ASTRAEA: A Fair Deep Learning Scheduler for Multi-tenant GPU Clusters

Zhisheng Ye, Peng Sun, Wei Gao, Tianwei Zhang, *Member, IEEE,* Xiaolin Wang, *Member, IEEE,* Shengen Yan, and Yingwei Luo, *Member, IEEE*

Abstract—Modern GPU clusters are designed to support distributed Deep Learning jobs from multiple tenants concurrently. Each tenant may have varied and dynamic resource demands. Unfortunately, existing GPU schedulers fail to thoroughly consider the fairness among the tenants and jobs, which can result in unbalanced resource allocation and unfair user experience. In this paper, we present an efficient solution to provide strong fairness while maintaining high scheduling effectiveness in multi-tenant GPU clusters. First, we introduce a novel Long-Term GPU-time Fairness metric, which can comprehensively evaluate the fairness at both the tenant and job levels, based on both the temporal and spatial impacts of resource allocation. Second, we design a new and practical GPU scheduler, ASTRAEA, to enforce the desired fairness among tenants and jobs. Large-scale evaluations show that ASTRAEA can improve tenant fairness by up to 9.42× compared to state-of-the-art schedulers, without sacrificing the average job completion time.

Index Terms—Distributed systems, Deep Learning, GPU Cluster Scheduling.

1 INTRODUCTION

DEEP learning (DL) has been utilized in a wide range of real-world applications, e.g., image recognition, natural language processing, recommendation systems [1]. State-of-the-art DL models are trained from massive data samples, involving a large amount of computations. It is a common practice to leverage GPUs or other accelerators to speed up the training process [2], [3]. To deal with the ever-growing complexity of DL training (DLT) workloads and increased demands for computation resources, enterprises and institutes commonly set up large-scale GPU clusters.

A cluster is typically shared by multiple user groups (i.e., tenant), which can significantly improve resource utilization and reduce operational costs [4], [5]. It can concurrently serve a large number of jobs with different features. Table 1 shows the running time distribution of DLT jobs from two production GPU clusters operated by Microsoft and SenseTime. We observe that there are long-term jobs which take hours or days to complete. Besides, about 41% (Philly) and 67% (Venus) of the DLT jobs can be completed within 10 minutes. These short-term jobs generally adopt the feedback-driven exploration method for the research and debugging purposes [6], [7].

The mixed workloads of long-term and short-term jobs call for fair resource allocation, and it is crucial to provide *sharing incentive* in a shared cluster. Here sharing incentive refers to the property that if N jobs share a cluster, then each job should not have worse performance compared to an independent cluster with $1/N$ of the resources [8]. On one hand, long-term jobs should not monopolize the entire cluster, which can significantly block the execution of short-term jobs (a.k.a. *head-of-line blocking* [6], [7]). On the other hand, arbitrarily prioritizing short-term jobs can cause starvation for long-term jobs. It is necessary to balance all these types of jobs and satisfy every tenant. Unfortunately, most of the existing DLT schedulers, e.g., Slurm [9], Kubernetes [10], Yarn-CS [4], Tiresias [7], Gandiva [6], Hived [11] and Pollux [12] ignore the fairness issue in GPU clusters, which can result in severe discrimination of resource allocation across DLT jobs. $Gandiva_{fair}$ [13] targets the fairness of GPU clusters. However, it mainly considers fair share of GPU resources across tenants instead of DLT jobs.

TABLE 1: Running time distributions of DLT jobs in two GPU clusters: Philly from Microsoft and Venus from SenseTime.

Time	<10 min	<1 hour	<6 hour	<1 day	<6 day
Philly	40.8%	71.6%	88.5%	94.1%	98.5%
Venus	67.3%	83.3%	90.6%	96.2%	99.6%

This paper aims to *design new scheduling solutions to achieve fairness for DLT jobs in multi-tenant GPU clusters.* Past works have studied similar problems in the big-data cluster environment. The mainstream mechanism is *instantaneous resource-allocation fairness*, which can ensure instantaneous fair allocations across big data jobs (e.g., map-reduce tasks) by balancing the allocated resources among all the jobs in real-time [4], [5], [14], [15], [16]. Unfortunately, it is difficult to achieve instantaneous fairness for DLT jobs in GPU clusters, due to their distinct characteristics. (1) Mainstream DL frameworks [17], [18] require gang scheduling, i.e., all the requested GPUs should be offered to the job in an all-

- *Z.Ye is with School of Computer Science, Peking University, Beijing 100871, China, and also with Peng Cheng Laboratory, Shenzhen 518000, China and SenseTime Research, China. E-mail: yezhisheng@pku.edu.cn.*
- *P. Sun and S. Yan are with SenseTime Research, China. E-mail: {sunpeng1,yanshengen}@sensetime.com.*
- *W. Gao and T. Zhang are with School of Computer Science and Engineering, Nanyang Technological University, and also with S-Lab, Nanyang Technological University, Singapore 639798. E-mail: {gaow0007,tianwei.zhang}@ntu.edu.sg.*
- *X. Wang and Y. Luo are with School of Computer Science, Peking University, Beijing 100871, China, and also with Peng Cheng Laboratory, Shenzhen 518000, China. E-mail: {wxl,lyw}@pku.edu.cn. (Corresponding author: Y. Luo.)*

or-nothing manner. Under this restriction, it is infeasible to split partial resources from one job and give them to others for instantaneous fairness. (2) DLT jobs have high GPU affinity requirements. The performance of communication-intensive DLT jobs exhibits high sensitivity to inter-GPU topography. The instantaneous resource-allocation fairness mechanism cannot take this factor into consideration. (3) The preemption overhead of a DLT job is large for saving and loading the model. Frequent preempting DLT jobs for instantaneous fairness can cause huge performance penalty.

Hence, it is necessary to design a fairness solution dedicated to DLT workloads. Themis [8] made such an attempt by proposing the *long-term finish-time fairness* metric for DLT jobs. It considers the gang scheduling and GPU affinity requirements, and leverages a lease-based scheduling to handle the preemption overhead of DLT jobs. At each scheduling round, this mechanism tries to figure out the optimal resource allocation strategy based on the remaining time of each job. However, the metric only measures fairness in terms of completion time without considering different GPU requirements of DLT jobs. This can disincentive small DLT jobs with smaller GPU requirements from using the shared cluster. More seriously, a dishonest user may submit DLT jobs with overclaimed GPU demands to intentionally get higher throughputs with the same scheduling priority, which could lead to more serious waste of cluster resources and exacerbate the unfairness issue. We will give detailed analysis about these mechanisms in Section 4.1.

Motivated by the above limitations, we present a novel solution for fair GPU resource allocation across DLT workloads. We make the following two key contributions. First, we design a new metric, Long-Term GPU-Time Fairness (`LTGF`), to comprehensively characterize and evaluate the fairness of DLT jobs. The comprehensiveness of this metric is reflected from two aspects. (1) `LTGF` considers *both the temporal and spatial impacts of a resource allocation on the fairness of DLT jobs*. At a long-term scale, it quantifies the gap between the amount of resources allocated to the job and the amount of resources the job deserves to obtain under the sharing incentive. This can give more reasonable assessment than the finish-time fairness in Themis, which only considers the temporal factor. (2) `LTGF` considers *fairness at both the tenant and job levels*. At the tenant level, it distributes to different tenants fair amounts of GPU resources, which are proportional to their contributions (e.g., budget or GPU resources) over a period of time. At the job level, it ensures that GPU time is fairly allocated among concurrent DLT jobs in the long term within a tenant. This can provide guaranteed service for tenants, as well as high cluster utilization.

Second, we design ASTRAEA[1], a practical and efficient scheduling system to achieve fairness in GPU clusters. It can simultaneously ensure fair share of GPU resources among tenants as well as jobs without bringing noticeable overhead for DLT jobs. Specifically, ASTRAEA adopts the `LTGF` metric to evaluate fairness and identify the optimal resource allocation decisions. It also utilizes a lease-based scheduling scheme, which can efficiently preempt running jobs to balance fairness and Job Completion Time (JCT). Evaluation shows that ASTRAEA outperforms state-of-the-art fair schedulers, and improves tenant-level fairness by up to 9.42× and job fairness by up to 10.3× without sacrificing the average JCT.

1. ASTRAEA is the name of a Greek goddess for justice.

2 BACKGROUND

2.1 Multi-Tenant GPU Cluster

It becomes popular to operate the GPU cluster in a multi-tenant fashion. This can bring many benefits, such as reducing operational cost and improving resource utilization. A shared GPU cluster is divided into multiple Virtual Clusters (VCs), with each one assigned to a tenant. Based on a tenant's resource weight, its VC is associated with a static or dynamic resource quota in terms of the number of GPUs and other resources (e.g., CPU, RAM) [19]. Users of a tenant can only submit DLT jobs to their corresponding VC. A cluster scheduler is in charge of selecting DLT jobs from the pending job pool and placing them on GPUs for execution.

2.2 Deep Learning Training

A Deep Learning training (DLT) job aims to determine the optimal parameter values of a Deep Learning model from training data. Users submit their jobs to the cluster. Then a scheduler selects appropriate jobs from the pending queue and places them to the demanded GPU resources for execution. A running DLT job may be preempted by other jobs. There are three possible final states for a DLT job: (1) COMPLETED: a DLT job successfully executes all the epochs of its training procedure. (2) CANCELED: users may manually early stop the execution of a DLT job based on the intermediate results. (3) FAILED: DLT jobs could be terminated by force due to programming or hardware issues [19]. DLT jobs have following features.

Iterative computing. A DLT job is executed in an iterative fashion. At each iteration, a batch of training samples are selected as the input of two steps of computation: forward and backward propagation. A DLT job may conduct millions of iterations to achieve model convergence, resulting in long running time (e.g., tens of days) [19], [20]. It is common to leverage hardware accelerators (e.g., GPU) to improve the computation efficiency.

Feedback-driven exploration. Users usually perform exploration tasks (e.g., searching different configurations or hyper-parameters, selecting the optimal network structures) in a trial-and-error manner before obtaining the final model [6], [7]. This exploration can be performed by hyperparameter-tuning frameworks [21], [22]. During this exploration process, poor-performing jobs will be killed in advance based on the intermediate results.

Gang scheduling. Distributed DLT jobs could use multiple GPUs to train a single model using data-parallel or model-parallel techniques. They usually require gang scheduling: all the requested GPUs should be offered in an all-or-nothing manner. While some new training solutions [23] with elastic GPU resources have been proposed, they are still at an early stage without broad deployment.

Placement sensitivity. Distributed training procedure needs to exchange data between GPUs at every iteration. To guarantee training performance, communication-sensitive jobs require strict topology of allocated GPUs. For example,

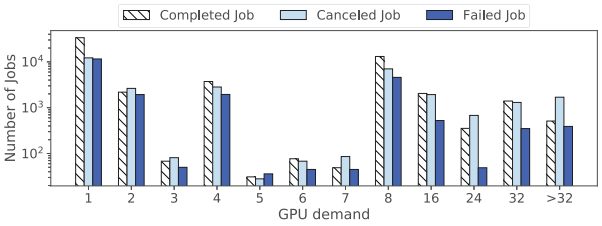


Fig. 1: Distribution of requested GPU resources.

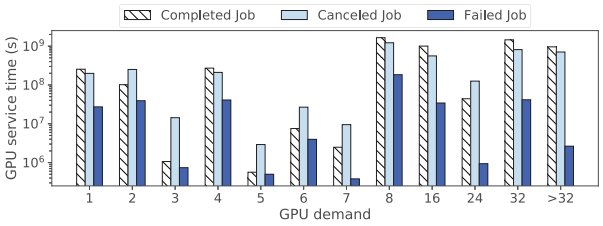


Fig. 2: Distribution of DLT job GPU service time.

a job with 8 GPUs should be placed on a single 8-GPU node for the high performance of the nvlink-based communication. If this job is allocated with two nodes, the inter-node communication could be the performance bottleneck [3].

2.3 DLT Job Scheduling

A scheduler is indispensable in a cluster to manage the computing resources and schedule jobs. At runtime, the scheduler continuously receives jobs submitted by users with the explicit resource demands. Then it decides when and where to run by selecting the proper jobs to satisfy the scheduling objectives and placing them to the appropriate servers for execution [24]. Different scheduling systems may pursue various scheduling objectives, e.g., improving resource utilization [5], [20], [25], optimizing the performance of workloads [26], [27], maximizing scheduling efficiency [14], [28], guaranteeing service and user experience [29], [30], etc. Different from the traditional big-data or HPC workloads, DLT jobs exhibit unique characteristics (Section 2.2). This brings new challenges for GPU cluster scheduling, and it is necessary to design dedicated schedulers for DLT jobs, which will be discussed in Section 7.2.

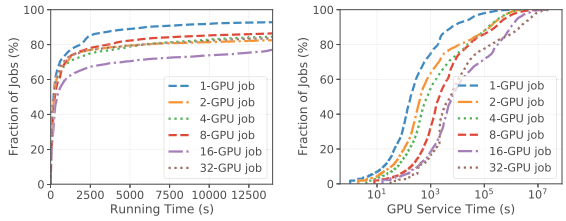
3 A Study of Real-World DLT Job Trace

In this section, we study Venus [20], a production DLT cluster in SenseTime. Venus contains 133 GPU servers connected by RDMA network. Each server is installed with 8 NVIDIA Volta GPUs. The cluster is shared among 16 tenants. Each tenant is allocated with a static number of GPUs. Venus uses Slurm [9] as the scheduler. We collect and analyze 109734 GPU-based DLT jobs over a period of 6 months.

3.1 Characteristics of DLT Jobs in Shared Clusters

Unpredictable Training Time. In Venus, 52.2% of jobs are successfully completed, 27.9% are canceled, and 19.9% fail. Due to the high cancellation/failure ratio, it is difficult to predict the running time of DLT jobs just based on the remaining number of iterations. [31].

Various GPU Demands. Figure 1 shows the distribution of DLT job in terms of requested GPU resources. About 52.5% of jobs use single GPU, 22.6% require 8 GPUs, and 10.3% need more than 8 GPUs. Jobs have high cancellation/failure ratio in all the cases. We define the *GPU service time* of a DLT job as the product of its GPU demand and running time. Figure 2 shows such distribution. While single-GPU jobs account for the largest proportion of all the jobs, they only consume 4.7% of total GPU service time. The major GPU service time is consumed by DLT jobs with no less than 8 GPUs (85.7%).



(a) CDF of running time. (b) CDF of GPU service time.

Fig. 3: CDF of DLT job running time and GPU service time.

Mixed Workload of Long-Term and Short-Term Jobs. Table 1 demonstrates that DLT job running time varies from minutes to days. Figure 3 further shows the cumulative distributions of job running time and service time with different resource demands. We observe that DLT job running time follows the long-tail distribution. For example, for single-GPU jobs, 87.4% of jobs last for less than 1 hour, while 2.2% spend more than 1 day. In addition, DLT jobs with larger GPU demands tend to have longer running time.

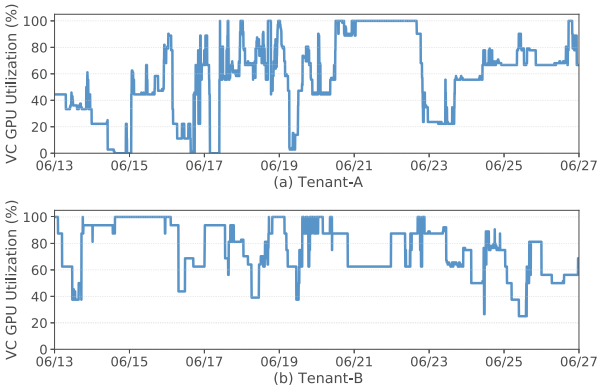
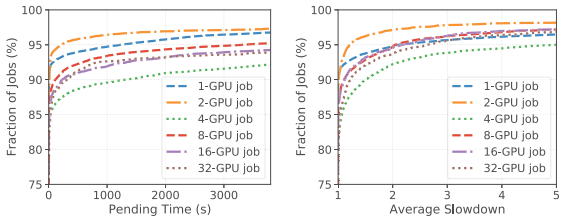


Fig. 4: GPU utilization of two VCs.

Unbalanced Resource Demands. The GPU resource demands of VCs in a shared DLT cluster are unbalanced over time. Figure 4 gives the VC utilization of two tenant. On June 15, Tenant-B has a high cluster GPU utilization of over 90%, but Tenant-A has an average utilization of less than 50%. On June 21, Tenant-A achieves near 100% utilization, while Tenant-B only uses 60% of its GPUs.



(a) CDF of job pending time. (b) CDF of job slowdown ratio.

Fig. 5: CDF of DLT job pending time and slowdown ratio.

Implication #1: The unpredictability of training time, and high variety in the GPU demands and running time significantly increase the difficulty of making optimal scheduling decisions and allocating resources.

3.2 Issues of DLT Clusters Without Fairness

Venus uses FIFO (First-In-First-Out) to schedule jobs without considering fair resource allocation across parallel DLT jobs. Long-term jobs may monopolize the whole cluster or VC, blocking the execution of pending jobs (a.k.a. head-of-line blocking). As shown in Figure 5(a), a portion of jobs have extremely long pending time. For example, 4.4% of single-GPU jobs wait for over 30 minutes to be scheduled. Jobs with 4 or more GPUs may have higher pending time. Blocked DLT jobs (especially for short-term jobs) may suffer from large execution slowdown ratio, which is defined as $(running\_time+pending\_time)/running\_time$. As shown in Figure 5(b), a certain number of jobs have large (> 5) slowdown ratios.

Implication #2: Scheduling without considering fairness could worsen the job pending situation and result in unfair execution across different jobs.

4 Long-Term GPU-Time Fairness

In this section, we analyze the limitations of existing fairness mechanisms, and then present our new fairness metric. This will further inspire the design of our fair scheduler.

We consider two types of sharing incentive in this work.

Definition 1. *(Sharing incentive for DLT jobs) Consider a M-GPU cluster hosting N DLT jobs with the same weight. We say there is sharing incentive for a job if its performance is no worse than the situation where it runs in an independent cluster of M/N GPUs.*

Definition 2. *(Sharing incentive for tenants) Consider a M-GPU cluster shared by T tenants with the same weight. We say there is sharing incentive for a tenant, if he could be allocated with at least the same amount of GPU service time as an independent cluster of M/T GPUs over a period of time, regardless of the workloads of other tenants.*

4.1 Analysis of Existing Fairness Mechanisms

We give some illustrative examples to demonstrate the limitations of two common fairness mechanisms, as well as the innovation and superiority of our metric.

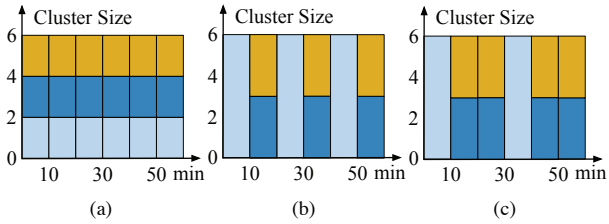


Fig. 6: Illustration of fair resource allocation. (a) Instantaneous resource-allocation fairness. (b) Long-term finish-time fairness. (c) Our proposed Long-term GPU-Time fairness.

4.1.1 *Instantaneous Resource-allocation Fairness*

This mechanism is widely adopted by existing big-data clusters to achieve instantaneous fairness [4], [5], [14], [15], [16]. Consider there are N active jobs in the cluster for scheduling at one instant t. Each job i receives $r_i(t)$ resources from the scheduler. Then the goal of the instantaneous resource-allocation fairness is to figure out the optimal policy with the following objective:

$$\texttt{maximize}: \min_{i \in N} r_i(t).$$

If a job does not fully utilize the allocated resources, these excess resources will be redistributed to other jobs.

Figure 6(a) gives an example of this mechanism. Consider a cluster of 6 CPUs with three active jobs. The scheduler should allocate these CPUs equally to each job regardless of their types or demands. Following this principle, the scheduler is able to provide *instantaneous* fairness across all the jobs at any time.

Unfortunately, instantaneous resource-allocation fairness is hard to achieve for DLT jobs. As discussed in Section 1, DLT jobs require gang scheduling with high placement sensitivity and preemption overhead. These characteristics make it inflexible for the scheduler to adjust the resource allocation instantaneously to guarantee fairness.

4.1.2 *Long-term Finish-time Fairness.*

This metric was proposed in [8] to address the fairness issue for Hyper-Parameter Optimization (HPO) workloads. It allocates resources to balance the finish time of each HPO job. It is defined as $\rho_i(t) = T_i(t)/(T_i^*(t) \cdot N)$, where $T_i(t)$ and $T_i^*(t)$ is the estimated time to finish job i using the actual allocated resources and the entire cluster, respectively. At each scheduling round, the scheduler tries to identify the optimal solution with the following objective:

$$\texttt{minimize}: \max_{i \in N} \rho_i(t).$$

Figure 6(b) gives an example when directly using this metric to schedule DLT jobs. Three DLT jobs are submitted to a 6-GPU cluster. *Job1* requires 6 GPUs while *Job2* and *Job3* require 3 GPUs. Each of them needs 40 minutes to complete. The duration of one scheduling round is 10 minutes. Guided by this long-term finish-time fairness principle, these jobs will be scheduled alternatively until their completion.

This metric only considers fairness in terms of completion time, while ignoring the amount of allocated resources. In Figure 6(b), for each period of 20 minutes, *Job1* gets $1/2$ of the total GPU resources. In contrast, *Job2* and *Job3* only

get 1/4 of the resources each. This cluster loses sharing incentive for these two jobs. Moreover, since jobs with higher GPU demands can get more GPU service time, tenants may overclaim excessive resources for their jobs to increase the throughput, resulting in a waste of GPU resources.

A fair scheduler should consider the spatial information (GPU demand) in addition to the temporal information (running time) of DLT jobs. Using the above example, the scheduler should give one round for *Job1* and two rounds for *Job2* and *Job3* (Figure 6(c)). Then equal resources are allocated to each running job in any half hour. Our proposed metric can achieve this fairness, as described below.

4.2 Long-Term GPU-time Fairness Definition

We propose a novel metric, Long-Term GPU-time Fairness (`LTGF`) for DLT scheduling. In a single-tenant cluster, `LTGF` achieves sharing incentive for DLT jobs. In a multi-tenant cluster, `LTGF` also provides sharing incentive for tenants.

4.2.1 LTGF in Single-Tenant Clusters

Consider a single-tenant cluster of M GPUs shared by N jobs from time t_1 to t_2. Let $Demand_i^{job}$ be the GPU demand of job J_i, $Alloc_i^{job}(t)$ be the amount of GPUs allocated to J_i at time t. Due to the gang scheduling feature, we have:

$$Alloc_i^{job}(t) = \begin{cases} 0, & \text{if } J_i \text{ is not running} \\ Demand_i^{job}, & \text{if } J_i \text{ is running} \end{cases}. \quad (1)$$

Over a period time of $[t_1, t_2]$, the accumulated GPU service time allocated to J_i by the scheduler is

$$\overline{Alloc}_i^{job}(t_1, t_2) = \int_{t_1}^{t_2} Alloc_i^{job}(t)\, dt. \quad (2)$$

We use $Active(J_i, t)$ to denote the state of Job J_i at time t: pending or running jobs have $Active(J_i, t) = 1$ while failed or completed jobs have $Active(J_i, t) = 0$. Given the share weight W_i^{job} of J_i, based on the max-min fairness, its instantaneous fair number of allocated GPUs at time t is

$$FairShare_i^{job}(t) = \frac{M \times W_i^{job}}{\sum_{j=1}^{N}(Active(J_i, t) \times W_j^{job})}. \quad (3)$$

Due to gang scheduling, J_i could not use more GPUs than $Demand_i^{job}$ even if $FairShare_i^{job}(t)$ has a large value. The fair GPU service time of J_i over $[t_1, t_2]$ is

$$\overline{FairShare}_i^{job}(t_1, t_2) = \int_{t_1}^{t_2} \min(Demand_i^{job}, FairShare_i^{job}(t))\, dt. \quad (4)$$

The fairness degree $\rho_i^{job}(t_1, t_2)$ for the job i from t_1 to t_2 is defined as

$$\rho_i^{job}(t_1, t_2) = \frac{\overline{Alloc}_i^{job}(t_1, t_2)}{\overline{FairShare}_i^{job}(t_1, t_2)}. \quad (5)$$

$\rho_i^{job}(t_1, t_2) \geq 1$ implies good long-term fairness in terms of GPU service time for job J_i. In contrast, $\rho_i^{job}(t_1, t_2) < 1$ indicates unfairness since it violates the sharing incentive. For example, we assume all the jobs have the same weight and keep active over $[t_1, t_2]$. The GPU service time that J_i receives in the shared cluster is $\rho_i^{job}(t_1, t_2)(t_2 - t_1)\min(Demand_i^{job}, M/N)$. Considering an independent cluster of M/N GPUs, the GPU service time that J_i acquires is $(t_2 - t_1)\min(Demand_i^{job}, M/N)$. It is noted that J_i could not run with M/N GPUs if $Demand_i^{job} > M/N$ due to the gang scheduling. The performance of J_i in a shared cluster is no worse than the situation where it runs in an independent cluster of M/N GPUs, if $\rho_i^{job}(t_1, t_2) \geq 1$.

4.2.2 LTGF in Multi-Tenant Clusters

Consider a cluster of M GPUs shared by T tenants. It is divided into T VCs, with each one assigned to a tenant. Each tenant j is associated with a weight W_j^{tenant}. Then the GPU quota of this tenant $Quota_j$ is calculated as:

$$Quota_j = \frac{M \times W_j^{tenant}}{\sum_{k=1}^{T}(W_k^{tenant})}. \quad (6)$$

Let N_j be the number of jobs for tenant j, and $Demand_{i,j}^{job}$ be the GPU demand of $J_{i,j}$ from this tenant. The requested amount of GPUs by tenant j at any time is

$$Demand_j^{tenant} = \sum_{i=1}^{N_j}(Active(J_{i,j}, t) \times Demand_{i,j}^{job}). \quad (7)$$

If $Demand_j^{tenant}(t) < Quota_j(t)$, the tenant would not use more GPUs than its resource demands. The fair number of allocated GPUs for tenant tenant j at time t is

$$FairShare_j^{tenant}(t) = \min(Demand_j^{tenant}(t), Quota_j(t)). \quad (8)$$

The fair GPU service time that tenant j should receive over a period of $[t_1, t_2]$ is

$$\overline{FairShare}_j^{tenant}(t_1, t_2) = \int_{t_1}^{t_2} FairShare_j^{tenant}(t)\, dt. \quad (9)$$

As shown in Section 3, the total GPU demands of tenants are unbalanced and varying over time. The actual amount of GPUs allocated to tenant j at time t is

$$Alloc_j^{tenant}(t) = \sum_{i=1}^{N_j}(Alloc_{i,j}^{job}(t)), \quad (10)$$

where $Alloc_{i,j}^{job}(t)$ denotes the amount of GPUs allocated to job $J_{i,j}$ at time t (Equation 1).

When $Alloc_j^{tenant}(t) < Quota_j$, the tenant does not fully utilize its resource quota due to insufficient workloads or GPU resource fragmentation. In contrast, $Alloc_j^{tenant}(t) > Quota_j$ means the tenant uses more resources than its quota to improve the overall cluster utilization.

Over a period of $[t_1, t_2]$, the actual GPU service time allocated to tenant j by the scheduler is

$$\overline{Alloc}_j^{tenant}(t_1, t_2) = \int_{t_1}^{t_2} Alloc_j^{tenant}(t)\, dt. \quad (11)$$

The fairness degree $\rho_j^{tenant}(t_1, t_2)$ for tenant j from t_1 to t_2 is defined as follows:

$$\rho_j^{tenant}(t_1, t_2) = \frac{\overline{Alloc}_j^{tenant}(t_1, t_2)}{\overline{FairShare}_j^{tenant}(t_1, t_2)}. \quad (12)$$

$\rho_j^{tenant}(t_1, t_2) \geq 1$ implies good long-term fairness in terms of GPU service time for tenant T_i. It means the tenant can use more or at least the same amount of GPU service time than its fair share. In contrast, $\rho_i^{job}(t_1, t_2) < 1$ indicates unfairness since the tenant receives less GPU service time than a separate cluster of M/T GPUs.

We can also calculate the job fairness for $J_{i,j}$. Let $W_{i,j}^{job}$ be the share weight of $J_{i,j}$ in its corresponding VC. Its instantaneous fair resource share at time t should be

$$FairShare_{i,j}^{job}(t) = \frac{FairShare_j^{tenant}(t) \times W_{i,j}^{job}}{\sum_{i=1}^{N_j}(Active(J_{i,j}, t) \times W_{i,j}^{job})}. \quad (13)$$

Its fair GPU service time over $[t_1, t_2]$ should be

$$\overline{FairShare}_{i,j}^{job}(t_1, t_2) = \int_{t_1}^{t_2} \min(Demand_{i,j}^{job}, FairShare_{i,j}^{job}(t))\, dt. \quad (14)$$

The job fairness degree $\rho_{i,j}^{job}(t_1, t_2)$ for $J_{i,j}$ from t_1 to t_2 is defined as follows:

$$\rho_{i,j}^{job}(t_1, t_2) = \frac{\overline{Alloc}_{i,j}^{job}(t_1, t_2)}{\overline{FairShare}_{i,j}^{job}(t_1, t_2)}. \quad (15)$$

4.2.3 Application

Our proposed LTGF can provide guidance for realizing fairness scheduling at both the tenant and job levels. To achieve tenant-level fairness, we can use the max-min approach to maximize the minimal ρ_j^{tenant} in Equation 12:

$$\texttt{maximize:} \min_{j \in T} \rho_j^{tenant}(t_1, t_2).$$

Job level fairness can also be guaranteed in a similar way with the following objective:

$$\texttt{maximize:} \min_{i \in N_j} \rho_{i,j}^{job}(t_1, t_2).$$

Since these two parameters are used to regulate two different levels of fairness separately, we do not unify them. On one hand, our proposed two-phase scheduling algorithm (Section 5.2) enables to achieve both job-level and tenant-level fairness simultaneously. On the other hand, these two parameters give the cluster administrators more flexibility to focus on different types of fairness based on their demands and situations.

Compared to the long-term finish-time fairness in Themis, LTGF can provide more comprehensive assessment for both temporal and spatial impacts at the tenant and job levels. Besides, LTGF computes the fairness of a job or tenant over a past period. In contrast, Themis needs to predict the remaining time of each job to calculate the finish-time fairness, which is not always accurate (Section 3.1), and can affect the subsequent scheduling decisions.

5 A Fair Scheduling System

In this section, we introduce ASTRAEA, a fair DLT job scheduler for GPU clusters. ASTRAEA achieves our proposed Long-Term GPU-time Fairness with two techniques. The first one is a lease-based training scheme (Section 5.1). It breaks a long-term job into a sequence of multiple sub-jobs, which enables balancing jobs with different running time, and rearranging job execution orders. The second innovation is a two-phase scheduler based on LTGF across tenants and jobs (Section 5.2). It maintains the two-level fairness using a max-min approach.

Figure 7 shows the overview and workflow of ASTRAEA. Different tenants submit jobs to their pending queues individually (❶), which are then processed by the two-phase scheduler. The scheduler first selects tenants based on the tenant-fairness metric (❷), and then selects one job from the tenant's pending queue based on the job-fairness metric (❸). The selected job will be allocated to the resource pool according to the placement strategy (❹). A job will return to its pending queue for renewal when its lease is expired (❺). Since job placement strategy is not included in our contributions, we use a common consolidated job placement strategy along with ASTRAEA, which deploys a job on as few nodes as possible, following [6], [7], [19]. Other state-of-the-art GPU placement algorithms determined by cluster administrators can be integrated into ASTRAEA as well.

To summarize, we leverage the lease-based training scheme for a more flexible arrangement of DLT jobs. We utilize the two-phase fairness scheduling algorithm to make scheduling decisions and achieve LTGF in a multi-tenant cluster. Below we detail each technique.

5.1 Lease-based Training Scheme

To balance the jobs with different GPU demands and finish-time, we design a novel lease-based DL training scheme. As discussed in Section 2.2, a DLT job is usually a long iterative process, causing inflexibility to the scheduler. Thus, we propose to break a long training job into a series of periods with a fixed length, namely *lease terms*. As shown in Figure 8, at every scheduling round, each pending job needs to request for a lease term. If this request is approved, the job will also receive the demanded resources, and perform the computation. At the end of this lease, the running job needs to proactively make a checkpoint, and then submit the renewal request for the next lease term. If the renewal is successful, the job can continue the execution. Otherwise, it will be preempted and its resources are released to other jobs and tenants. Jobs that can be finished within one term can release the resources immediately after the completion without the need for lease renewal.

The lease-based training scheme is essential to the scheduling process and is integrated with the two-phase scheduling algorithm tightly. Based on the scheduling decision made by the algorithm, the scheme is used to enforce resource re-allocation between jobs. For the scheduler, at every cycle, it needs to decide whether the lease renewal request should be approved for each running job. If yes, the scheduler will send a notification of "successful renewal " to the job so it can continue the execution. Otherwise, the scheduler will suspend the job and reclaim the resources.

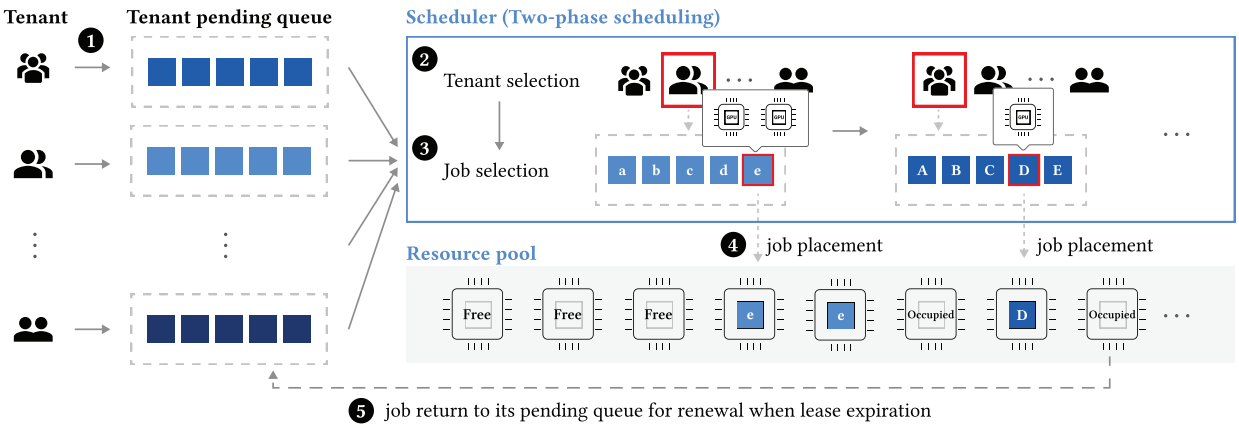


Fig. 7: Overview of ASTRAEA.

Fig. 8: Job execution under the lease-based training scheme.

In addition, the scheduler also needs to select pending jobs, grant lease terms and resources to them, and resume their computation. All these decisions are made by our two-phase scheduling algorithm with the LTGF metric.

The concept of lease was adopted in various domains, e.g., networking [32], cloud computing [33], computer architecture [34], etc. Previous studies also leveraged similar ideas for resource management and DLT job scheduling [6], [7], [8]. Below we present some design details.

5.1.1 On-demand Checkpoint

The lease-based training scheme brings imperative resource re-allocation and reclamation, introducing the need of on-demand checkpoint for DLT jobs. A job needs to make a checkpoint before the end of the lease term to avoid the loss of progress. We make some changes in the user-side DL library to achieve this function. A daemon is created when a job starts to execute. When the job losses its lease term and resources, the scheduler sends a POSIX signal to notify the daemon of saving the current checkpoint. However, a checkpoint has to be saved at the end of one iteration. There may be not enough time for the job to finish the current iteration before the resources are reclaimed. To solve this issue, ASTRAEA implements a *last store* technique. At every iteration, the job updates the model and other information as a snapshot. Once receiving the preemption signal, the daemon flushes the latest snapshot to the disk and waits for the current iteration to complete. An updated snapshot is made and stored if this iteration is finished before the job preemption. Otherwise, the previous flushed snapshot will be used for the future lease term.

5.1.2 Lease Renewal

There is minor modification over the conventional scheduling algorithm to accommodate the lease-based training scheme, along with other renewal mechanisms. At each scheduling cycle, the scheduler needs to handle the requests from both the running and pending jobs. ASTRAEA processes these requests with the following steps. (1) The scheduler treats all the jobs as in the pending state, and all the resources as available. It selects the jobs to be scheduled based on the fairness algorithm in Section 5.2. (2) For each running job in the current lease term, if it is selected by the scheduler, then its lease renewal is successful, and the execution will continue in the next lease with the current assigned resources. Otherwise, the job will be preempted and the resources are reclaimed. (3) After all the resources of the preempted jobs are freed, the scheduler performs the placement policy (consolidated placement strategy in our implementation) to allocate resources to the selected pending jobs, and then resume their execution.

We introduce a coordinator to reconcile the job renewal. This coordinator is responsible for not only triggering the lease renewal and reclaiming the resources of preempted jobs, but also resuming jobs with new assigned lease terms. The resumed jobs will continue running from the last checkpoint with the help of a network-attached storage system.

5.1.3 Lease Term Length

The length of a lease term is critical to determine the scheduling efficiency and training performance. A short lease term can increase the frequency of making checkpoints and training overhead, and cause heavy loads for the scheduler. A long lease term fails to appropriately adjust the job execution order, leading to poor fairness and JCT. Therefore, how to set a proper lease term is a challenging problem, as it depends on many configurations and behaviors of the cluster and DLT jobs.

We can heuristically identify the proper length for the target cluster based on its historical record. Specifically, we collect a trace of past jobs, simulate the scheduling behaviors, and measure the corresponding job fairness and performance under different lease term lengths. Based on the simulation results, we pick the optimal value that best balances the fairness and performance for the cluster. Based on the running time distribution of DLT jobs in Section 3, a majority of jobs have short duration (nearly 70% jobs can be finished within 10 minutes). Therefore, an appropriate length ensures that most jobs can be completed within one lease term. This significantly reduces the overhead. We can periodically collect the traces and simulate the scheduler to

adjust the configuration of term length adaptively. Detailed results can be found in Section 6.2.

5.2 Two-Phase Fairness Scheduling Algorithm

To achieve the two-level LTGF defined in Section 4.2, we design a novel two-phase scheduling algorithm, as shown in Algorithm 1. The core of this algorithm is the AllocateJob function, which is called in every scheduling round.

In this function, the scheduler first updates the fairness metric for each tenant based on Equation 12 (line 2), and for each job based on Equation 15 (line 3). These metrics are computed from the beginning t_{begin} to the current moment t_{now}. Then at Phase 1, the scheduler first maintains tenant-fairness by selecting the tenant p^* with the smallest fairness index (line 5). If this tenant does not have any pending jobs, the scheduler will skip it. Otherwise, it will continue with Phase 2, which selects the job with the highest priority from the tenant's pending queue (line 7). If there are enough resources and the job could be satisfied with the current placement strategy, the scheduler will grant the lease term and demanded GPUs to this selected job (line 10-15). Otherwise, it will skip to the next tenant to prevent starvation. Finally, it removes the job from its corresponding pending queue (line 13) and updates the tenant level fairness metrics as if the allocated job was running (line 14-15). The scheduler repeats the above two phases until there are no pending jobs or adequate resources.

Algorithm 1: Two-phase job scheduling.

Input: T tenants and N jobs, a cluster of M free GPUs.

1 **Function** AllocateJob
2 Update ρ^{tenant} for each tenant with Equation 12
3 Update ρ^{job} for each job with Equation 15
4 **while** there exist pending jobs $\in N$ with demand $r \leq M \wedge$ satisfy placement strategy **do**
// Phase 1: Tenant selection
5 $p^* = \arg\min_{j \in T} \rho_j^{tenant}(t_{begin}, t_{now})$
// Phase 2: Job selection
6 $J_{p^*} \leftarrow$pending job list of tenant p^*
7 i^* = SelectJob(J_{p^*})
8 $r_{i^*} \leftarrow$ GPU demand of job i^*
// Apply current placement strategy
9 $\mathcal{A}_{i^*} \leftarrow$ Placement(M, i^*)
10 **if** $r_{i^*} \leq M \wedge \mathcal{A}_{i^*} \neq \varnothing$ **then**
// Allocate r_{i^*} resources to job.
11 Allocate($\mathcal{A}_{i^*}$, i^*)
12 $M = M - r_{i^*}$
13 $J_{p^*} = J_{p^*} \setminus \{i^*\}$
// Update the related metrics of tenant p^*.
14 $\overline{Alloc}_j^{tenant}(t) = \overline{Alloc}_j^{tenant}(t) + r_{i^*} * t_{lease}$
15 Update ρ_j^{tenant} according to Equation 12.
16 **else**
// Continue with the rest tenants.
17 $T = T \setminus \{p^*\}$

5.2.1 Job Selection

In Phase 2, the scheduler selects a job from the tenant's pending queue for scheduling. Algorithm 2 shows the procedure of this function. Basically, it calculates the reward for each job based on the job-level fairness index. Then it selects the job with the maximal reward (e.g., smallest fairness index) as the candidate.

Note that this job selection process can be integrated with other job-level prioritization approaches. For instance, three types of jobs can be assigned with higher priority for selection if not scarifying the fairness. (1) Some jobs have ultra-short duration. If we prioritize the newly submitted jobs, these short jobs can be scheduled and completed promptly, which can reduce the job pending overhead. (2) Some jobs have been executed for a large number of lease terms, and more likely have a relatively long duration. Thus, special attention to the lease renewal of these jobs can reduce the total overhead caused by frequent preemption. (3) Users may specify some important and urgent jobs that need to meet certain deadlines. The scheduler can also increase the reward of these jobs during selection, to ensure they are not affected by lease expiration.

Algorithm 2: Select a candidate job from a job list.

Input: A job_list J
Output: Selected job from J.

1 **Function** SelectJob
2 **forall** each job i in J **do**
// Calculate job reward.
3 $i.reward = -\rho_i^{job}(t_{begin}, t_{now})$
4 $i^* = \arg\max_{i \in J} i.reward$
5 **return** i^*

6 Evaluation

6.1 Experimental Methodology

6.1.1 Implementation Details

We develop a trace-driven simulator to implement and evaluate our fairness metric and AStraea system. We adopt several techniques to optimize the system implementation. First, to select a tenant at Phase 1, we check the gap between the resources a tenant can utilize in a scheduling round and his weighted quota. Specifically, we slightly modify Equation 8 for the tenant-level fairness as below:

$$FairShare_j^{tenant}(t) = Quota_j(t). \tag{16}$$

This can better reveal the discrimination of utilized resources with different resource weights of the tenants. The corresponding fairness metric (Equation 12) illustrates the weighted amount of resources allocated to tenant p during $[t_1, t_2]$. This adjusted metric can motivate a tenant to *lend* his unused resources to others and reclaim more than his share when he has a burst demand in the future, which can achieve the sharing incentive more effectively. Moreover, although all tenants may remain fair on a long-time scale, it is possible that a burst request from a tenant may prevent him from submitting new jobs as long as he has a higher LTGF metric. Therefore, this modification can compensate for the problem that LTGF cannot be quickly adjusted on short-time scales.

Second, to select a job at Phase 2, we consider more job-level prioritization, as discussed in Section 5.2.1. We

prioritize the newly submitted job, which can help short jobs be scheduled and completed promptly. This achieves overall better results considering the fact that short jobs take the majority in GPU clusters.

6.1.2 Workloads

For the fidelity and generality of our simulation, we select two real-world DLT job traces and use a two-week snapshot of both two traces. The first one is Venus, collected from a production cluster of 1064 GPUs in SenseTime [20]. This trace contains 11304 jobs from 16 different tenants. The second one is Philly from the Microsoft cluster [19], containing 44329 jobs from 15 VCs with 2490 GPUs. Note that the Philly trace does not provide the size of each VC cluster, so we make an assumption that the size of a VC is proportional to the total number of GPUs requested by all the jobs in this VC.

6.1.3 Simulation configurations

We set the lease term length as 900s, which can best balance the cluster efficiency and fairness. Identification of this value is detailed in Section 6.2. The scheduling interval is set as 10s to provide balanced response time for newly submitted jobs. These configurations are compatible with the default settings of real-world round-based schedulers [35]. Considering there exist CPU jobs in Venus and general under-utilization for production clusters, we follow [36] to scale down the cluster size to make usage more intensive: we set 100 nodes for Venus and 210 nodes for Philly, with each node comprised of 8 GPUs. We will fully investigate the effectiveness of ASTRAEA under different cluster sizes in Section 6.5. The duration and GPU demand of each job is unmodified in the workload, regardless of whether the submission time has been normalized. The cluster is initialized with all the nodes available and starts to schedule jobs submitted in the last scheduling round. We apply a consolidated placement strategy in the simulation.

6.1.4 Metrics

We use the following metrics to quantify the effectiveness of our proposed ASTRAEA.

- **Fairness.** We measure the LTGF fairness at both the tenant and job levels (Section 4.2). For tenant-fairness, we calculate ρ^{tenant} at multiple fixed intervals for each tenant (Equation 12), as well as the distribution. For job-fairness, we calculate ρ^{job} for every job (Equation 15).
- **Average slowdown.** We use the average slowdown to quantify the normalized pending situation, which is calculated as the sum of pending time and running time divided by the running time (Section 3.2). This can represent users' experience: a small average slowdown indicates higher fairness and shorter waiting time, which is beneficial especially for short-term jobs.
- **Job Completion Time (JCT).** We compute the average JCT for the cluster efficiency. An efficient cluster has better scheduling performance with lower average JCT.

6.1.5 Comparison with Other Baselines

We consider the following six baselines for comparisons. The first four are mainstream DLT job schedulers for different purposes (e.g., improving job performance and resource utilization, handling heterogeneous environments). The last two are state-of-the-art DLT schedulers specifically for job-level fairness and tenant-level fairness, respectively.

- **YARN-CS** [4]: a static quota strategy is adopted to allocate resources according to users' weights.
- **HiveD** [11]: a dynamic quota is implemented in this scheduler, allowing users to submit spot jobs to other VCs with lower utilization.
- **Allox** [37]: this scheduler adopts the min-cost bipartite matching to solve the placement problem with heterogeneous resources, and select tenants with the lowest progress in each scheduling round.
- **Tiresias-L** [7]: it measures the aggregated GPU time each job receives, and uses the Least Attained Service to maintain the resource allocation.
- **Themis** [8]: it proposes the long-term finish-time fairness as a new metric to evaluate fairness. We implement this scheme by calculating the remaining iterations and predicting throughput with non-shared situation, with a reasonable estimation error rate.
- **Gandiva$_{\text{fair}}$** [13]: this scheduler utilizes a gang-aware split stride, ticket-based mechanism to manage the resources in heterogeneous systems, focusing on the tenant-level fairness[2].

6.2 Selection of Lease Term Length

Fairness and cluster efficiency highly depend on the lease term length. We evaluate the impact of lease term lengths on the scheduling system. Figure 9 shows the average JCT (doted line) and preemption overhead (bars) with different lease term lengths. We observe the preemption overhead is relatively small, especially when the lease term becomes longer. When we increase the lease term length, the JCT is first decreased, as more jobs can be completed within one term without being preempted, and the frequency of job switching is reduced with smaller overhead. However, when the lease term becomes longer, the JCT also becomes larger because jobs have fewer chances to be adjusted. We can find a length that achieves strong fairness while not sacrificing the job performance. Figure 9 also shows the relationship between the job-level fairness and lease term length. We observe that a longer term can reduce the fairness, as the frequency of job execution rearrangement is reduced either, making it less efficient to adjust the fairness via scheduling. Based on the above analysis, we select a proper length as 900s from our historical trace considering both fairness and efficiency, because it has similar performance while satisfactory fairness compared with the most efficient length (1200s). In practice, the lease term length could be adjusted by cluster administrators. We will use this configuration for the following evacuations. Scheduler designers can identify the lease term length for their clusters and jobs similarly.

2. $Gandiva_{fair}$ also introduced an automated trading mechanism to handle GPU heterogeneity and time sharing. This is not the focus of this paper and will not be included in the evaluation. This mechanism can be integrated with ASTRAEA as well.

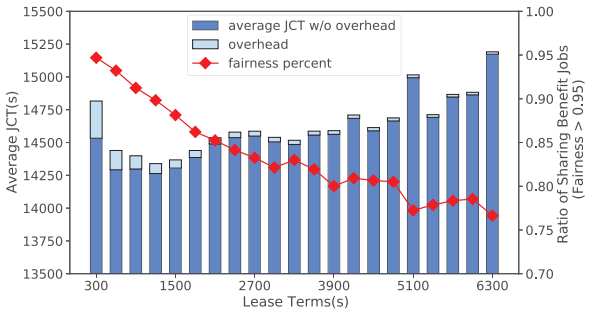
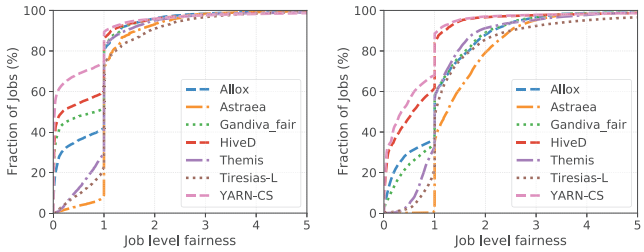


Fig. 9: Fairness affected by different lease terms.

6.3 Fairness Comparisons

Figure 10 shows the cumulative distribution of job-level fairness under different schedulers. We observe more jobs maintain higher fairness with ASTRAEA than the other schedulers. If we treat jobs with fairness metric smaller than 0.95 as *sharing loss* jobs, then ASTRAEA only produces 7.1% *sharing loss* jobs in Venus. In contrast, Themis and YARN-CS schedulers give 19.7% and 73.6% *sharing loss* jobs, 2.8× and 10.3× of ASTRAEA. ASTRAEA exhibits even higher advantage in Philly, which has more uneven distributions of resources and jobs among tenants.



(a) Venus trace (b) Philly trace

Fig. 10: Cumulative distributions of job-level fairness.

Figures 11 shows the tenant-level fairness metrics with different schedulers for the Venus trace. The x-axis in each figure denotes tenants, and y-axis shows the distribution of tenant-level fairness metric for each tenant over many periods. A fair scheduler should maintain a higher metric all the time. We observe that ASTRAEA can guarantee most tenants have the fairness metric larger than or equal to 1. The other schedulers give relatively worse tenant-level fairness, indicating they fail to meet the *resource-as-you-contribute* requirement and sharing incentive. Quantitatively, we calculate the ratio of cases whose metric value is smaller than 1 as the indicator that the tenant is experiencing sharing loss. ASTRAEA gives such an unfairness ratio of 5.2%. For the other schedulers, the best one is Allox (6.9%), similar with $Gandiva_{fair}$ (8.0%), and the worst are Tiresias-L (49.0%) and YARN-CS (44.6%). ASTRAEA outperforms these schedulers by 1.3× to 9.42×. Similar conclusions can be drawn from the results for Philly trace.

6.4 Cluster Efficiency Comparisons

We show how effective our scheduler is in mitigating the queuing congestion and reducing the average JCT in the cluster. Figure 12 shows the cumulative distributions of average slowdown. We observe that ASTRAEA can achieve the least pending overhead compared to other schedulers, indicating that it can provide prompt responses to short-term jobs and reduce the waiting time of long-term jobs. A very small portion of jobs suffer from long pending overhead, because they have very short running time (e.g., 5 seconds) compared to one scheduling round. A shorter scheduling round can decrease these jobs' pending time at the cost of larger scheduling overhead.

Figure 13 shows the average JCT of each scheduler. The fairness enforcement in ASTRAEA can ensure work conservation for short-time jobs and reduce the pending time

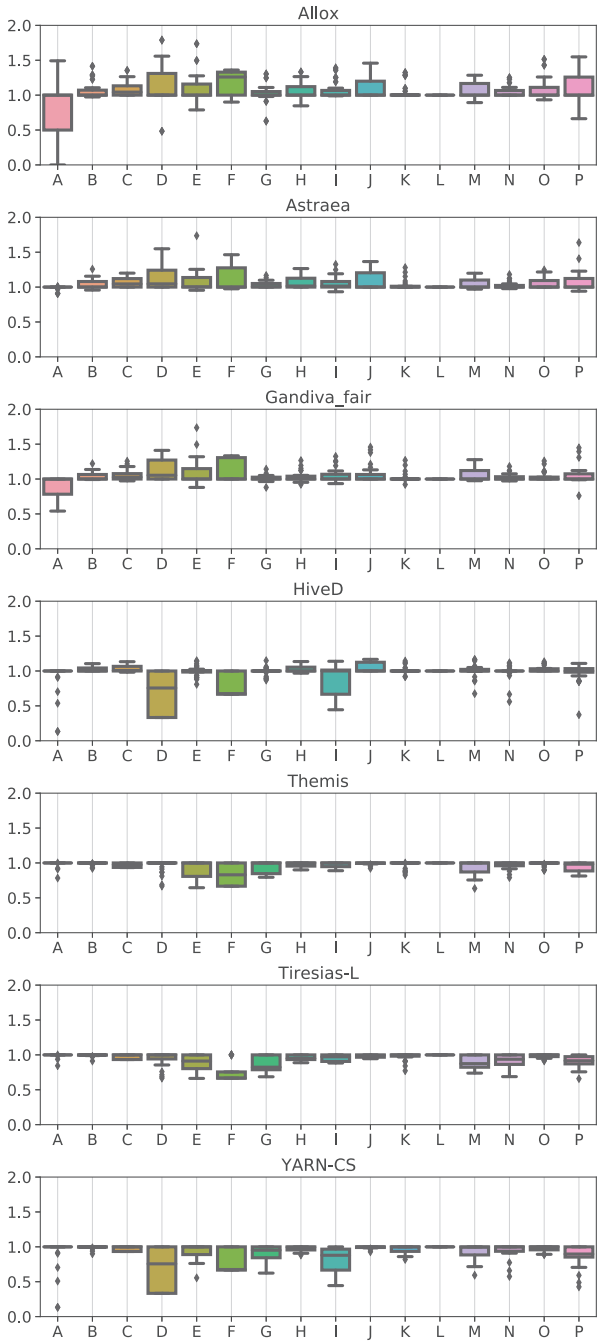
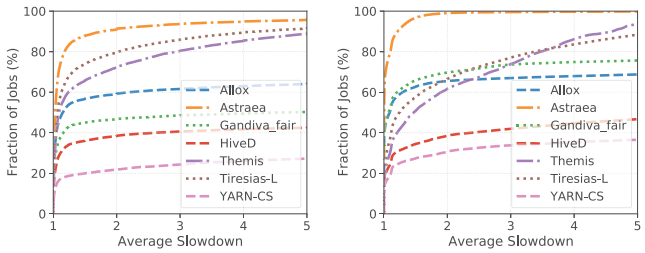


Fig. 11: Distributions of tenant-level fairness in Venus.

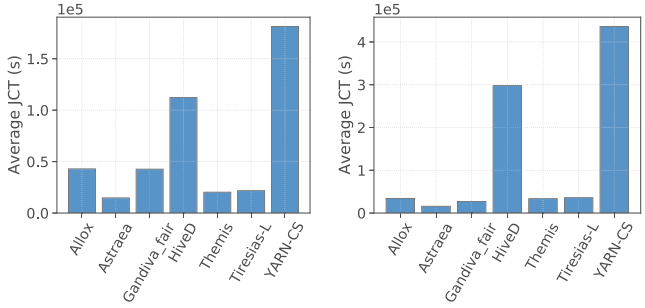
for long-time jobs. It also enables the flexible adjustment of tenant quota. Due to these mechanisms, ASTRAEA has smaller average JCT compared to other schedulers.

Similar as other preemption-based schedulers, ASTRAEA introduces extra overhead when rearranging the job execution order. This overhead can be divided into two components: making checkpoints, and cold-start job. Checkpoints are made for every lease expiration in case of preemption, while cold start occurs only when the job lease renewal fails. Previous work has shown that cold start dominates the job preemption overhead [7]. Our lease-based training scheme enables the job to renew its lease and continue the execution without the cold start, which can improve the JCT. In general, this scheme only brings approximately 0.8% overhead to the average JCT with a lease term length of 900s (Figure 9) in the Venus trace, which is negligible.



(a) Venus trace (b) Philly trace

Fig. 12: Cumulative distribution of average slowdown.

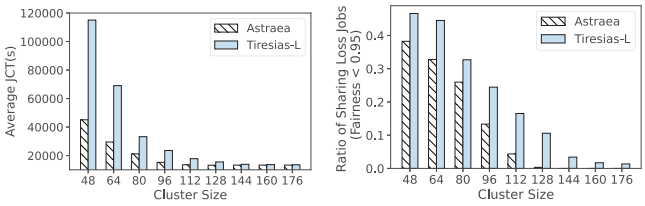


(a) Venus trace (b) Philly trace

Fig. 13: Average JCT for different schedulers.

6.5 Impact of Cluster Sizes

The cluster loads can also affect the effectiveness of ASTRAEA and other scheduling algorithms. We quantitatively show the impact of cluster sizes on the average JCT and ratio of *sharing loss* jobs with a fairness metric < 0.95. We compare ASTRAEA with Tiresias-L, which outperforms other baseline methods according to our evaluations in previous sections. Figures 14(a) and 14(b) give the comparison results with different cluster sizes (i.e., numbers of GPU nodes). We observe that ASTRAEA always beats Tiresias-L for both cluster efficiency and fairness, regardless of the cluster size. In addition, both ASTRAEA and Tiresias-L exhibit more advantages when the cluster becomes larger. This is because a larger cluster with more available resources can satisfy more jobs with reduced pending time, and these idle resources can be better utilized to achieve fairness for various jobs.



(a) Average JCT (b) Job-level fairness

Fig. 14: Impact of cluster sizes on ASTRAEA and Tiresias-L.

7 Related Works

7.1 Fairness Metrics

We review and analyze the fairness metrics for conventional CPU workload or DLT jobs in prior works. Some metrics consider the relative job performance (e.g. job execution time) between the shared and independent systems [8], [14]. Some methods calculate the benefit gap between different tenants: a smaller gap indicates a fairer scheduler [38], [39]. Some methods utilize queuing theory to consider the job experience in the pending queue [40], [41]. A variety of works also proposed fairness solutions for OS processes [13], [42], [43], [44] or distributed workload scheduling [45], [46], [47], [48]. They cannot be applied to DLT job scheduling due to the unique job characteristics.

7.2 DLT Job Management

Over the past years, researchers have designed a quantity of scheduling algorithms and systems to optimize the execution of DLT jobs in GPU clusters from different perspectives. (1) Some works designed schedulers to maximize the cluster utilization [6], [12], [19], [49], [50]. Particularly, GPU sharing across jobs is a facilitating mechanism [51], [52] to improve GPU utilization, which is adopted in these works. (2) Some works aim to minimize the average JCT of DLT jobs [7], [31], [37], [53], [54]. (3) New schedulers were introduced to optimize the job performance [6], [7], [8], [11], [13], [19], [31], [37], [55]. Different from those works, our goal is to enhance both the job-level and tenant-level fairness while maintaining the job performance.

8 Conclusion

In this paper, we present a novel study about resource allocation fairness for DLT jobs in GPU clusters. We perform a quantitative analysis of a real-world job trace from a production cluster in SenseTime, to uncover the severe fairness problem in DLT scheduling. To mitigate this issue, we propose a new yet practical metric, `LTGF`, to accurately measure the fairness at both the job and tenant levels , without prediction of job remaining time and future throughput. We further design ASTRAEA, a new and efficient scheduler with a two-phased scheduling algorithm based on `LTGF` to

enforce fairness. We evaluate ASTRAEA via large-scale simulations on real-world cluster traces. Experimental results show that ASTRAEA can guarantee stronger fairness without sacrificing the job performance or cluster utilization, compared to other state-of-the-art schedulers.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments. The research is supported in part by the National Science Foundation of China (Grants No. 62032001, No. 61672053, No. U1611461, and No. 62032008). It is also supported under the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contributions from the industry partner(s).

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Imagenet training in minutes," in *Proceedings of the 47th International Conference on Parallel Processing*, 2018, pp. 1–10.

[3] P. Sun, Y. Wen, R. Han, W. Feng, and S. Yan, "Gradientflow: Optimizing network performance for large-scale distributed dnn training," *IEEE Transactions on Big Data*, 2019.

[4] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013, pp. 1–16.

[5] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, 2011.

[6] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *OSDI*, 2018.

[7] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A gpu cluster manager for distributed deep learning," in *NSDI*, 2019.

[8] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and efficient gpu cluster scheduling," in *NSDI*, 2020.

[9] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 2003.

[10] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, 2016.

[11] H. Zhao, Z. Han, Z. Yang, Q. Zhang, F. Yang, L. Zhou, M. Yang, F. C. Lau, Y. Wang, Y. Xiong, and B. Wang, "Hived: Sharing a GPU cluster for deep learning with guarantees," in *OSDI*, 2020.

[12] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing, "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *OSDI*, 2021.

[13] S. Chaudhary, R. Ramjee, M. Sivathanu, N. Kwatra, and S. Viswanatha, "Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning," in *EuroSys*, 2020.

[14] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *SOSP*, 2009.

[15] M. Zaharia, "Job scheduling with the fair and capacity schedulers," *Hadoop Summit*, 2009.

[16] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *EuroSys*, 2013.

[17] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, 2019.

[19] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant gpu clusters for dnn training workloads," in *USENIX ATC*, 2019.

[20] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021.

[21] Q. Zhang, Z. Han, F. Yang, Y. Zhang, Z. Liu, M. Yang, and L. Zhou, "Retiarii: A deep learning exploratory-training framework," in *OSDI*, 2020.

[22] U. Misra, R. Liaw, L. Dunlap, R. Bhardwaj, K. Kandasamy, J. E. Gonzalez, I. Stoica, and A. Tumanov, "Rubberband: Cloud-based hyperparameter tuning," in *EuroSys*, 2021.

[23] L. Xie, J. Zhai, B. Wu, Y. Wang, X. Zhang, P. Sun, and S. Yan, "Elan: Towards generic and efficient elastic training for deep learning," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 78–88.

[24] D. G. Feitelson, "A survey of scheduling in multiprogrammed parallel systems," Oct 1994.

[25] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013, pp. 351–364.

[26] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys*, 2010.

[27] W. Gao, Z. Ye, P. Sun, Y. Wen, and T. Zhang, "Chronus: A novel deadline-aware scheduler for deep learning training jobs," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 609–623.

[28] D. Narayanan, F. Kazhamiaka, F. Abuzaid, P. Kraft, A. Agrawal, S. Kandula, S. Boyd, and M. Zaharia, "Solving large-scale granular resource allocation problems efficiently with pop," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM*, 2021, pp. 521–537.

[29] J. W. Park, A. Tumanov, A. Jiang, M. A. Kozuch, and G. R. Ganger, "3sigma: distribution-based cluster scheduling for runtime uncertainty," in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–17.

[30] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *Nsdi*, vol. 11, no. 2011, 2011, pp. 24–24.

[31] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *EuroSys*, 2018, pp. 1–14.

[32] D. Banerjee, L. Fernandes, V. Vallabhaneni, and V. Jain, "Method, system and article for advance lease negotiation in dhcp," Jul. 13 2006, uS Patent App. 11/034,274.

[33] A. Adya, J. Dunagan, and A. Wolman, "Centrifuge: Integrated lease management and partitioning for cloud services," in *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10)*. San Jose, CA: USENIX Association, Apr. 2010.

[34] P. Li, C. Pronovost, W. Wilson, B. Tait, J. Zhou, C. Ding, and J. Criswell, "Beating opt with statistical clairvoyance and variable size caching," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 243–256.

[35] Kubernetes contributors, 2021. [Online]. Available: https://kubernetes.io/

[36] A. Tumanov, T. Zhu, J. W. Park, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "Tetrisched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters," in *EuroSys*, 2016.

[37] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "Allox: Compute allocation in hybrid clusters," in *EuroSys*, 2020.

[38] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, "Altruistic scheduling in multi-resource clusters," in *OSDI*, 2016.

[39] D. Raz, H. Levy, and B. Avi-Itzhak, "A resource-allocation queueing fairness measure," *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, pp. 130–141, 2004.

[40] G. Sabin, G. Kochhar, and P. Sadayappan, "Job fairness in non-preemptive job scheduling," in *ICPP*, 2004.

[41] G. Sabin and P. Sadayappan, "Unfairness metrics for space-sharing parallel job schedulers," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 2005.

[42] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Proceedings of the 1st*

USENIX conference on Operating Systems Design and Implementation, 1994, pp. 1–es.

[43] D. H. Epema, "An analysis of decay-usage scheduling in multiprocessors," *ACM SIGMETRICS Performance Evaluation Review*, vol. 23, no. 1, pp. 74–85, 1995.

[44] D. H. J. Epema, "Decay-usage scheduling in multiprocessors," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 4, pp. 367–415, 1998.

[45] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[46] A. Mu'alem and D. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.

[47] E. Shmueli and D. G. Feitelson, "Backfilling with lookahead to optimize the packing of parallel jobs," *Journal of parallel and distributed computing*, vol. 65, no. 9, pp. 1090–1107, 2005.

[48] S. Tang, B.-s. Lee, B. He, and H. Liu, "Long-term resource fairness: Towards economic fairness on pay-as-you-use computing systems," in *Supercomputing*, 2014.

[49] P. Yu and M. Chowdhury, "Salus: Find-grained GPU sharing primitives for deep learning applications," in *MLSys*, 2020.

[50] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "Antman: Dynamic scaling on GPU clusters for deep learning," in *OSDI*, 2020.

[51] L. Shi, H. Chen, J. Sun, and K. Li, "vcuda: Gpu-accelerated high-performance computing in virtual machines," *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 804–816, 2012.

[52] A. J. Pena, C. Reaño, F. Silla, R. Mayo, E. S. Quintana-Ortí, and J. Duato, "A complete and efficient cuda-sharing solution for hpc clusters," *Parallel Computing*, vol. 40, no. 10, pp. 574–588, 2014.

[53] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *NSDI*, 2016.

[54] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "Slaq: quality-driven scheduling for distributed machine learning," in *SoCC*, 2017.

[55] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *OSDI*, 2020.