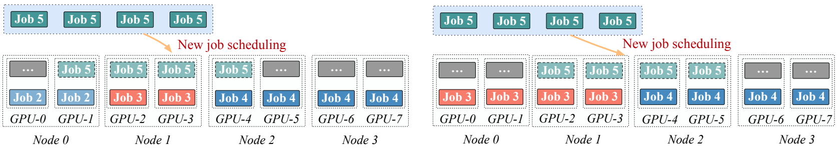
Figure 5-(b) shows the first option, which squeezes the workers of Job 2 and swells the workers of Job 1 on GPUs 0 and 1. Figure 5-(c) shows the second option where we swell the workers of Job 1 on GPUs 0 and 1 and squeeze the workers of Job 1 on GPUs 2 and 3 while squeezing the workers of Job 2 on GPUs 0 and 1. The two options both aim to accelerate Job 1. The coordinator can choose the best option based on the runtime metrics and the current status of the packed jobs.

4.3 Superiority of Calibration

There are two key advantages of the calibration primitives in Gimbal.

First, the primitives decouple the coordination policy construction from the intricate utilization of underlying coordination opportunities. The decoupling is essential for Gimbal, as it aims to provide a generic and scalable solution of job packing for various schedulers. Different schedulers may have different objectives, and the calibration primitives allow them to focus on the coordination policy construction without worrying about the underlying mechanisms.

Second, the primitives allow the coordinator to make fine-grained adjustments to the packed jobs. In this case, Gimbal can find the optimal packing configuration by incremental coordination. As the incremental calibration is done on the fly, the coordination overhead is minimal. Moreover, the fine-grained adjustments do not require the restart of the training jobs. Due to the fact that



(a) Violation of topology leading to cascading.

(b) Cascading occurs in the optimal topology.

Fig. 6. In unequal packing, cascade effects occur when jobs are interconnected through co-location, leading to one job's changes impacting numerous others. Cascade effects arise from two kinds of scheduling decisions. (a) Violation of optimal topology. (b) The lack of constraints on newly packed jobs.

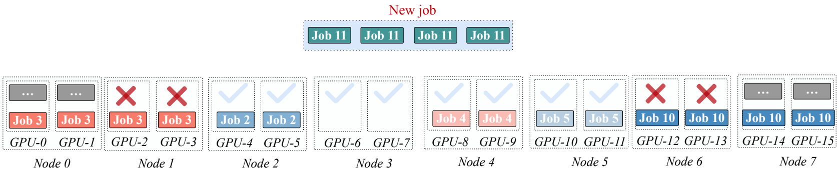


Fig. 7. To prevent cascade effects, job placement must be carefully managed. Since job 3 and job 10 are already co-located with other jobs, placing job 11 on GPUs (1, 2, 12, 13) would cause cascade effects. Therefore, nodes (1, 6) are excluded. Nodes (2, 3, 4, 5) remain available, and any two can be chosen.

different schedulers may be designed with varying requirements for job preemption, the on-the-fly calibration further enhances the flexibility of GIMBAL to adapt to different schedulers.

5 COORDINATION OF AGGRESSIVE JOB PACKING

GIMBAL is designated to enhance job packing for various schedulers instructed by the calibration primitives. We now demonstrate how to use the calibration primitives to coordinate workers for different scheduling objectives. This section is organized as follows. We first discuss the constraints for job packing in §5.1. While GIMBAL relaxes the conservative job packing, it still adheres to several rules, which are crucial for maintaining the performance of the training jobs. Schedulers should follow the rules to make job packing decisions, but it is optional since GIMBAL can sequentially execute the packed jobs that violate the rules. Then, we introduce the coordination policy in §5.2

5.1 Constraints for Job Packing

There are two constraints for job packing in GIMBAL. Firstly, GIMBAL only allows two job workers to be co-located on a single GPU, since co-location of more workers does not provide more benefits[23, 34, 50]. Secondly, GIMBAL add constraints for topological guaranteeing. Without such constraints, job packing may become unbounded. We demonstrate the necessity of the topology constraints as follows.

Topological guaranteeing. In unequal packing, the absence of constraints on co-location relationships can initiate a cascading effect among jobs. Specifically, unequal packing leads to a complex interdependency among jobs, where a perturbation in one job can trigger a ripple of subsequent changes throughout the chain. For instance, in Figure 6, the scheduling of Job 5 introduces new interference for Job 4, which then sequentially impacts Job 3, and so on, potentially disrupting all active jobs.

The cascading effects occur when the topological structure of training jobs is overlooked in GPU resource allocation. As depicted in Figure 6-(a), Job 3 requires 4 GPUs but is incorrectly assigned to 3 nodes, leading to cascading impacts on all jobs in the cluster. We follow existing works [55] to ensure that the scheduler makes decisions that adhere to the optimal topology, with each job placed on the theoretically smallest number of nodes.

While optimal topological constraints can avoid the majority of cascade scenarios, instances of overly complex co-location relationships may persist, as shown in Figure 6-(b). To comprehensively eliminate cascade effects, we introduce an additional constraint: jobs with a small world size (fewer GPUs) are permitted to co-locate with another larger job. In this case, the jobs with the same world size only engage in balanced co-location. Figure 7 demonstrates this decision in Gimbal, where nodes (2, 3, 4, 5) are suitable for scheduling Job 11, while nodes (1, 6) are excluded.

This constraint ensures that the co-location relationships within the cluster are uniform. All jobs form a series of job packing sets, each containing a job with the largest world size (denoted as J_{max}). This job co-locates with all other jobs in the set, while the remaining jobs only co-locate with it. Gimbal's subsequent policy are constructed based on this co-location pattern.

5.2 Constructing Coordination Policy

Gimbal constructs the coordination policy in the following ways. We first define the functions of obtaining the local objective and the job priority for each job packing set. The local objective aligns with the scheduler's objectives. Within each set, Gimbal iteratively monitors the local objective and generates coordination plans based on job priority to calibrate the workers.

Defining local objectives and job priority. In Gimbal, we define the objective functions and job priorities for the following three scheduling goals.

For **JCT**, we define the objective function and job priority as follows:

$$\text{Local Objective} = \min \frac{1}{\text{JobSet.size}} \sum_{J \in \text{JobSet}} \frac{\text{RemainingSteps}_J}{\text{Throughput}_J} \tag{1}$$

$$\text{Job Priority}_J = -\text{Utilization}_J \tag{2}$$

Where RemainingSteps is the number of steps remaining for the job, Throughput is the current throughput of the job, and Utilization is the GPU utilization of the job. The objective is to minimize the average JCT. Job priority is determined by the GPU utilization of each job. This prioritization method is chosen because the performance loss caused by reducing resources for large jobs is often smaller than the performance gain achieved by allocating the same resources to smaller jobs. For instance, slightly squeezing a large job can still maintain near-optimal performance, while co-located small jobs can be significantly sped up. Overall, this coordination strategy can effectively reduce the average JCT.

For **deadline-aware**, the objective and priorities of jobs are as follows:

$$\text{Local Objective} = \min \max_{J \in \text{JobSet}} \max(0, \text{Deadline}_J - \frac{\text{RemainingSteps}_J}{\text{Throughput}_J}) \tag{3}$$

$$\text{Job Priority}_J = \text{Deadline}_J - \frac{\text{RemainingSteps}_J}{\text{Throughput}_J} \tag{4}$$

Where Deadline is the job's deadline. The objective is to minimize the maximum duration of jobs that exceed their deadlines. This objective value illustrates the benefits of new calibration during the search process, serving as a useful indicator for subsequent search strategies. Jobs closer to their deadlines have higher priority, while those with more headroom have lower priority. This means we will squeeze jobs with longer headroom to accelerate those with shorter headroom or exceeded deadlines, thus ensuring more jobs meet their deadline requirements.

For **makespan**, we define the objective function and job priority as follows:

$$\text{Local Objective} = \min \max_{J \in \text{JobSet}} \frac{\text{RemainingSteps}_J}{\text{Throughput}_J} \tag{5}$$

$$\text{Job Priority}_J = -\frac{\text{RemainingSteps}_J}{\text{Throughput}_J} \tag{6}$$

The objective is to minimize the longest expected duration of the current job set, aiming to achieve the shortest possible completion time. Job priority is defined by its duration. This means that shorter jobs are squeezed to speed up the completion of longer jobs, in alignment with the overall objective.

Searching algorithm. As previously mentioned, packed jobs in the cluster form independent job sets. Therefore, the coordinator only needs to consider the calibration of jobs within a single job set. Algorithm 1 presents the pseudocode for the Gimbal coordination searching process over a packed job set.

Algorithm 1 Multi-Job Coordination Search

```
    Function: CoordinationSearch (jobset)
 1: $J_{\max}$ = FindJobWithMaxWorldSize(jobset)
 2: subset1, subset2 = {$J_{\max}$}, jobset - {$J_{\max}$}
 3: if Priority(subset2) < Priority(subset1) then
 4:     Swap(subset1, subset2).
 5: swell(subset2)
 6: while True do
 7:     GeneratePrimitives(subset1, $J_{\max}$)
 8:     Collect runtime info, compute calibration gain.
 9:     if Three consecutive unsuccessful attempts then
10:         break

    Function: GeneratingPrimitives (subset1, $J_{\max}$)
11: slow_workers, fast_workers = $J_{\max}$.GetWorkers()
12: if slow_workers is $\emptyset$ then
13:     squeeze(subset1)
14: else
15:     if $J_{\max} \in$ subset1 then
16:         swell(slow_workers)
17:         squeeze(fast_workers)
18:     else
19:         workers = GetColocatedWorker(slow_workers)
20:         squeeze(workers)
```

First, it identifies J_{max} and divides jobs into two subsets based on packing relationships, with subset2 as the high-priority set (lines 1-4). It then swells high-priority subsets, as jobs with higher priority contribute more to the scheduling objective (lines 5). Next, primitives are generated and mapped to the underlying mechanism (line 7), followed by the collection of updated runtime information (line 8). To minimize search duration, an *early exit* mechanism terminates the search after three consecutive plans yield no positive gains (lines 9-10). Function GeneratingPrimitives generates the calibration primitive combinations based on the current runtime information. It squeezes the low-priority workers. If J_{max} is low-priority and there are stragglers, it simultaneously squeezes the faster workers to achieve balance (lines 11-20).

The early exit mechanism terminates the search after three consecutive attempts yielding no gains. In a packed job set, the benefits of gradually squeezing a job initially increase before declining, creating a concave curve, as shown in Figure 2-(a). Thus, a continuous decline in gains signals that the best result has been achieved, allowing for an early exit to avoid unnecessary searches. The correctness and effectiveness of this mechanism will be presented in §7.4.

Searching algorithm complexity. The time complexity of Algorithm 1 is determined by the for loop at line 6. Each iteration of this loop corresponds to the activation of the underlying mechanism, consisting of mechanism switching and new runtime data collection. In the mechanism switching phase, for LISS, the switch involves only parameter changes and incurs no real overhead. In contrast, LGRR introduces additional overhead due to CUDA context creation and switching. As for collecting runtime information, Gimbal limits the process to a maximum of 60 seconds or 5 training steps. This ensures that data collection can be completed within a reasonable timeframe, regardless of job types and resource constraints. It should be noted that collecting new runtime information should not be considered overhead, as the training process continues uninterrupted.

Therefore, when analyzing the time complexity of Algorithm 1, we only need to consider the overhead introduced by LGRR. Denote the size of the job set as M. In the worst case, each worker's resources are squeezed from 100% to 0%, requiring 10 iterations under an LGRR granularity of 10% (§7.4). Since LGRR applies only to low-priority jobs, at most $M - 1$ jobs are involved. Moreover, the overhead for each calibration is stable (§7.4). The time complexity of the Algorithm 1 is $O(M)$.

Other scheduling objectives and more advanced policies. As for other schedulers with job packing, the cluster maintainer can support their objectives by directly defining the function of the local objective and job priority. Moreover, our current coordination policy is a greedy one. Maintainers can implement more advanced policies with the proposed calibration primitives.

6 IMPLEMENTATION

We implement a prototype of Gimbal using 6700 lines of Python code based on PyTorch [37]. The implementation breakdown includes 2200 lines for the scheduler, 920 lines for the aggressive packing coordinator, 1500 lines for the worker calibrator, 1500 lines for supporting mechanisms, and 1000 lines for other runtime components such as monitoring. Gimbal builds the CUDA context pool with 500 lines of C++ and integrates it into the framework to support low-overhead resource restriction. At runtime, we use gRPC [4] to communicate between the scheduler and the local workers on each server. Gimbal respects the scheduling manner of the upper scheduler (e.g., round-based in Gavel) and monitors the runtime status of training jobs every 30 seconds. Although Gimbal is implemented using PyTorch, it does not rely on any PyTorch-specific features. Its supporting mechanisms, such as LGRR, depend on the NVIDIA Driver API, while LISS can be implemented in major deep learning frameworks like Tensorflow [6]. This allows Gimbal's design to maintain broad compatibility with various machine learning platforms.

Low-overhead resource restriction through context pool. Gimbal utilizes Nvidia MPS [35] to provide resource restriction among co-located jobs. Nvidia MPS supports configuring multiple CUDA contexts with different resource ratios and switches the CUDA contexts for a job accordingly. However, integrating it directly into PyTorch is challenging. PyTorch currently only supports submitting the kernels to the primary CUDA context (100% resources). Reconfiguring the resource usage of a training job necessitates restarting the entire job. To tackle the above limitation, we rewrite the internal management of CUDA contexts and expose an API. This API allows Gimbal to initialize a CUDA context pool with varying resource configurations for calibration. The context is dynamically created to enable fine-grained resource restriction. Specifically, Gimbal creates a new context and releases the old one during switching context for calibration. After calibration, Gimbal retains only the chosen context. In this case, the memory overhead of maintaining the context pool is minimal—one extra CUDA context of 200 MB—since calibration is short and infrequent compared to the overall training process.

Data samplers for stealing input samples. Efficient data re-dispatching, which involves adjusting the local batch size for each worker within the global batch, is required. Gimbal extends the native data sampler in PyTorch to support this mechanism, enabling runtime data re-dispatching for workers. After each stealing operation, worker calibrators inform the global data sampler of the updated batch size distribution. Subsequently, the sampler adjusts the data dispatching rules for subsequent iterations. Specifically, we measure the forward computation latency of each worker to evaluate performance, as backward propagation does not accurately reflect computation time differences due to synchronous communication.

Table 1. Model and dataset configurations used in experiments.

Model	Dataset	Batch size
BERT[19]	SQuAD[41]	16,32
LLaMA[46]	Alpaca[45]	16,32
DeepSpeech2[7]	LibriSpeech[36]	8,16
ResNet-50 (RN50)[21]	ImageNet[18]	32,64,128
MobileNetV3 (MNV3)[22]	ImageNet[18]	32,64,128
VGG-11[43]	ImageNet[18]	32,64,128
VGG-16[43]	ImageNet[18]	32,64,128
ProxylessNas (PLNAS)[11]	ImageNet[18]	32,64,128
Once-for-All (OFA)[10]	ImageNet[18]	32,64,128
Resnet-18 (RN18)[21]	CIFAR-10[27]	32,64,128
MobileNetV2 (MNV2)[42]	CIFAR-10[27]	32,64,128
EfficientNet (EFN)[44]	CIFAR-10[27]	32,64,128

Table 2. Analysis of traces used in end-to-end experiments.

Model size	Number of jobs	Average number of GPUs per job
`Small`	43	2.39
`Medium`	86	2.41
`Large`	21	4.85

(a) Philly trace.

Model size	Number of jobs	Average number of GPUs per job
`Small`	112	1.98
`Medium`	132	2.03
`Large`	56	3.54

(b) Alibaba Trace

7 EVALUATION

7.1 Experimental Setup

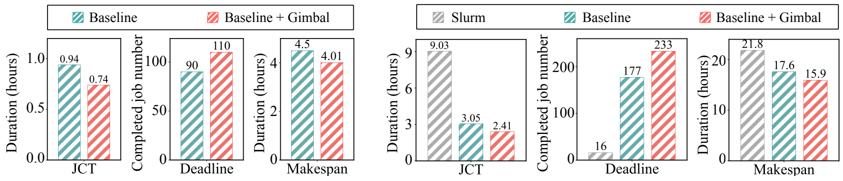
Testbed. All experiments are conducted on a cluster with 32 GPUs. Each node in the cluster is equipped with 2 Nvidia A40 GPUs (48GB memory), 48 CPU cores, and 256GB of CPU memory. These nodes are interconnected using Nvidia Mellanox InfiniBand ConnectX-5. Each node operates on Ubuntu 22.04 with CUDA 11.7 and cuDNN 7 installed. PyTorch 2.0 is used for the experiments.

Workloads. We use 12 representative models with various scales as our workloads. The model details are listed in Table 1. These workloads are categorized into three types based on their model/dataset size and resource requirements: (1) `Small` workloads include RN18, MNV2, and EFN models with the CIFAR-10 dataset; (2) `Middle` workloads consist of DeepSpeech2, RN50, MNV3, VGG-11, VGG-16, PLNAS, and OFA models using LibriSpeech and ImageNet datasets; (3) `Large` workloads involve LLM models such as BERT (for pre-train) and LLaMA (for fine-tune) with SQuAD and Alpaca datasets.

Traces. To evaluate Gimbal performance under different job distributions, we conducted comprehensive experiments using two real production-level traces: Philly [26] and Alibaba [49]. The experiments included 150 jobs from the Philly trace and 300 jobs from the Alibaba trace. Table 2 presents the trace analysis on the model and hardware configurations. In the traces, each job record includes submission time, job ID, and duration. Following prior research [34, 50], we randomly assign model types, GPU requirements, and calculated iteration counts based on the model's computation profiles. Meanwhile, we also generate the `large` workloads to resemble the longer-term jobs in production scenarios. In addition, job deadlines were generated by multiplying the duration with λ, plus submission time. λ ranges randomly from 1.5 to 2.5, aligning with established practices [20].

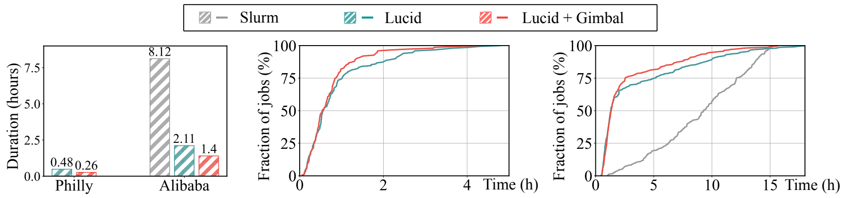
Baselines and scheduling objectives. There are many works that focus on optimizing the performance of scheduling deep learning jobs in the cluster. We use the following state-of-the-art systems as baselines.

- *Lucid* [23]: a state-of-the-art scheduler designed to enhance average JCT. It evaluates the sharing score of jobs using pre-profiled metrics to decide whether to co-locate them. We use it to conduct the experiments for the JCT.



(a) Philly trace with 150 jobs. (b) Alibaba trace with 300 jobs.

Fig. 8. End-to-end performance in terms of JCT, deadline satisfied ratio, and overall makespan. For these metrics, `Baseline` refers to Lucid, Gandiva, and Gavel, respectively.



(a) Average queuing time. (b) CDF of JCT with Philly trace. (c) CDF of JCT with Alibaba trace.

Fig. 9. In-depth analysis of JCT-oriented evaluation in terms of JCT distribution, average queuing time, and cluster throughput.

- *Gandiva* [50]: a First-Come-First-Serve (FCFS) scheduler that utilizes domain-specific knowledge to introspectively co-locate jobs based on runtime profiling and heuristic packing. We extend it to support the deadline-aware feature.
- *Gavel* [34]: a scheduler capable of accommodating diverse scheduling objectives by formulating and solving optimization problems. We select the makespan minimization as the scheduling policy in the experiments.

These baselines enhance resource utilization through job packing and are designed with specific scheduling objectives, leading to better performance compared to traditional deep learning schedulers, such as Slurm [5]. However, to provide a more comprehensive comparison, we also include Slurm in our evaluation under the Alibaba trace.

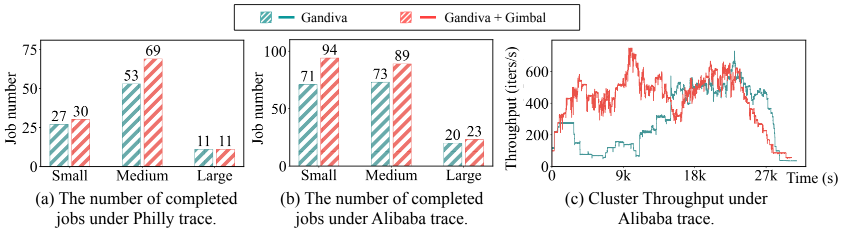
Methodology. We integrate Gimbal into the aforementioned schedulers and demonstrate its effectiveness by quantifying the performance improvement achieved after enabling Gimbal. To provide a comprehensive assessment of the enhancement, we evaluate the specific performance metrics targeted by each baseline scheduler: Lucid for JCT performance, Gandiva for deadline satisfied ratio[20], and Gavel for makespan.

Since the baseline schedulers are limited by the conservative nature of job packing, directly integrating into the schedulers could not unleash the potential of Gimbal. Therefore, while the baseline system operates in an original way, the scheduler enhanced with Gimbal is relaxed for more aggressive packing decisions. Detailed discussion for each baseline is demonstrated in §7.2.

7.2 End-to-end Performance

We evaluate the end-to-end performance of Gimbal on the 32-GPU physical testbed with a 150-job trace from Microsoft Philly trace and a 300-job trace from the Alibaba trace..

JCT. The original Lucid calculates a sharing score for each possible job packing combination based on the job metrics profiled offline. Based on this sharing score, it decides whether to co-locate the jobs. As for the enhanced Lucid with Gimbal, we adjust the score threshold to enable more aggressive job packing. Meanwhile, the original Lucid does not support the job packing with different GPU numbers. We also remove this restriction for the enhanced Lucid.



(a) The number of completed jobs under Philly trace.

(b) The number of completed jobs under Alibaba trace.

(c) Cluster Throughput under Alibaba trace.

Fig. 10. In-depth analysis of deadline-oriented evaluation in terms of the number of completed jobs and cluster throughput.

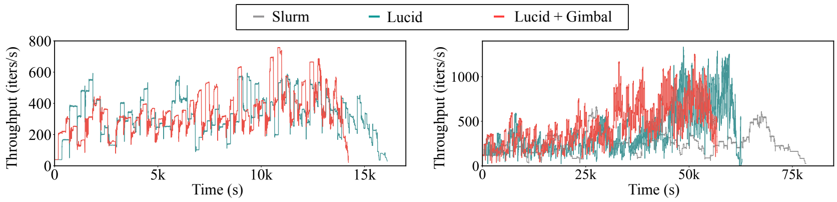
As shown in Figure 8-(a), the enhanced Lucid with GIMBAL reduces the average JCT by 1.28× and 1.26× on the Philly trace and Alibaba trace, respectively, compared to the original Lucid. Compared to Slurm, the enhanced Lucid with GIMBAL achieves a 3.75× reduction in average JCT on the Alibaba trace. This improvement can be attributed to the fine-grained worker calibration from GIMBAL. Although unequal packing and aggressive co-location may bring severe interference and straggler, GIMBAL could minimize the interference and eliminate the straggler to improve the system performance.

Figure 9 presents a detailed analysis of the source of improvement. In Figure 9-(a), the average queue time for all jobs is evaluated, showing that, compared to the baseline, enabling GIMBAL reduces the average queue time by 1.87× on the Philly trace and 1.51× on Alibaba trrace (reduces 5.8× compared with Slurm). As the JCT of jobs decreases, the queuing time for jobs is also mitigated. In Figure 9-(b) and (c) show the CDF distribution of job JCT for Philly and Alibaba trace, demonstrating that the enhancement primarily benefits long-term jobs. This is because Lucid schedules jobs based on a short-job-first approach, but its decisions are overly conservative, causing long-term jobs to wait. GIMBAL, on the other hand, can perform more aggressive packing, reducing the queuing time while ensuring that the performance of short-term jobs is not compromised through fine-grained control of packed jobs.

Deadline-aware. The original Gandiva first makes the job packing decision and collects the runtime metrics to check whether to reschedule the jobs. The original Gandiva does not support the job packing with different GPU numbers. We remove this restriction for the enhanced Gandiva with GIMBAL.

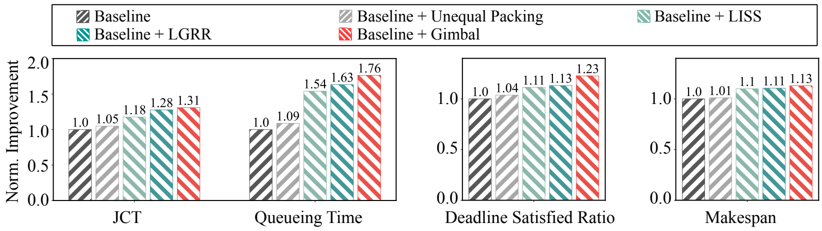
Figure 8 illustrates that the enhanced Gandiva with GIMBAL completes 110 out of 150 jobs in the Philly trace, whereas the original Gandiva completes only 90 jobs. In the Alibaba trace, GIMBAL accomplished 233 jobs, while the original Gandiva completed just 177 jobs, and the Slurm only finished 16 jobs due to its lack of support for deadline-aware scheduling. This represents a 1.22× and 1.32× increase in the job completion ratio for the Philly and Alibaba traces, respectively, over the same period. This improvement stems from GIMBAL's ability to adjust resources dynamically. For jobs with strict deadlines, GIMBAL supports restricting resource usage by another job at co-location. In contrast, the baseline system could not find such a co-location opportunity due to the lack of fine-grained resource management, which resulted in queued jobs awaiting scheduling.

Figure 10 offers further insights into why GIMBAL improves the deadline satisfaction ratio. In Figure 10-(a) and (b), it is evident that the enhanced Gandiva completes more jobs across three job types under Phiily and Alibaba trace. Meantime, Figure 10-(c) indicates that the enhanced Gandiva achieves higher cluster throughput, allowing more jobs to meet their deadline requirements. These improvements are all attributed to the more job packing opportunities and fine-grained resource adjustment brought by GIMBAL.



(a) Cluster Throughput under Philly trace. (b) Cluster Throughput under Alibaba trace.

Fig. 11. In-depth analysis of makespan-oriented evaluation in terms of cluster throughput.



(a) Lucid for JCT. (b) Gandiva for deadline-aware. (c) Gavel for makespan.

Fig. 12. Ablation study on (1) unequal packing, (2) low-level GPU resource restriction (LGRR), and (3) lossless input sample stealing (LISS).

Makespan. The original Gavel first makes the job packing decision using the optimization problems. Then, it schedules the highest-priority job pairs in a round-robin manner and continuously updates the priorities of all job pairs. Meantime, the original Gavel also does not support the job packing with different GPU numbers. We remove this restriction for the enhanced Gavel with Gimbal.

As shown in Figure 8, under the Philly trace, the original Gavel system takes 4.5 hours to complete all 150 jobs, while under the Alibaba trace, it takes 17.6 hours to complete 300 jobs. The enhanced Gavel with Gimbal reduces this time to 4.05 hours and 15.9 hours, respectively, resulting in a 1.12× and 1.1× enhancement for the overall makespan. Additionally, Figure 11 demonstrates that the enhanced Gavel could increase the peak cluster throughput, allowing for the jobs to complete computations in less time.

Gimbal shows a relatively modest improvement over the baseline in terms of makespan, especially when compared with other scheduling objectives. This can be attributed to two main reasons. In high-load clusters, utilization is already near its maximum, which makes further improvements to the makespan challenging. For example, Lucid's original makespan improvement is limited to 1.09×, while Gavel's is capped at 1.2×. Second, the primary limitation arises from Gavel's frequent rescheduling. This reduces opportunities for unequal packing, thereby limiting the effectiveness of LISS within Gimbal. Notably, a modest decrease in makespan can significantly reduce cluster operating costs by shortening overall runtime, especially in larger clusters.

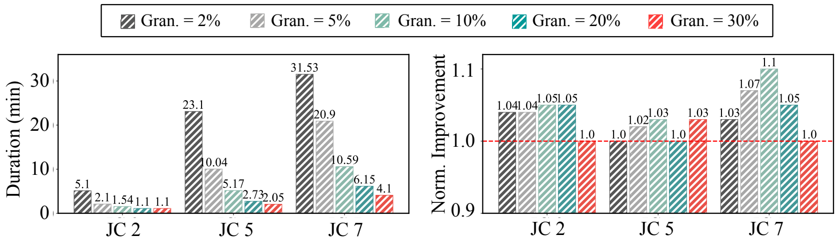
7.3 Ablation Study

We perform the ablation study for the two mechanisms introduced by Gimbal on the 16-GPU physical testbed with a 100-job trace from Microsoft Philly trace.

Unequal packing. We first evaluate the impact of integrating unequal packing directly into the baseline schedulers. As shown in Figure 12, the integration of unequal packing led to a 1.05× improvement in JCT (with queuing time enhancement by 1.09×), a 1.04× improvement in meeting deadlines, and a 1.01× improvement in makespan compared to the baseline. Although the unequal packing brings the straggler worker for the training job, it does not always bring a negative effect

Table 3. Representive job combinations used in the evaluation of calibration duration.

ID	Combinations
JC1	MobileNet_V2 (4GPUs) + EfficientNet (2GPUs) + ResNet-18 (2GPUs)
JC2	MobileNet_V2 (4GPUs) + EfficientNet (2GPUs) + VGG-11 (2GPUs)
JC3	ResNet-50 (4GPUs) + EfficientNet (2GPUs) + MobileNet_V2 (2GPUs)
JC4	ResNet-50 (4GPUs) + EfficientNet(2GPUs) + VGG-11 (2GPUs)
JC5	ResNet-50 (4GPUs)+ VGG-11 (2GPUs) + BERT (2GPUs)
JC6	LLaMA (8GPUs) + EfficientNet (4GPUs) + Mobilenet_v2 (4GPUs)
JC7	LLaMA (8GPUs) + ResNet-50 (4GPUs) + EfficientNet (4GPUs)
JC8	LLaMA (8GPUs) + ResNet-50(4GPUs) + VGG-11 (4GPUs)



a) Time to search for the optimal configuration.

b) Normalized average JCT.

Fig. 13. The three different job combinations under various search granularities: (a) the time required to search for the best performance, (b) the overall average JCT.

on the job. This is because the job co-location could gain performance throughput. Therefore, these systems with unequal packing obtain tiny improvements. These results are consistent with those in subsection 2.2.

Lossless input sample stealing. After integrating the unequal packing, we further enable the lossless input sample stealing mechanism for all the systems. As depicted in Figure 12, this mechanism yielded a 1.18× improvement in JCT (accompanied by a 1.54× improvement in queuing time), a 1.11× increase in the number of jobs meeting their deadlines, and a 1.1× enhancement in makespan. These improvements are largely attributed to the migration of batch sizes, which helps mitigate slowdown issues associated with unequal packing.

Low-level GPU resource restriction. Furthermore, we add the low-level GPU resource restriction mechanism in addition to the above two modules. As depicted in Figure 12, compared to the baseline, there is a 1.28× improvement in JCT (with a 1.63× improvement in queuing time), a 1.13× increase in the ratio of satisfied deadlines, and a 1.11× improvement in makespan. These improvements are attributed to the mechanism's ability to allocate resources more effectively for packed jobs based on scheduling objectives.

7.4 Coordination Analysis

In this section, we will discuss the hyperparameters in Gimbal's coordination algorithm, along with its search complexity. To this end, we carefully selected several representative job combinations. As shown in Table 3, these pairs nearly encompass all types of co-located workloads. We use JCT as the scheduling objective for evaluation and analysis.

Impact of different LGRR granularity. To explore the impact of different LGRR granularities (MPS resource percentages), we test three representative job combinations from Table 3: JC2, JC5, and JC7 with varied LGRR granularity. Figure 13 illustrates the corresponding results and the results of other job combinations align with this trend.

Setting a small granularity enables more precise configurations but leads to excessive search steps, increasing total overhead and degrading overall performance, as demonstrated by JC2 and

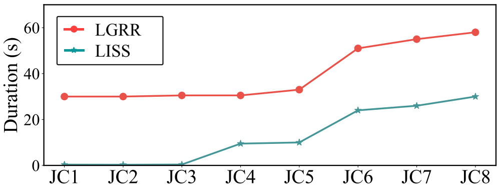
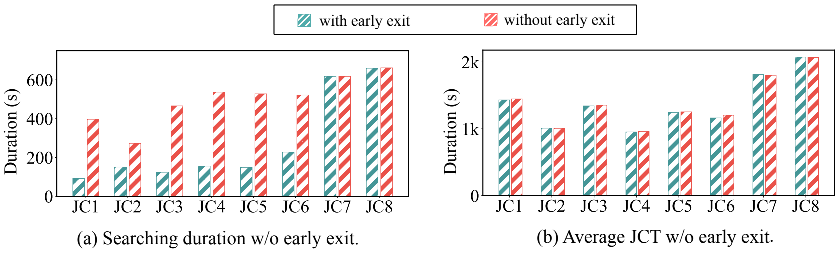


Fig. 14. Duration of LISS and LGRR for one search iteration.



(a) Searching duration w/o early exit.

(b) Average JCT w/o early exit.

Fig. 15. Search duration and average JCT of job combinations w/wo early exit mechanism

JC7 in Figure 13-(b). Conversely, overly large granularity can miss optimal points, such as a 30% in JC2 or 20% setting in JC7. Therefore, a granularity of 10% strikes a balance, reducing overhead while ensuring optimal performance points are not missed.

Duration of a single calibration iteration. We first investigate the effectiveness of Algorithm 1 by exploring the time for each calibration iteration with different mechanisms. Specifically, we test various job combinations with both LISS and LGRR applied to J_{max} (the first job). Figure 14 illustrates the total time per iteration for different job combinations under both mechanisms. It can be observed that the average duration for one iteration of LISS is 9 seconds, while LGRR averages 39 seconds. The longer duration for JC6, JC7, and JC8 is due to the involvement of LLaMA, a large model with relatively low training throughput, resulting in longer times for both mechanism switching and runtime information collection.

Analysis of real overhead and algorithm complexity. The above describes the total time for a single iteration. As mentioned in (§5.2), this includes the time for mechanism switching and runtime information collection, with overhead occurring only during LGRR switching. We test the overhead of LGRR mechanism switching across all benchmarks under various resource restrictions. The results show that the average overhead for switching once is 3.6 seconds. Moreover, it is very stable across all benchmarks, since workers are switching CUDA contexts in parallel.

Let M represent the number of jobs in a job set. As discussed in §5.2, in the worst-case scenario, Algorithm 1 requires $10 \times (M - 1)$ iterations. Therefore, the total overhead during the search process is $3.6 \times 10 \times (M - 1)$ *seconds*. It is evident that the overall search time and overhead are directly proportional to the number of jobs being packed.

When M is large enough, the benefits of the search may be outweighed by the overhead, leading to no net gains. In our end-to-end experiments, the maximum value of M observed is 5, under which Gimbal achieves the noted performance improvements. Therefore, when designing the upper-level scheduler with unequal packing, it is essential to set appropriate limits on the number of jobs in a job set, guided by practical considerations.

The effectiveness and correctness of early exit. Figure 15-(a) shows the time required to complete the search with and without the early exit mechanism. It can be observed that a complete search averages 480 seconds, while with early exit, this duration is reduced to 250 seconds, a decrease of 48%. Figure 15-(b) further evaluates the rationale behind the early exit mechanism, which should not significantly compromise the optimality of the searched restriction scheme. It is observed that with early exit, Gimbal achieves 99.4 % JCT performance compared to a full search.

This is because, in most cases, the benefits of conducting resource restriction schemes exhibit a convex trend, as shown in Figure 2-(a). The early exit mechanism's ability to quickly identify optimal performance points and terminate unnecessary searches.

8 RELATED WORK

Packing training jobs for better utilization. Many DL training schedulers [23, 34, 50, 51] aim to enhance GPU resource utilization through job packing. Gandiva [50] leverages online profiling information to greedily pack jobs on underutilized GPUs. Gavel [34] evaluates all candidate job pairs and employs an optimization approach to determine the optimal co-location. Lucid [23] packs jobs based on offline-profiled job scores. While they allow equal packing of packed jobs, Gimbal further expands the searching space by introducing unequal packing, and eliminates the performance stragglers through on-the-fly calibration.

Towards various scheduling objectives. Many existing schedulers are tailored to a singular scheduling objective. Synergy [33] prioritizes the optimization of JCT. ElasticFlow [20] targets maximizing the number of jobs that satisfy their deadline requirements. Gandiva$_{fair}$ [12] is dedicated to ensuring fairness and overall efficiency on heterogeneous clusters. Gimbal aligns with the objectives of the upper schedulers, further optimizing their scheduling decisions via calibration mechanisms.

Adaptive scheduling of training jobs. Many schedulers adaptively modify job hyperparameters to achieve performant scheduling objectives. Pollux [40] modifies batch size to unify metric modeling with resource scheduling for co-optimization. KungFu [30] supports user-defined policies that adjust hyperparameters based on runtime metrics. ONES [8] employs evolutionary algorithms to determine the optimal batch size for each job based on runtime execution. In contrast to them, Gimbal only employs non-intrusive optimizations and therefore provides a strict guarantee that the model convergence is not affected.

Other schedulers. Some schedulers focus specifically on host-side resource management for deep learning jobs. For instance, SiloD [54] identifies cache and remote I/O as critical scheduling factors, and Synergy [33] enhances job efficiency through adaptive CPU and memory management. In Gimbal, we assume ample CPU resources and use local disk storage for training purposes. In the future, we plan to integrate the insights gained from these approaches into Gimbal.

There are also many schedulers designed for traditional CPU jobs. Pegasus [2] is a workflow management system designed for complex scientific computations. Slearn [24] estimates the whole job's runtime properties based on a few sampled tasks. Hyungro Lee et al., 2024 [28] propose using data flow analysis to identify performance bottlenecks and opportunities for improvement. These works are orthogonal to Gimbal. Moreover, distributed DL training jobs typically run on GPUs in an SPMD (single process multiple data) mode. This setup makes performance estimation challenging, requiring offline or online profiling. Gimbal incorporates runtime profiling during coordination to monitor training performance.

Co-location of workloads Many researches focus on co-locating multiple applications on the same hardware. Co-location on CPUs has been studied in various contexts [15, 29, 38]. Heracles [29] proposed to co-locate latency-critical (LC) and best-effort (BE) jobs on the same server to improve resource utilization. It leverages offline profiled data and runtime monitoring proactively to avoid SLO (Service Level Objective) violations of LC jobs. PARTIES [15] further analyses the sensitivity of various LC services to system-shared resources for co-location. There are also works that focus on co-locating jobs on GPUs [14, 17, 53]. They are designed for different scenarios with latency-sensitive applications, such as deep learning inferences, cloud gaming, etc. Gimbal draws insights from the research above and tailors the co-location mechanism for deep learning training jobs, that are long-running and throughput-sensitive.

9 CONCLUSION

In this paper, we identify the root cause of job packing's conservative nature as the scope and granularity mismatch between job packing and cluster scheduling. We propose Gimbal, a novel job-packing middleware that locally and fine-grainedly coordinates the workers of packed jobs. The key of Gimbal is the calibration primitives, which adjust workers' execution status incrementally to achieve optimal packing efficiency. The primitives let cluster maintainers implement coordination policies of job packing for various dedicated DL schedulers without caring about complex resources and job management. The schedulers can aggressively select appropriate jobs for packing while leaving the tapping of job-packing potential to Gimbal. Our evaluation shows Gimbal can enhance the efficiency of popular schedulers by up to 1.32×.

ACKNOWLEDGMENTS

This work is partially sponsored by the National Key Research and Development Program of China (2023YFB3001504), the National Natural Science Foundation of China (62302302, 62232011), and Natural Science Foundation of Shanghai Municipality (24ZR1430500). Quan Chen is the corresponding author.

REFERENCES

[1] 2017. Nvidia Volta Architecture. https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/.

[2] 2019. Pegasus: Makes the Work Flow. https://pegasus.isi.edu/.

[3] 2020. Nvidia Ampere Architecture. https://www.nvidia.com/en-us/data-center/ampere-architecture/.

[4] 2023. gRPC: An RPC library and framework. https://grpc.io.

[5] 2024. Slurm workload manager. https://slurm.schedmd.com/documentation.html.

[6] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 265–283.

[7] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*. PMLR, 173–182.

[8] Zhengda Bian, Shenggui Li, Wei Wang, and Yang You. 2021. Online evolutionary batch size orchestration for scheduling deep learning workloads in GPU clusters *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 100, 15 pages. https://doi.org/10.1145/3458817.3480859

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[10] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *International Conference on Learning Representations*. https://arxiv.org/pdf/1908.09791.pdf

[11] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *International Conference on Learning Representations*. https://arxiv.org/pdf/1812.00332.pdf

[12] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. 2020. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.

[13] Chen Chen, Qizhen Weng, Wei Wang, Baochun Li, and Bo Li. 2020. Semi-dynamic load balancing: Efficient distributed learning in non-dedicated environments. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 431–446.

[14] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. 2016. Baymax: Qos awareness and increased utilization for non-preemptive accelerators in warehouse scale computers. *ACM SIGPLAN Notices* 51, 4 (2016), 681–696.

[15] Shuang Chen, Christina Delimitrou, and José F. Martínez. 2019. PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 107–120. https://doi.org/10.1145/3297858.3304005

[16] NVIDIA Corporation. 2023. NVIDIA Multi-Instance GPU User Guide. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html Accessed: 2024-06-23.

[17] Weihao Cui, Han Zhao, Quan Chen, Ningxin Zheng, Jingwen Leng, Jieru Zhao, Zhuo Song, Tao Ma, Yong Yang, Chao Li, and Minyi Guo. 2021. Enable simultaneous DNN services based on deterministic operator overlap and precise latency prediction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 15, 15 pages. https://doi.org/10.1145/3458817.3476143

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[20] Diandian Gu, Yihao Zhao, Yinmin Zhong, Yifan Xiong, Zhenhua Han, Peng Cheng, Fan Yang, Gang Huang, Xin Jin, and Xuanzhe Liu. 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.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[22] Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming Pang, Hartwig Adam, and Quoc Le. 2019. Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 1314–1324. https://doi.org/10.1109/ICCV.2019.00140

[23] Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2023. Lucid: A Non-intrusive, Scalable and Interpretable Scheduler for Deep Learning Training Jobs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 457–472.

[24] Akshay Jajoo, Y. Charlie Hu, Xiaojun Lin, and Nan Deng. 2022. A Case for Task Sampling based Learning for Cluster Job Scheduling. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 19–33. https://www.usenix.org/conference/nsdi22/presentation/jajoo

[25] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R Ganger. 2023. Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 642–657.

[26] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 947–960. https://www.usenix.org/conference/atc19/presentation/jeon

[27] Alex Krizhevsky. 2023. The CIFAR10 Dataset. https://cs.toronto.edu/~kriz/cifar.html.

[28] Hyungro Lee, Luanzheng Guo, Meng Tang, Jesun Firoz, Nathan Tallent, Anthony Kougkas, and Xian-He Sun. 2023. Data Flow Lifecycles for Optimizing Workflow Coordination. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. Association for Computing Machinery, New York, NY, USA, Article 58, 15 pages. https://doi.org/10.1145/3581784.3607104

[29] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '15)*. Association for Computing Machinery, New York, NY, USA, 450–462. https://doi.org/10.1145/2749469.2749475

[30] Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis, Andrei-Octavian Brabete, and Peter Pietzuch. 2020. KungFu: Making Training in Distributed Machine Learning Adaptive. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 937–954. https://www.usenix.org/conference/osdi20/presentation/mai

[31] Meta. 2024. Building Meta's GenAI infrastructure. https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/ Accessed: 2024-06-23.

[32] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Looking beyond GPUs for DNN scheduling on Multi-Tenant clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 579–596.

[33] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Looking Beyond GPUs for DNN Scheduling on Multi-Tenant Clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 579–596. https://www.usenix.org/conference/osdi22/presentation/mohan

[34] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*. 481–498.

[35] NVIDIA Corporation. 2023. NVIDIA Multi-Process Service Documentation. https://docs.nvidia.com/deploy/mps/ Accessed: 2024-06-23.

[36] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: An ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 5206–5210. https://doi.org/10.1109/ICASSP.2015.7178964

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv preprint arXiv:1912.01703* (2019).

[38] Tirthak Patel and Devesh Tiwari. 2020. CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 193–206. https://doi.org/10.1109/HPCA47549.2020.00025

[39] Princeton University. 2024. Princeton invests in new 300-GPU cluster for academic AI research. https://ai.princeton.edu/news/2024/princeton-invests-new-300-gpu-cluster-academic-ai-research Accessed: 2023-10-12.

[40] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R Ganger, and Eric P Xing. 2021. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning.. In *OSDI*, Vol. 21. 1–18.

[41] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

[43] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[44] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6105–6114. https://proceedings.mlr.press/v97/tan19a.html

[45] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[46] 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).

[47] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[48] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *19th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 22)*.

[49] Qizhen Weng, Lingyun Yang, Yinghao Yu, Wei Wang, Xiaochuan Tang, Guodong Yang, and Liping Zhang. 2023. Beware of Fragmentation: Scheduling GPU-Sharing Workloads with Fragmentation Gradient Descent. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 995–1008. https://www.usenix.org/conference/atc23/presentation/weng

[50] 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 *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 595–610.

[51] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning.. In *OSDI*. 533–548.

[52] Gingfung Yeung, Damian Borowiec, Renyu Yang, Adrian Friday, Richard Harper, and Peter Garraghan. 2021. Horus: Interference-aware and prediction-based scheduling in deep learning systems. *IEEE Transactions on Parallel and Distributed Systems* 33, 1 (2021), 88–100.

[53] Wei Zhang, Binghao Chen, Zhenhua Han, Quan Chen, Peng Cheng, Fan Yang, Ran Shu, Yuqing Yang, and Minyi Guo. 2022. PilotFish: Harvesting Free Cycles of Cloud Gaming with Deep Learning Training. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 217–232. https://www.usenix.org/conference/atc22/presentation/zhang-wei

[54] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Mingxia Li, Fan Yang, Qianxi Zhang, Binyang Li, Yuqing Yang, Lili Qiu, Lintao Zhang, and Lidong Zhou. 2023. SiloD: A Co-design of Caching and Scheduling for Deep Learning Clusters. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 883–898. https://doi.org/10.1145/3552326.3567499

[55] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Fan Yang, Lidong Zhou, Mao Yang, Francis CM Lau, Yuqi Wang, Yifan Xiong, et al. 2020. HiveD: Sharing a GPU cluster for deep learning with guarantees. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation.* 515–532.

Received 20 May 2024