Ares: Fair and Efficient Scheduling of Deep Learning Jobs with Elastic Fair Queuing

YIFEI LIU, Shanghai Jiao Tong University, Shanghai, China
CHEN CHEN, Shanghai Jiao Tong University, Shanghai, China
QIANG WANG, Computer Science and Technology, Harbin Institute of Technology Shenzhen, Shenzhen, China
YU FENG, Shanghai Jiao Tong University, Shanghai, China
WEIHAO CUI, Shanghai Jiao Tong University, Shanghai, China
QUAN CHEN, Department of Computer Science, Shanghai Jiao Tong University, Shanghai, China
MINYI GUO, Computer Science, Shanghai Jiao Tong University, Shanghai, China

Schedulers play a vital role for GPU cluster serving model training jobs, and an ideal scheduler shall behave well in both fairness and efficiency. However, existing clusters mostly focus on only one aspect and fall short in the other. To solve that problem, given that the resource demand of a model training job can often be approximated a priori, our insight is to preferentially service jobs that complete earlier under instantaneous fair sharing, which can emulate shortest job first while avoiding starvation. Following that insight, in this paper we propose Ares, an efficient and also fair scheduler for deep learning jobs. Ares leverages the conception of virtual finish time in network fair queuing methods, which supports efficient estimation of job completion order at job arrival time. For the jobs with earlier virtual finish times, we allow it to use more resources than it originally demands to attain fast completion—so that those resources can also be released sooner and no job is actually hurt. We keep the global batch size unchanged to ensure accuracy validity, and also ensure that the degradation of resource utilization caused by scaling-out is bounded. We call such scheduling method elastic fair queuing, which can provide theoretical fairness guarantee. We evaluate Ares performance with both testbed experiments and large-scale simulations. The results show that Ares can reduce the average job completion time by over 20% and also reduce the number of unfairly-served jobs by over 40%.

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

Additional Key Words and Phrases: Job scheduling, DNN training, GPU cluster, Scaling

1 Introduction

Deep learning training (DLT) jobs are prevalent in modern GPU clusters [25, 31, 36]. In such shared clusters with intense resource contention, the cluster scheduler has played a crucial role in the overall job performance. In particular, the scheduling requirement of an idealized scheduler is *two-fold*: high efficiency and guaranteed fairness [36]. In the efficiency aspect, the average *job completion time* (JCT) shall be minimized; in the fairness

Authors' Contact Information: Yifei Liu, Shanghai Jiao Tong University, Shanghai, Shanghai, China; e-mail: nephrencake@sjtu.edu.cn; Chen Chen, Shanghai Jiao Tong University, Shanghai, China; e-mail: chen-chen@sjtu.edu.cn; Qiang Wang, Computer Science and Technology, Harbin Institute of Technology Shenzhen, Shenzhen, China; e-mail: qiang.wang@hit.edu.cn; Yu Feng, Shanghai Jiao Tong University, Shanghai, Shanghai, China; e-mail: y-feng@sjtu.edu.cn; Weihao Cui, Shanghai Jiao Tong University, Shanghai, China; e-mail: weihao@sjtu.edu.cn; Quan Chen, Department of Computer Science, Shanghai Jiao Tong University, Shanghai, Shanghai, China; e-mail: chen-quan@cs.sjtu.edu.cn; Minyi Guo, Computer Science, Shanghai Jiao Tong University, Shanghai, Shanghai, China; e-mail: guo-my@cs.sjtu.edu.cn.

ACM 1544-3973/2025/9-ART
https://doi.org/10.1145/3766896

aspect, it shall be guaranteed that each job can have predictable worst-case performance irrespective of other jobs' behavior.

Existing cluster schedulers [15, 16, 42, 43] primarily focus on the *efficiency* aspect. Some works like Optimus [42], Pollux [43] and ElasticFlow [15] tentatively allocate resources to jobs with higher resource utilization; some other works like Tiresias [16] and Lucid [21] preferentially serve jobs with the shorter computation time. While those works can reduce the average job completion time, they fall short of providing fairness guarantee. That is, large jobs with low marginal resource utilization may have to wait forever, i.e., suffering the *starvation* problem.

Meanwhile, some other schedulers [32, 36, 39] set *fairness* as the primary objective. In that regard, Themis [36] keeps offering resources to jobs whose allocations are farthest from the theoretical fair share. Gavel [39] is a heterogeneity-aware scheduler that uses a round-based scheduling mechanism to ensure that jobs can attain an equal throughput despite allocated heterogeneous hardware. However, those schedulers are essentially approximating instantaneous fairness, and each job has to bear long completion time due to its fair share limitation.

In a word, existing schedulers fail to simultaneously achieve fair and efficient job scheduling. In light of their deficiencies, we seek to design a scheduling method that can yield fast job completion without compromising fairness. We note that users primarily care about the job completion time instead of the instantaneous resource allocation; therefore, we adopt the long-term fairness notion—seeking to guarantee that each job finishes no later than it would have under instantaneous fair sharing. Compared to short-term fairness (i.e., rigidly enforcing instantaneous fair share), the objective of long-term fairness is more flexible and makes it possible to simultaneously reduce the average job completion time if prioritizing those jobs in a proper order.

In designing an efficient and also (long-termly) fair scheduler, our insight is to preferentially serve the job that has the earliest completion time if under idealized fair sharing policy (under which the cluster resources are equally allocated to each pending job). Such an approach can yield two benefits: first, since smaller jobs usually complete earlier under fair sharing, we can emulate the shortest job first policy; second, long jobs arriving early would—under idealized fair sharing—eventually complete earlier than short jobs arriving late, with the potential to attain guaranteed performance without starvation. Recent works have shown that the total resource consumption of a DLT job can be approximated a priori—by learning from the historical runs [51, 54], predicting with a model [21, 42], or having users directly specify the iteration/epoch number [15] (e.g., for fine-tuning jobs). Such demand information can be readily leveraged for determining the job prioritizing order.

Yet, when putting the above insight into practice, we face several immediate challenges. First, proportionally increasing the per-iteration data processing amount with the allocated GPU number may potentially affect the training accuracy [21]. Second, given the dynamic resource availability with varying job number at runtime, it is hard, if not impossible, to keep maintaining the exact fair share as well as the JCT of each job under idealized fair sharing. Third, due to the network bottleneck, DLT jobs often exhibit *non-linear* scaling curves [43], meaning that excessive resource provisioning for the prioritized jobs may compromise the resource utilization and further blow up the average JCT. Our solution must elegantly tackle these challenges.

In this paper, we propose Ares, a *fair* and *efficient* scheduler for DLT jobs in shared GPU clusters. In principle, it keeps prioritizing the job with the shortest completion time under idealized fair sharing. In particular, to accelerate the training of prioritized jobs without accuracy degradation, we adopt *endo-elastic* training paradigm; that is, we allow a job to use more GPUs than it originally requests while keeping the global batch size unchanged. With such endo-elasticity, each DLT job can be monopolistically served like a *network flow*, with the entire cluster working as the network router. We thus propose *Elastic Fair Queuing* (EFQ), resembling the art of Fair Queuing (FQ) [8, 40] in network scheduling research, which is a cornerstone mechanism that achieves fair link sharing atop holistic packet-by-packet transmission. With EFQ, we can acquire the relative job scheduling order in one shot without frequently refreshing the predicted job completion time. Moreover, given the non-linear scaling pattern, we over-allocate resources to a job only when its resultant resource utilization does not degrade over a threshold. We mathematically prove that, for each DLT job, its completion time under Ares is delayed by no more than a constant compared to it would have under instantaneous fair sharing.

We have implemented the system prototype of Ares, and further evaluate its efficiency and fairness performance with both testbed experiments and simulations (replaying three trace sets from production clusters [20, 24, 25]). Our results show that, when compared with a set of state-of-the-art schedulers including Pollux [43] and Themis [36], Ares can reduce the average JCT by over 20%. Meanwhile, regarding fairness, Ares can decrease the unfair fraction (i.e., the proportion of jobs which complete later than under idealized fair sharing) by up to 41.3%, and the average slow down level is also reduced by up to 44.2%.

In summary, this paper makes the following contributions:

- We reveal the incapability of existing works in attaining fair and efficient scheduling, and find out that the constraint of instantaneous fair sharing can be relaxed to reach higher efficiency without affecting user-perceived long-term fairness.
- We propose Ares, a fair and also efficient scheduler for DLT jobs adopting elastic fair queuing. Ares does not incur accuracy loss or severe utilization degradation, and it can be theoretically guaranteed that the maximum completion delay a job could encounter compared to the fair-sharing case is bounded by a constant.
- We have conducted both testbed measurements and simulations to verify the effectiveness of Ares. Compared with state-of-the-art schedulers, Ares can remarkably reduce the average job completion time and also reduce the number of unfairly-served jobs.

2 Background and Motivation

2.1 Distributed Training in Shared Clusters

Deep Learning Training (DLT) jobs typically process large-volume datasets with huge-scale model parameters. Such processes are computationally intensive, often necessitating distributed training over multiple workers.

Due to the prohibitively high cost to maintain dedicated GPUs, distributed model training is usually conducted in shared GPU clusters [24, 39]. Recent studies [20, 51] show that the workloads in shared production clusters are highly heterogeneous: large jobs may constitute only 10% of the total workload but consume over half of the computational resources [53]. Therefore, with ever-growing resource contention, it is in urgent need to properly design the scheduler for good cluster performance.

In particular, when submitting a DLT job, users expect to have short response time with guaranteed service quality. That is, a good cluster scheduler shall behave well in both efficiency and fairness [36]: on the one hand, the average job completion time (JCT) shall be as short as possible; on the other hand, the scheduler shall avoid negative interferences among jobs, i.e., each job can have service isolation with guaranteed worst-case performance. In the meantime, note that the accuracy performance shall not be affected by the schedulers, and a high resource utilization is also desirable for the cluster operators.

2.2 Prior Scheduling Research for DLT Jobs

In the literature, a series of scheduling methods have been proposed for GPU clusters [32, 36, 42]. Based on the primary optimization objective, these schedulers can be broadly categorized into two categories: *efficiency-centric* schedulers and *fairness-centric* schedulers.

Efficiency-centric Schedulers. Many production schedulers are efficiency-centric. For example, Tiresias [16] and E-LAS [49] apply the Least-Attained-Service heuristic to emulate shortest-remaining-time-first (SRTF) when scheduling duration-agnostic jobs. Meanwhile, Optimus [42] and Pollux [43] propose to dynamically approach the efficiency-optimal allocation by granting the available resources to jobs with larger marginal benefit. Some other methods like Lucid [21], SCHED^2 [35], MLFS [50] and Helios [20] adopt machine learning techniques to optimize efficiency. However, these schedulers fall short in providing service guarantee. Under those schedulers, long jobs may get starved if short jobs keep arriving.

Fairness-centric Schedulers. In the fairness aspect, being agnostic to jobs' total resource demands, schedulers like Themis [36] gradually approach the target fair allocation by preferentially serving the job with the least current share. AlloX [32] and Gavel [39] extend the fair allocation problem to *heterogeneous* resources involving various GPU types or even CPU-GPU hybrid systems. However, all such fair schedulers seek to enforce instantaneous fair allocation among different jobs, limiting each job to use only its (small) fair share and thus incurring long job completion time.

To summarize, fairness and efficiency are two conflicting scheduling objectives for DLT jobs, and existing schedulers fail[1] to behave well in both worlds. Our objective in this paper is thus to design a new scheduling mechanism that can simultaneously yield short average JCT while providing worst-case service guarantee. We next formalize this objective and elaborate our design insights.

2.3 Insight and Challenges

Scheduling objectives with long-term fairness. Efficiency and fairness are two key essentials an ideal scheduler should possess. In efficiency optimization, we focus on *user-perceived* efficiency criterion, seeking to minimize the average JCT. Meanwhile, in fairness optimization, we focus on long-term fairness (i.e., providing guarantee on the worst-case JCT). Existing fairness-centric schedulers [32, 39, 56] typically focus on *short-term fairness*—equally dividing the cluster resources to all active jobs at each instant; yet such a short-term fairness definition is over-rigid for efficiency optimization and indeed also unnecessary: when submitting a DLT job, users care primarily about the JCT instead of the instantaneous allocation. In that regard, we can set the JCT under instantaneous fair sharing as a service target: as long as a job can be guaranteed to finish no later (or within a small delay bound) than that service target, it can be deemed to be fairly served. Such a long-term fairness definition has also been adopted in earlier works like [13, 36], and we will show that it can allow for more flexible efficiency optimizations.

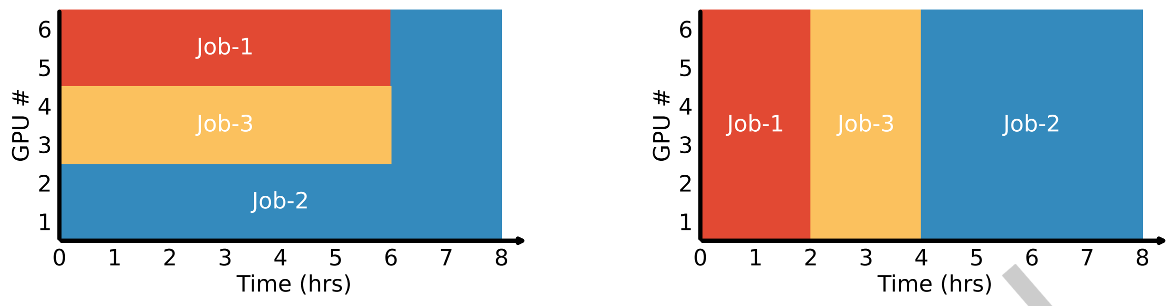
To put it formally, given N jobs, let a_j and f_j respectively be the arrival and completion time of job-j ($j = 1, 2, ...N$), and let $\bar{f}_j$ be the JCT of job-j under instantaneous fair sharing, then our scheduling objective is to minimize the average JCT while ensuring predictable performance, i.e.,

$$\begin{aligned}&\text{minimize } \frac{1}{N}\sum_{j=1}^{N}(f_j - a_j),\\&\text{subject to } f_j - \bar{f}_j \le C, \text{ for all } j \in \{1, 2, ..., N\},\end{aligned} \tag{1}$$

where C is a small constant.

Insight. With the generalized fairness notion above, we find that it is now possible to reduce the average JCT without sacrificing fairness. Fig. 1 presents an illustrative example for that, where the three jobs are all submitted at time 0 (hour). As shown in Fig. 1a, under instantaneous fair sharing, each job is forced to use only its fair share,

[1]Some works [16, 36] claim to be both fair and efficient by patching their main algorithms with a naive remediating measure. For example, to avoid starvation, Tiresias [16]—an efficiency-centric scheduler—manually promotes the job that has been waiting over a given time threshold to the highest priority; meanwhile, for better efficiency, Themis [36]—a fairness-centric scheduler—adopts a knob to select a portion of jobs that are most efficiency-sensitive and allocate available resources only among them. However, it is hard to set a generally-appropriate threshold given the vast job diversity [56]. For example, in Tiresias, an over-small waiting-time threshold may cause large jobs be executed too early (leading to head-of-line blocking), and an over-large threshold would hurt the service guarantee level. We argue that, fair and also efficient scheduling shall be attained elegantly as an *inner property* instead of as an additional solution *patch*. In this paper we seek to make that objective with theoretical guarantee.



(a) Fair sharing. (b) Prioritizing jobs one by one.

Fig. 1. Illustration of the efficiency benefits in prioritizing jobs based on their completion order under instantaneous fair sharing (assuming that jobs exhibit linear scalability).

and the average JCT is 6.6 hrs; however, if we instead prioritize those jobs one by one (based on the relative job completion order shown in Fig. 1b), the average JCT can be reduced to 4.6 hrs—with no job delayed compared with the case in Fig. 1a.

The above scheduling method can be viewed as emulating the *earliest-deadline first* (EDF) algorithm [27]—where, however, the deadline is *not predefined* but set as the *JCT under instantaneous fair sharing*; meanwhile, the job with the earliest "deadline" would be serviced with *as many resources as possible*—instead of with only the fair share. We note that such a method possesses two advantages. First, for jobs arriving at a similar time, the shortest job would have the highest priority in scheduling—meaning that we can emulate shortest-job-first and attain high efficiency. Second, by determining the job service order based on the expected fair completion time, we can avoid starvation. With abundant resources, high-priority jobs can attain fast completion and soon yield the GPU resources to others. For a large job arriving early, as time goes by, its JCT under fair sharing would eventually be earlier than that of new-coming short jobs, thereby having a higher scheduling priority and naturally avoid starvation.

Note that the scheduling optimization in Fig. 1 requires estimating the job resource demand at job arrival time, which is indeed feasible and similar assumptions are widely adopted in many related works [16, 20, 21]. First, users often specify the maximum sample processing amount (or equivalently maximal iteration/epoch number) as the training termination condition (e.g., for fine-tuning jobs where the models are routinely trained for a fixed number of epochs over the newly-collected data [15]), and we can efficiently transfer that data processing amount to the GPU execution time via a short profiling process. Second, as reported in existing works [51, 54], a majority (over 90%) of workloads in DLT clusters are recurring, and their execution time can be faithfully estimated with the prior execution data. Third, the resource demand of a job can also be estimated by analytical [42] or machine learning models [21]. Such information can be leveraged when making scheduling decisions.

Challenges. While the intuitive method above can potentially attain both fairness and efficiency, we find there nonetheless exist several challenges due to the distinct characteristics of DLT jobs.

First, when serving a prioritized job with additional GPU resources, its accuracy performance—if proportionally increasing the sample processing amount per iteration—may be negatively affected [21, 23, 26]. Second, given that the active job number in a cluster (so for the per-job fair share) may fluctuate drastically, it is hard to efficiently and also accurately obtain the relative job scheduling order under an idealized fair-sharing scheduler. Third, at the end of each training iteration, workers of a DLT job need to synchronize their gradients via remote communication; with larger parallelisms, the synchronization overhead may become non-negligible, potentially compromising the resource utilization [7, 23]. We will further study and address those challenges in the next section.

3 Ares

In this section, we introduce the design of Ares, a *fair* and *efficient* scheduler for DLT jobs in shared GPU clusters, without accuracy loss or severe utilization degradation. We also theoretically demonstrate that Ares can guarantee job completion within a bounded delay compared to that under idealized fair scheduling.

3.1 Endo-Elastic Training for Accuracy Validity

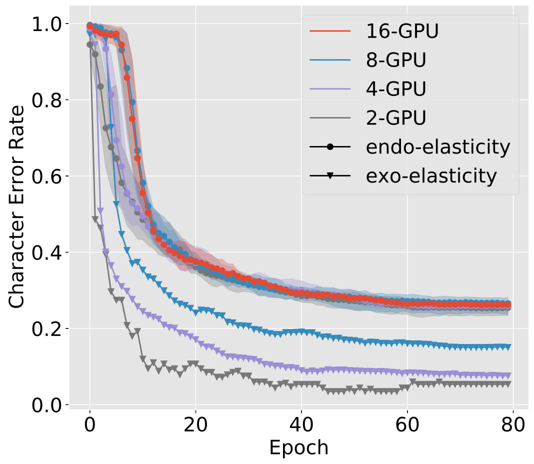
Note that achieving the scheduling effect in Fig. 1b requires that DLT jobs utilize more resources than they originally demand. There are two paradigms for a DLT job to utilize additional resources: *exo-elastic* training—keep the local batch size on each worker untouched yet proportionally increase the global batch size, and *endo-elastic* training—keep the global batch size unchanged yet reversely shrink the local batch size. To attain model training speedup without accuracy degradation, we adopt the *endo-elastic* paradigm, with which we can shrink or expand the *degree-of-parallelism* (DoP) of a DLT job without compromising model training quality. In this way, we thus can attain both *work-conservation* (i.e., flexibly expand resource consumption) and *accuracy-consistency* (i.e., models trained with identical user-specified parameters yield the same accuracy). In contrast, if with *exo-elastic* scaling—meaning that the global batch size also changes), the ultimate model training accuracy would be compromised [34, 56].

We further empirically compare the impact of endo-elasiticity and exo-elasticity on job accuracy performance, the results shown in Fig. 2. It illustrates the character error rate for training DeepSpeech2 [4] on VoxForge [1] as well as the top-5 accuracy for training ResNet50 [18] on ImageNet [9]. For *endo-elasticity*, we vary the DoP from 2 to 16 while maintaining a constant global batch size (640 for DeepSpeech2 and 6400 for ResNet50), and conduct experiments respectively with three different random seeds. For *exo-elasticity*, we vary the DoP from 2 to 16 while changing the global batch size linearly, maintaining a constant local batch size per GPU (40 for DeepSpeech2 and 400 for ResNet50). Our results show that, when changing the DoP with a constant global batch size (and identical random seeds), the final accuracy variations are always within 2% (for reference, the maximum accuracy gap with different random seeds but under the same parallelism level is already 4.5%). In contrast, the exo-elastic method of changing the global batch size linearly with the parallelism level results in a maximum accuracy variation of 23% which is unacceptable. These observations are consistent with existing works like [26].

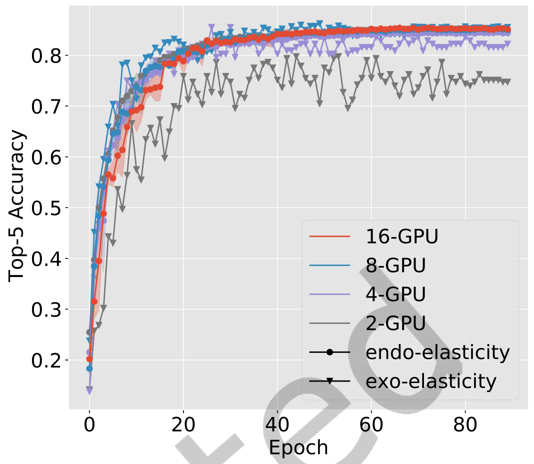
3.2 Elastic Fair Queuing

To implement the EDF-like scheduling algorithm described in Fig. 1, we need to obtain the fair completion time as the scheduling priority. Given the dynamic and unpredictable job arrival patterns in realistic scenarios, it is inefficient to keep refreshing the fair completion time of each active job at runtime. We thus need to find a method to determine the scheduling priority of each incoming job in one shot, without the need to make re-estimation upon the arrival or completion of any other job.

We note that, with endo-elasticity (such that a job can theoretically monopolize the cluster), the DLT job scheduling problem we face is similar to *packet scheduling* on a switch. In network scheduling domain [14], the link bandwidth shall be fairly shared among different flows, each consisting of multiple packets. However, each switch port can transmit only one packet at a time, and the network scheduler must determine which packet to serve to emulate idealized fair sharing effect. In that case, it is also expected that no packet is transmitted later than it would have under instantaneous fair sharing—similar as our objective for each DLT job. Meanwhile, with endo-elastic demand modeling, each (fluid) DLT job can be treated as a network packet to be served with



(a) Character Error Rate for training DeepSpeech2 on VoxForge.



(b) Top-5 Accuracy for training ResNet50 on ImageNet.

Fig. 2. When training models with different GPU numbers, endo-elastic training can yield consistent accuracy, yet exo-elastic training would cause unstable accuracy performance.

full backend capacity. With the *same demand model* and also the *same scheduling objective*, we can transfer the classical solution in network scheduling research to our problem.

In network scheduling, *fair queuing* (FQ) [8, 40] is a cornerstone method to determine the scheduling order of different packets. Note that a router can only transmit one packet at one time; given N flows sharing the link, FQ deliberately selects the flow to transmit the next packet, with the objective to mimic the effect that each flow evenly shares the link bandwidth (in particular to prevent flows with large packets from consuming more bandwidth than other flows). In implementation, FQ calculates a virtual theoretical departure time for each data packet, defined as the time it would depart under *Generalized Processor Sharing* (or GPS, an idealized fair sharing scheduler where the resource demands are *arbitrarily divisible* and always evenly allocated to each flow), and uses it as the packet scheduling priority. In particular, when the number of pending packets changes, instead of changing the expected transmission completion time of each active packet, FQ changes the *scale of the time axis*, which is much more efficient. Moreover, in this way, the fair completion "time" of each packet (used as its scheduling priority) can be fixed at packet arrival time.

For efficient and fair execution of DLT jobs in shared GPU clusters, we thus propose Elastic Fair Queuing (EFQ)—applying the insight of FQ to DLT job scheduling with endo-elastic training. To get the job scheduling order, EFQ also uses the idealized fair scheduler (GPS) as the reference system in the background.

Specifically, EFQ defines the virtual time $V(t)$ as a function of the real time t as follows:

$$\begin{aligned} V(0) &= 0, \\ \frac{\mathrm{d}}{\mathrm{d}t}V(t) &= M/N_t. \end{aligned} \tag{2}$$

Here, M is the total GPU number and N_t is the number of active jobs in GPS at time t. M/N_t thus represents the instantaneous fair share at that moment. Further, $V(t)$ can be interpreted as increasing at the marginal rate at which jobs receive services in GPS. When a job-j arrives at time a_j, Ares acquires its execution time (GPU×time, denoted by L_j) with the methods discussed in Sec. 2.3. EFQ then associates the job with its *virtual finish time*

F_j—the time at which the job would complete in GPS, which is defined as follows:

$$F_j = V(a_j) + L_j. \tag{3}$$

The virtual finish time of a job, once calculated, requires no update in the future. This is because the one with the smallest F_j will always complete first under GPS, and thus has the highest scheduling priority in EFQ. The scheduling complexity is—similar to FQ—$O(\log N)$ where N is the number of jobs.

Note that however, our EFQ strategy differs with FQ in that we allow a new-coming job to *preempt* existing ones as long as its virtual completion time is smaller; in contrast, in FQ the packet transmission process is non-preemptable. For typical DLT jobs which may take hours or even weeks to finish, it is indeed a common practice to enable preemption to avoid head-of-line blocking [16, 49]. Our testbed evaluations later reveal that such preemption overhead is negligible compared to the performance benefit.

3.3 Bounded Utilization Degradation

As aforementioned in Sec. 2.3, DLT jobs often exhibit non-linear scaling curves, which must be properly handled when prioritizing a job in scheduling. In previous example in Fig. 1, we in fact assumed that each job's total resource consumption (GPU×time) be a fixed constant; however, due to the synchronization overhead, the relationship between a DLT job's throughput and the allocated GPU number is not linear but exhibiting a *non-linear pattern with decreasing marginal benefit.*

Fig. 3 shows the variation of per-GPU throughput as well as the overall speedup when training six models with different GPU numbers (please refer to Sec. 5.1 for the detailed introduction of those models). It shows that the job speedup has a non-linear scaling curve with the GPU allocation amount, and the non-linearity levels are heterogeneous across different models. For instance, the ResNet-50 model demonstrates an almost-linear scaling curve, whereas the NeuMF model exhibits the worst scaling effect due to its small size, making it unsuitable for distributed training. These observations are consistent with existing works [15, 43]. For a job with strongly-nonlinear scaling curve, prioritizing it with an over-large parallelism may substantially compromise the resource utilization and inflate the JCT of other jobs backlogged after it. Therefore, such excessive resource allocation shall be avoided in scheduling.

To prevent low resource utilization and job backlogging, we seek to limit the maximum job parallelism to avoid excessive allocation to jobs with poor scalability. We note that such a parallelism limitation shall be made adaptive to different jobs based on their own scaling curves. Thus we introduce a *scaling control threshold* (α), which bounds the worst-case utilization degradation ratio for each prioritized job under Ares. For each job to be scheduled under EFQ, we incrementally double its GPU allocation starting from its originally-demanded value, halting if the normalized per-GPU efficiency falls below α. In particular, an over-large threshold would prevent jobs to use idle GPUs in lightly-loaded periods, and an over-small threshold would cause low utilization in heavily-loaded periods. In practice we set the default value of α to 0.75, and our experiments in Section 5.5 demonstrate that it can make a good balance and maximize the overall performance.

Putting all the above techniques together, we illustrates how Ares works with the example shown in Fig. 4. In Fig. 4, the job completion order under GPS (the reference system) is job-1, job-3 and job-2, which is then the prioritizing order under Ares. Note that job-1 and job-3 experience a significant efficiency degradation when allocated over 4 GPUs (violating the scaling-control threshold), whereas job-2 does not. Consequently, job-1 is initially allocated four GPUs and job-2 two. At time 2, when job-3 arrives, it preempts job-2. At time 3, when job-1 completes, job-3 is allocated 4 GPUs (given the scaling-control threshold α). Finally, when job-3 finishes at time 5.5, job-2 gains all the GPUs and complete its remaining computation. In this way, the average JCT of the three jobs is substantially reduced, and meanwhile no job is in fact delayed compared to the cases with idealized fair sharing.

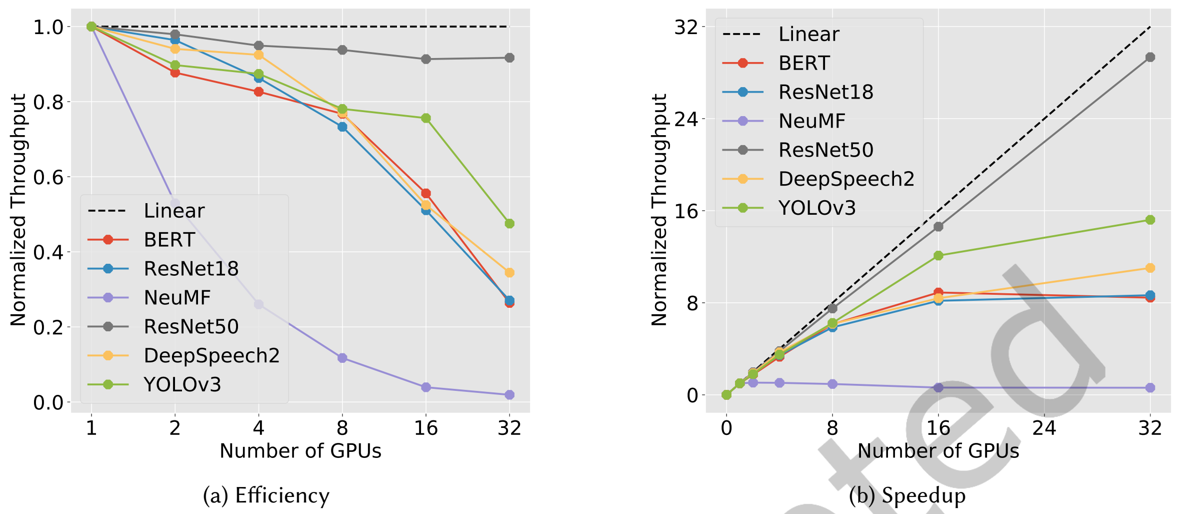


Fig. 3. Non-linear scaling characteristics of different distributed training jobs. (a) The single GPU efficiency degradation when the parallelism doubles. (b) The job speedup achieved with different numbers of GPUs.

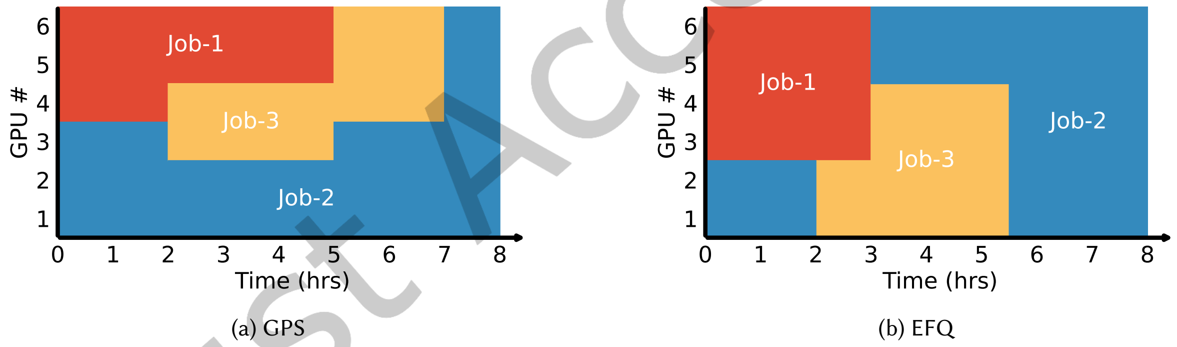


Fig. 4. Illustration of GPS and EFQ. GPS (Generalized Processor Sharing) rigidly allocates a fair share to each active job; EFQ uses the completion order under GPS as the service priority—with a scaling-control threshold (α) to implicitly bound the maximal allocation of a prioritized job.

3.4 Delay Analysis

We next show that Ares can provide worst-case performance guarantee for general DLT jobs. Specifically, for an arbitrary job, we seek to theoretically compare its JCT under EFQ (i.e, f_j) with that under GPS (i.e., $\bar{f_j}$). We let jobs be indexed in ascending order of the arrival time, i.e., job-j is the j^{th} job that is served in GPS. For job-j, we let job-i be the one with the smallest index whose arrival starts a *busy period* containing job-j, which means that all the GPUs are busy during $[a_i, f_j]$. We denote the maximum busy period length of a cluster as $\mathcal{L}$, and Table 1 summarizes the main notations used in our analysis. Then we have the following theorem:

Table 1. Summary of important notations and definitions.

M	The number of GPUs in the cluster.
$\mathcal{L}$	The maximum busy-period length of a cluster.
L_j	The GPU-time of job-j with default number of GPUs.
$\bar{f}_j$	The job finish time of job-j in GPS.
f_j	The job finish time of job-j in EFQ.
a_j	The arrival time of job-j.
α	The worst efficiency controlled by EFQ.

Theorem 1 (Delay Bound Under EFQ): Under EFQ, DLT jobs are guaranteed to complete within a bounded delay after it completion under GPS. Specifically, for each job-j, we have

$$f_j - \bar{f}_j \leq (\frac{1}{\alpha} - 1)\mathcal{L}. \tag{4}$$

Proof. While EFQ preferentially offers GPUs to jobs in ascending order of their GPS completion times, jobs may start services out of order due to dynamic arrivals. In our analysis, we focus on the busy period $[a_i, f_j]$. Specifically, we categorize all the jobs arriving in the interval $[a_i, f_j]$ into three types:

- The first type consists of jobs that arrive before job-j and complete also earlier than job-j (but no earlier than a_i) under GPS, causing job-j to wait under EFQ. This job set is denoted as $\mathcal{A} = \{k \mid a_i \leq a_k < a_j \ and \ a_i < \bar{f}_k < \bar{f}_j\}$, and its total GPU-time consumption within the range $[a_i, \bar{f}_j]$ under GPS is denoted by $R(\mathcal{A})$.
- The second type consists of jobs that arrive no earlier than job-j but complete no later than job-j under GPS, potentially causing job-j to be preempted during execution under EFQ. This job set is denoted as $\mathcal{B} = \{k \mid a_j \leq a_k \ and \ \bar{f}_k \leq \bar{f}_j\}$, and the total GPU-time within the range $[a_i, \bar{f}_j]$ under GPS is $R(\mathcal{B})$. Note that job-j itself is an element in $\mathcal{B}$.
- The third type consists of jobs that arrive after the arrival (but before the completion) of job-j and complete later than job-j under GPS. These jobs would complete also later than job-j under EFQ. This job set is denoted as $C = \{k \mid a_j < a_k < \bar{f}_j \ and \ \bar{f}_j < \bar{f}_k\}$, and the total GPU-time within the range $[a_i, \bar{f}_j]$ under GPS is $R(C)$.

We then respectively analyze the overall GPU-time consumption during $[a_i, \bar{f}_j]$ under GPS and during $[a_i, f_j]$ under EFQ.

We first consider the GPS reference system. We note that jobs in $\mathcal{A} \cup \mathcal{B}$ all complete no later than job-j in GPS. Therefore, starting from the moment job-i arrives, GPS has finished at least $R(\mathcal{A}) + R(\mathcal{B})$ GPU-time before $\bar{f}_j$. We let L_k represent the GPU-time of job-k, and assume there are totally M GPUs in the cluster. Since the jobs in C are also serviced with its fair share during $[a_i, \bar{f}_j]$ under GPS, we have

$$\begin{aligned} \bar{f}_j &\geq a_i + \frac{1}{M}[R(\mathcal{A}) + R(\mathcal{B})] \\ &= a_i + \frac{1}{M}\sum_{k \in \mathcal{A} \cup \mathcal{B}} L_k. \end{aligned} \tag{5}$$

We next consider the realistic EFQ system over the interval $[a_i, f_j]$. Since we serve jobs in the order of fair completion time, by the time we complete job-j under EFQ, we will have already completed the jobs in $\mathcal{A} \cup \mathcal{B}$.

Meanwhile, no job in C has been serviced because they have lower priority than job-j. Note that in EFQ the per-GPU efficiency may suffer a slow-down of no more than $\frac{1}{\alpha}$, we thus have

$$\begin{aligned} f_j &\leq a_i + \frac{1}{M}\frac{R(\mathcal{A}) + R(\mathcal{B})}{\alpha} \\ &= a_i + \frac{1}{M \cdot \alpha}\sum_{k\in\mathcal{A}\cup\mathcal{B}} L_k. \end{aligned} \tag{6}$$

Subtracting Eq. 5 from Eq. 6, we can obtain

$$f_j - \bar{f}_j \leq (\frac{1}{\alpha} - 1)\frac{1}{M}\sum_{k\in\mathcal{A}\cup\mathcal{B}} L_k. \tag{7}$$

Since we bound the maximum busy period length by $\mathcal{L}$—that is, $\sum_{k\in\mathcal{A}\cup\mathcal{B}} L_k \leq \mathcal{L} * M$, we further have

$$f_j - \bar{f}_j \leq (\frac{1}{\alpha} - 1)\mathcal{L}. \tag{8}$$

□

Remarks. We make two remarks on Theorem 1.

First, the delay bound is determined by the scaling control threshold α as well as the maximum busy period length, and the resultant value is usually not large in most production clusters. On the one hand, α is a tunable knob usually set close to 1, rendering the item $\frac{1}{\alpha} - 1$ close to 0. Meanwhile, GPU resources are often *under-utilized* in production clusters [51, 52], thus the busy period durations are usually not long.

Second, Theorem 1 states that EFQ will not delay a job long after its GPS completion in the worst case; yet in fact, most jobs can complete much faster under EFQ than under GPS. This is because EFQ indeed emulates a SJF-like method that can attain a near-optimal average JCT. We will verify the superiority of EFQ in both worst-case and average job performance through extensive experiments in Sec. 5.

3.5 Discussions

Integration with 3D Parallelism. While in this paper we focus on data parallelism (DP, meaning that allocating more GPUs would proportionally increase the number of parallel batch processing streams), there do exist two other parallelizing dimensions: tensor parallelism (TP) [33] and pipeline parallelism (PP) [22, 38], which are primarily proposed to host a large model over multiple GPUs. Although they differ in some characteristics (e.g., the bandwidth consumption of DP is in general smaller than TP but larger than PP [55]), the general trend always holds that more GPUs would yield a larger throughput. In that sense, our Ares scheduler is also effective in cases with 3D parallelism; we will explore such solution extension (e.g., granting a prioritized job the flexibility to select the best parallelizing scheme) in the future.

Compatibility for jobs with unknown resource consumption. Ares relies on the estimated job resource demand (GPU×time) when making scheduling decisions, which is often feasible as explained in Sec. 2.1. Yet, there do exist some cases where such estimation is infeasible or highly inaccurate (for example, when training stops only if the accuracy/loss curve—with irregular shapes—reaches a given target). For compatibility, we can host such information-agnostic jobs with a dedicated resource pool and schedule them under the default scheme (e.g., instantaneous fair sharing)—without intervening (prioritizing or temporally delaying) their execution.

Mapping to heterogeneous GPUs. Ares functions by properly determining the scheduling order and allocation amount of each DLT job, not involving how to map the specific GPUs to a job so far. This is a trivial issue in a homogeneous cluster (with identical GPUs) [21, 43, 55]. In cases with heterogeneous GPUs, we can use the average per-GPU computing power to quantify jobs' GPU×time demands (for virtual-time estimation in Eq. 2),

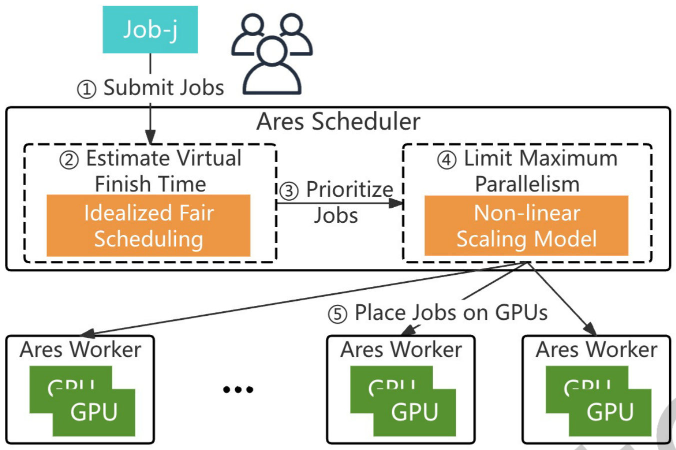


Fig. 5. Ares workflow.

and further adopt a simple heuristic that grants a prioritized job the privilege to select any available GPU nodes it prefers (e.g., based on the sorted GPU computing power or GPU consolidation effect).

4 Implementation

We have implemented Ares with over 7,800 lines of Python code. We use PyTorch [41] DistributedDataParallel (DDP) for distributed training, NCCL [3] for communication between DL training workers and Linux NFS [2] for checkpoint storage and recovery.

Scheduling Interface. Ares currently supports DLT jobs written in PyTorch and provides an elastic training framework (`AresTrainer`) for user interaction. Users need to subclass `AresTrainer`, which involves defining their DL model, dataset, optimizer, as well as implementing the forward-backward process in the functions `train_acc_step()` and `val_acc_step()`. Users submit application-level hyper-parameters such as batch size, learning rate and the default GPU demands to Ares.

Overall Workflow. Ares consists of an Ares scheduler and an Ares worker on each available GPU machine. Communication between the scheduler and workers is carried out via RPC. As shown in Fig. 5, the scheduler starts by checking for new job submissions (①) and estimating the job's virtual finish time (②) for each newly-arriving job. Once a new job arrives or an existing job finishes, Ares makes scheduling decisions based on the current resources and job states. Specifically, it repeatedly prioritizes the job with the earliest virtual finish time (③) and limits the maximum job parallelism (④) to bound the worst-case utilization degradation, until all the pending jobs are scheduled. Finally, Ares tries to place jobs on idle GPUs using a *topology-aware* method to minimize fragmentation and maximize job locality (⑤)—similar to [15].

Preemptive and Elastic Training. Ares's elastic training framework, `AresTrainer`, supports all necessary functionalities for preemptive and elastic training. When a DLT job's allocation changes, the scheduler notifies all its training processes to exit gracefully—first capturing termination signals, then saving the necessary checkpoints of the latest model parameters and optimizer states to NFS, and finally releasing all GPUs allocated to the job. If an job is still allocated some GPUs (although at a different quantity) according to the latest scheduling results, Ares will adjust each worker's local batch size to maintain the same global batch size before reissuing the

Table 2. Models used in the evaluation.

Task	Model	Dataset	Batch Size	Size
Image Classification	ResNet50 [18]	ImageNet [9]	6400, 12800	XL
	ResNet18 [18]	CIFAR-10 [30]	2048, 4096	S
Object Detection	YOLOv3 [47]	PASCAL-VOC [12]	256, 512	L
Question Answering	BERT [11]	SQuAD [46]	192, 384	M
Speech Recognition	DeepSpeech2 [4]	CMU-ARCTIC [29]	320, 640	M
Recommendation	NeuMF [19]	MovieLens [17]	16384, 32768	S

execution commands. If the job cannot support the specified batch size due to insufficient GPU memory, *gradient accumulation* [6] would be employed. As we show later in Sec. 5, both the per-preemption time overhead (Sec. 5.2) and the average preemption-counts (Sec. 5.6) are quite limited in our experiments.

Throughput Profiling. Ares ensures cluster utilization and job fairness based on the non-linear scaling curve of each job. To construct the non-linear performance model, we modify PyTorch DDP to profile the communication overhead. For each job, Ares first pre-runs it to profile its throughput with different numbers of GPUs. For each DoP setup, we use the first 10 iterations as the profiling benchmark (after excluding the initial iterations). This profiling takes only about ten seconds yet suffices for our purpose. By modeling the non-linear performance and allocating resources at *powers-of-two* numbers, we can complete the necessary pre-profiling tasks in an average time of about two minutes, which is negligible compared to the long model training time of typical DLT jobs.

5 Evaluation

In this part we compare the performance of Ares with a series of state-of-the-art DLT schedulers. Similar to the methodology in existing works [24], our evaluation includes both testbed experiments and simulations.

5.1 Setup

Platform. For testbed experiments, we build a 16-GPU cluster, where there are four nodes each equipped with 4 NVIDIA GeForce RTX 2080 Ti GPUs, 2 Intel Xeon Gold 6230 CPUs and 512GB DDR4 RAM. The network follows the Fat-Tree topology with 40 Gbps links connected to a 100-Gbps switch. Meanwhile, to evaluate our scheduler at a larger scale, we also write a cluster simulator. All our simulations were performed on a 64-GPU (16-node) cluster, using throughput data measured on realistic testbed.

Workloads. For generality, we adopt three trace sets all profiled from production clusters: *Philly* From Microsoft [25], *Helios* from SenseTime [20], and *newTrace* reported by the Sia work [24]. The first two trace sets are from moderately congested clusters and the last is from a much busier cluster. Meanwhile, all original job submission records are categorized into four classes based on the GPU time: Small (0-1 hrs), Medium (1-10 hrs), Large (10-100 hrs), and X-Large (>100 hrs). For each record, we map it to a job in Table 2 with the corresponded size.

Baselines. We compare Ares with seven DLT job schedulers: Gavel [39], Themis [36], AlloX [32], Tiresias [16], Lucid [21], Optimus [42], and Pollux [43]. For Pollux we use the default parameter $p = -1$, which as claimed in [43] can achieve the best efficiency-fairness trade-off (we also test with $p = 1$ and $p = -10$ and the results are not substantially different). In Gavel, we use the Max-Min Fairness [37] scheduling policy. We choose a scheduler

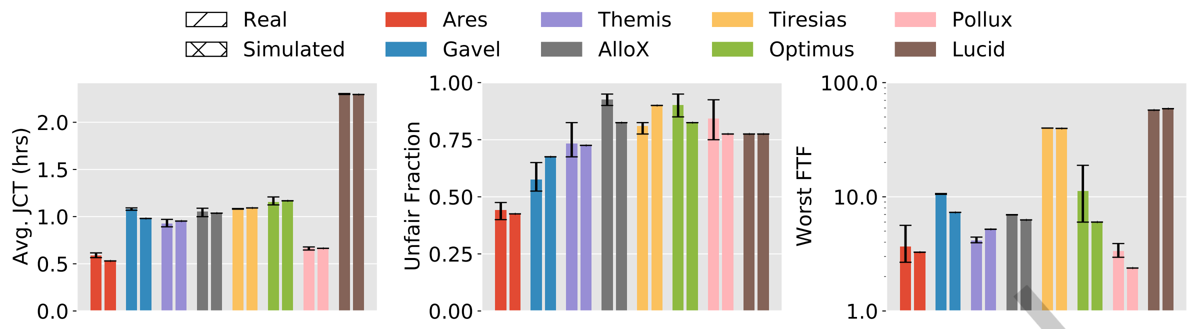


Fig. 6. Avg. JCT, unfair fraction and worst-case FTF (of testbed experiments and the corresponded simulations) when replaying a Philly trace under different schedulers.

round duration of 60s for Ares, Tiresias, Lucid, Optimus and Pollux (same as [43]), and choose 360s for Gavel, Themis and AlloX (same as [39]). For Ares, the scaling control threshold is set to 0.75.

Metrics. Ares aims to behaving well in both efficiency and fairness. We measure the *average JCT* to evaluate scheduler efficiency. Regarding fairness, we adopt *finish time fairness* (FTF) [36], which is defined as the ratio between *realistic JCT* and the *expected JCT under instantaneous fair sharing*. In particular, we measure the *unfair fraction* (the proportion of jobs whose FTF value is larger than 1) as well as the worst-case FTF to depict the level of fairness guarantee.

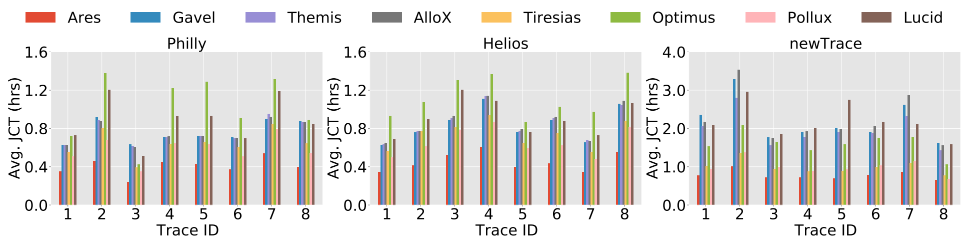
5.2 Testbed Experiments

For testbed experiments, we sample a trace period from the Philly trace set spanning a 4-hour submission window with totally 40 jobs. We repeat each experiment for three times to eliminate the effects of randomness.

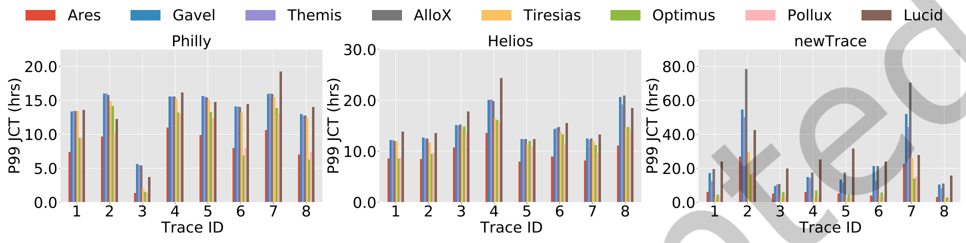
Overall Testbed Performance. In Fig. 6, we measure the overall job performance under different schedulers. In the efficiency aspect, Ares can reduce the average JCT by 12.2-74.5% compared with the other schedulers. In the fairness aspect, Ares can reduce the unfair fraction by 23.1-86.5%, and reduce the worst-case FTF by 11.5-93.6%. These results confirm that, our Ares scheduler, with the elastic fair queuing strategy, can behave well in both efficiency and fairness. Meanwhile, we also notice that those methods do achieve comparable accuracy performance: for example, the ResNet50 model (trained for 90 epochs) always reaches a test accuracy between 0.540 to 0.546 under each scheme.

In our testbed experiments, we have also measured the preemption overhead of each job. For ResNet50, ResNet18, YOLOv3, BERT and DeepSpeech2 (NeuMF is a small job never preempted), the time cost to stop and relaunch the DLT job is respectively 4s, 8s, 4s, 73s and 1s, all negligible compared to the entire model training time (the total preemption-count of each job is also quite limited as in Fig. 12). These numbers are later used in our simulations.

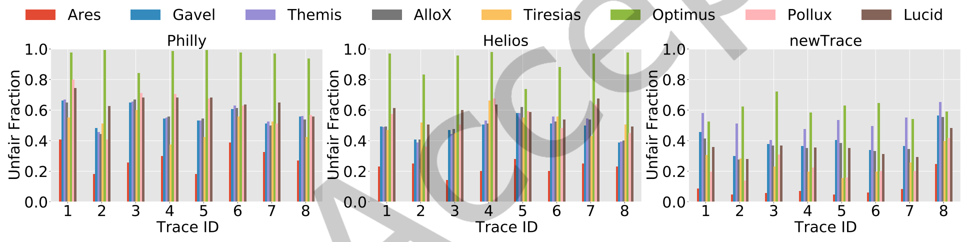
Simulator Fidelity. We also check the simulator fidelity with Fig. 6. We compare Ares and the baseline schedulers in our simulator under the same configurations of the above testbed experiments. As shown in Fig. 6, the performance differences between testbed experiments and the corresponded simulation results are quite small (mostly less than 5%) for all of the three performance aspects (avg. JCT, unfair fraction and worst-case FTF). It thus confirms that our simulator has a high fidelity, and our later simulation results at a larger scale can faithfully depict the true scheduling performance in reality.



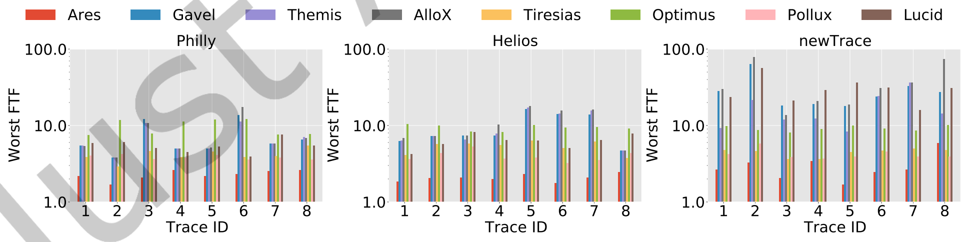
(a) Average Job Completion Time under different schedulers respectively with the thee trace sets.



(b) P99 Job Completion Time under different schedulers respectively with the thee trace sets.



(c) Unfair Job Fraction under different schedulers respectively with the thee trace sets.



(d) Worst-case Finish Time Fairness under different schedulers respectively with the thee trace sets.

Fig. 7. Simulation results for Ares and baseline schedulers on various aspects: (a) Average Job Completion Time, (b) P99 Job Completion Time, (c) Unfair Job Fraction, and (d) Worst Finish Time Fairness Ratio across eight traces respectively sampled from Philly, Helios, and newTrace.

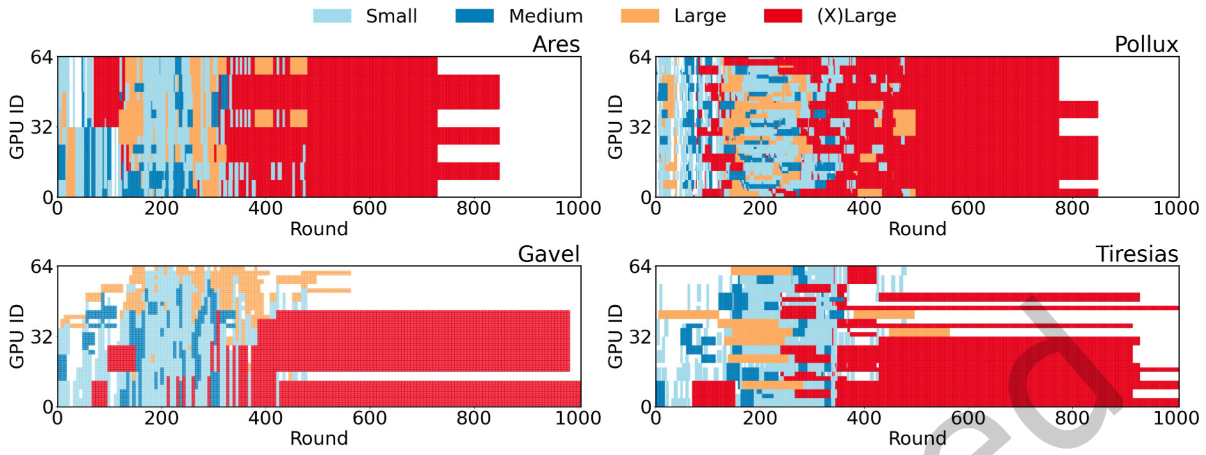


Fig. 8. Instantaneous GPU allocation status under Ares, Optimus, Pollux, and Tiresias when replaying a Helios trace.

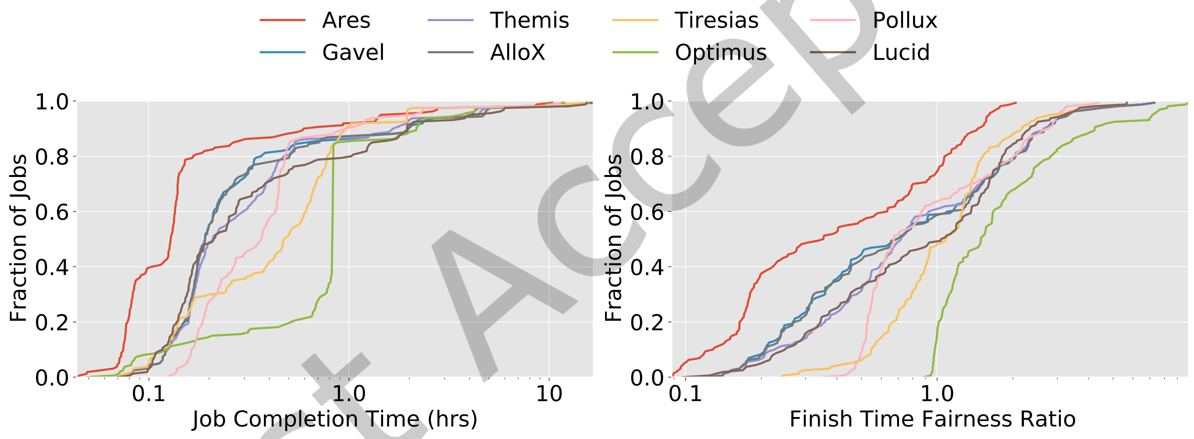


Fig. 9. CDFs of JCTs (left) and FTFs (right) respectively under Ares, Optimus, Pollux, and Tiresias.

5.3 Overall Simulation Results

For our simulation of the 64-GPU cluster, we respectively sample eight traces from the three trace set: from sets Philly and Helios, each sampled trace consists of 160 jobs submitted over 8 hours; from set newTrace, each sampled trace consists of 960 jobs submitted over 48 hours. Fig. 7 presents the scheduling performance of different schedulers for each trace.

Regarding efficiency performance, we find that Ares can also yield the shortest average JCT: compared with the second best (Pollux), Ares achieves an improvement of over 20%. We note that such efficiency improvement is larger in lightly-loaded clusters (30.3% for Philly and 31.3% for Helios) than in heavily-loaded ones (21.4% for newTrace). This is because there are more backlogged jobs in heavily-loaded clusters, where the GPU-time inflation of earlier jobs are more likely to impact the later ones in Ares. Given that the reported resource utilization

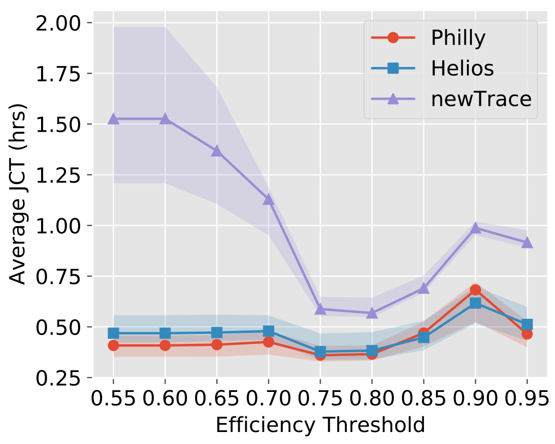


Fig. 10. The median Avg. JCT for Ares across different values of scalability control threshold α. The shaded region represents the 25^{th} and 75^{th} percentiles.

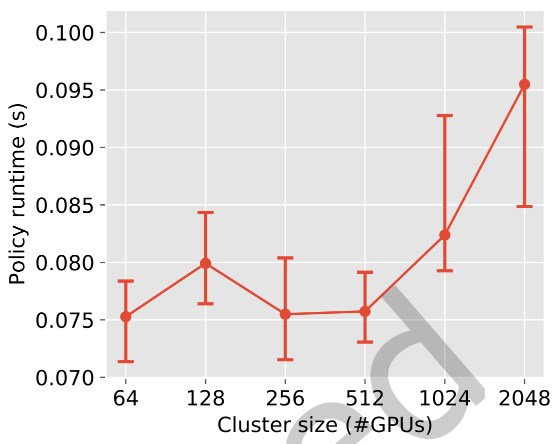


Fig. 11. The median policy runtime for Ares across different cluster sizes using proportionally-sized Helios traces. The error bars indicate the 25^{th} and 75^{th} percentiles.

is generally low in production clusters [20, 25, 51], Ares can be expected to bring large efficiency improvement in practice.

Regarding fairness performance, Ares can also yield the best unfair fraction and worst-case FTF. For example, for Philly trace, on unfair fraction, Ares surpasses the second best by 41.32%, and on worst-case FTF, it surpasses the second best by 44.17%. We note that the worst-case FTF might be larger than our theoretical bound (Sec. 3.4), because in simulation we retain the realistic job preemption overhead. That said, the fairness performance of Ares is still the best among all the schedulers we evaluate.

5.4 Performance Deep Dive

We further dive deep into the performance benefit of Ares when simulating with a Helios trace. In Fig. 8, we show the instantaneous resource allocation over the entire cluster for the four representative schedulers: Ares, Pollux, Tiresias and Themis. From Fig. 8, we can see that Themis and Tiresias fall short because they are not elastic: jobs under the two schedulers cannot fully utilize the available resources even when there are competitors, leading to an average JCT (as well as the makespan) larger than Ares and Pollux. Further, when comparing Ares with Pollux, we can see that Ares adopts the fair-queuing-style allocation without sticking to instantaneous fair allocation. As suggested in Fig. 1, Ares can thus reduce the average JCT while attaining comparable makespan.

In Fig. 9, we further show the cumulative distribution function (CDF) of JCT and FTF under the four schedulers. From the left plot of Fig. 9, we can observe that over 90% jobs can be accelerated by Ares, which is consistent with the toy example of Fig. 1: by prioritizing jobs with proper order, most jobs can complete faster than it would have under instantaneous fair sharing. In the fairness aspect, according to the right plot of Fig. 9, the portion of jobs with a FTF ratio larger than 1 is also smaller under Ares than under other schedulers—thanks to the appropriate queuing order obtained under our EFQ algorithm.

To summarize, these deep-dive results align with the previous results in Fig. 6 and Fig. 7, all demonstrating that Ares can surpass existing schedulers in both fairness and efficiency.

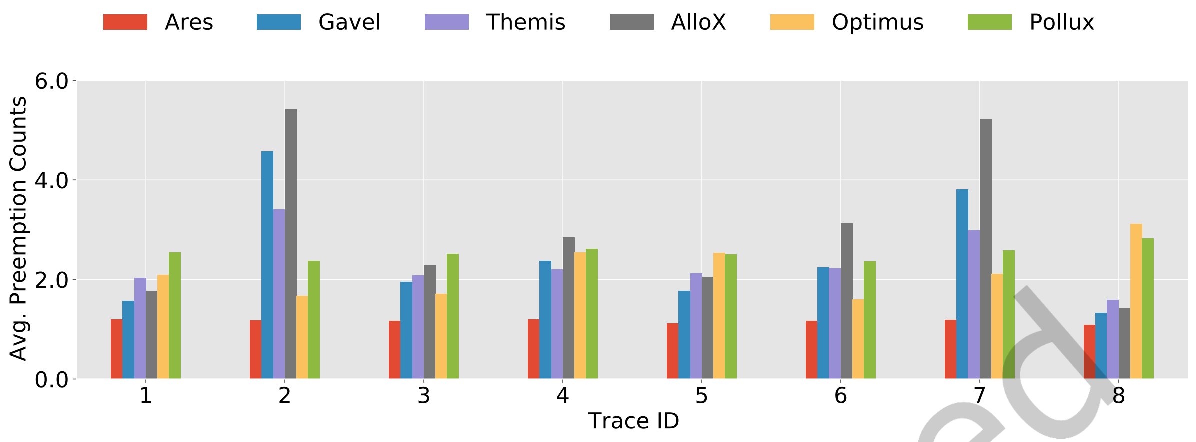


Fig. 12. The avg. preemption counts under different schedulers with the newTrace (for preemptive schedulers).

5.5 Sensitivity Analysis

Note that in Sec. 3.3 we use a threshold α to control the worst-case utilization degradation when prioritizing a job with enlarged parallelism, and here we check the impact of that threshold on Ares performance with all the three trace sets. From Fig. 10, we can see that the average JCT first decreases and then increases with α, and the optimal range is between 0.75 and 0.8. Regarding the reasons behind, on the one hand, setting α above 0.8 restricts maximal job parallelism, which can achieve high utilization for the allocated resources but may leave some resources unassignable; on the other hand, setting α below 0.75 compromises resource utilization, leading to significant increase in Avg. JCT due to job backlog (especially in heavily-loaded scenarios like newTrace). Therefore, it is indeed an appropriate choice to set α to 0.75.

5.6 Overhead Analysis

To check the scalability of our Ares scheduler, we measure Ares overheads in various scenarios with different cluster sizes, where the job arrival rate is proportionally increased with the cluster size. Fig. 11 shows the results, and we can observe that Ares' computational overhead indeed grows almost *logarithmically* with the cluster size (consistent with our analysis in Sec. 3.2). Moreover, the maximum scheduling overhead is indeed quite small in absolute time: for a cluster of 2048 GPUs with a job arrival rate of 640 jobs/hour, the scheduling overhead of Ares is less than 0.06s, negligible when compared to the typical JCT of DLT jobs.

Besides, in Fig. 12, we further show the average preemption times of each job under different schedulers over the newTrace dataset (with a relatively intense load level). From Fig. 12, we learn that the number of preemptions during the lifetime of a job is quite small; for Ares it is less than twice per job (smaller than others because it applies a stable GPS-based scheduling priority and also sets a scaling-control threshold to avoid excessively taking any newly-released resources).

6 Additional Related Work

Parallelization Strategies and Optimization. To accommodate large models that exceed the capacity of a single GPU, techniques such as tensor parallelism [33] and pipeline parallelism [22, 38] have been developed to distribute models across multiple GPUs. To further conserve GPU memory, the ZeRO series [44, 45, 48] offloads computations and weights to host memory. For accelerated job execution, Alpa [55] automates inter-operator and intra-operator parallelism. Similarly, Whale [26] automatically determines parallel strategies by considering the capabilities of heterogeneous GPUs. Although these methods offer various options for optimizing job execution plans, Ares primarily focuses on fair and efficient resource scheduling among jobs within a shared cluster, rather than adjusting individual job execution plans. Ares is orthogonal to these researches on parallel strategies and optimal job plan searching and can transparently support them. The further integration of adaptive parallelism and scheduling to enhance shared cluster performance is an interesting future direction.

Adaptive Batch Size Training. To enhance the efficiency of deep learning training, several studies explore dynamic batch size adjustments. AdaBatch [10] synchronously increases the batch size and learning rate during training iterations. CABS [5] dynamically adjusts the batch size and learning rate based on gradient statistics. Pollux [43] further integrates the statistical efficiency gains from adjusting these parameters at the application level with the dynamically available GPU resources at the hardware level, thereby accelerating training. However, some works emphasize that adjusting the batch size can significantly degrade model performance [21, 28, 34, 56], and our experiments have reached the same conclusion. To ensure model performance consistency with user-defined parameters, we refrain from altering the hyper-parameters of training jobs that impact model accuracy.

7 Conclusion

In this paper, we introduce Ares, an efficient and fair scheduler for model training jobs. Ares utilizes the concept of virtual finish time from network fair queuing methods, enabling efficient estimation of job completion order at arrival time. We prioritize the jobs with earlier virtual finish times and allow them to elastically use more resources than they originally demand to attain fast completion, as long as their resource utilization degradation is under a preset threshold. Our real cluster experiments and large-scale trace-driven simulations show that, compared to state-of-the-art schedulers, Ares can reduce the average job completion time by over 20% and decrease the number of unfairly-served jobs by over 40%.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (NSFC) (Grant No. 62202300 and 62402312). Chen Chen is the corresponding author.

References

[1] [n. d.]. Free Speech... Recognition (Linux, Windows and Mac) - voxforge.org. https://www.voxforge.org/. Retrieved on July 31, 2025.

[2] [n. d.]. Network File System (NFS). https://ubuntu.com/server/docs/network-file-system-nfs. Retrieved on July 31, 2025.

[3] 2014. NVIDIA Collective Communication Library (NCCL). https://developer.nvidia.com/nccl.

[4] 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 *PMLR ICML*.

[5] Lukas Balles, Javier Romero, and Philipp Hennig. 2016. Coupling adaptive batch sizes with learning rates. *arXiv preprint arXiv:1612.05086* (2016).

[6] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. 2024. Centauri: Enabling Efficient Scheduling for Communication-Computation Overlap in Large Model Training via Communication Partitioning. In *ACM ASPLOS*.

[7] Chen Chen, Wei Wang, and Bo Li. 2019. Round-robin synchronization: Mitigating communication bottlenecks in parameter servers. In *IEEE INFOCOM*.

[8] Alan Demers, Srinivasan Keshav, and Scott Shenker. 1989. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM Computer Communication Review* 19, 4 (1989), 1–12.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *IEEE CVPR*.

[10] Aditya Devarakonda, Maxim Naumov, and Michael Garland. 2017. Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029* (2017).

[11] 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).

[12] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2010. The pascal visual object classes (voc) challenge. *International journal of computer vision* 88 (2010), 303–338.
[13] Eric J Friedman and Shane G Henderson. 2003. Fairness and efficiency in web server protocols. *ACM SIGMETRICS Performance Evaluation Review* 31, 1 (2003), 229–237.
[14] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. 2012. Multi-resource fair queueing for packet processing. In *ACM SIGCOMM*.
[15] 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 *ACM ASPLOS*.
[16] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU cluster manager for distributed deep learning. In *USENIX NSDI*.
[17] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.
[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE CVPR*.
[19] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *ACM WWW*.
[20] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *ACM/IEEE SC*.
[21] 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 *ACM ASPLOS*.
[22] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *NeurIPS* (2019).
[23] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. 2021. Elastic resource sharing for distributed deep learning. In *USENIX NSDI*.
[24] 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 *ACM SOSP*.
[25] 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 *USENIX ATC*.
[26] Xianyan Jia, Le Jiang, Ang Wang, Wencong Xiao, Ziji Shi, Jie Zhang, Xinyuan Li, Langshi Chen, Yong Li, Zhen Zheng, et al. 2022. Whale: Efficient giant model training over heterogeneous {GPUs}. In *USENIX ATC*.
[27] Mehdi Kargahi and Ali Movaghar. 2006. A method for performance analysis of earliest-deadline-first scheduling policy. *The Journal of Supercomputing* 37 (2006), 197–222.
[28] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).
[29] John Kominek and Alan W Black. 2004. The CMU Arctic speech databases. In *SSW*.
[30] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
[31] Tan N Le, Xiao Sun, Mosharaf Chowdhury, and Zhenhua Liu. 2019. Allox: Allocation across computing resources for hybrid cpu/gpu clusters. *ACM SIGMETRICS Performance Evaluation Review* 46, 2 (2019), 87–88.
[32] Tan N Le, Xiao Sun, Mosharaf Chowdhury, and Zhenhua Liu. 2020. Allox: compute allocation in hybrid clusters. In *ACM EuroSys*.
[33] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020).
[34] Mingzhen Li, Wencong Xiao, Hailong Yang, Biao Sun, Hanyu Zhao, Shiru Ren, Zhongzhi Luan, Xianyan Jia, Yi Liu, Yong Li, et al. 2023. EasyScale: Elastic Training with Consistent Accuracy and Improved Utilization on GPUs. In *ACM/IEEE SC*.
[35] Yunteng Luan, Xukun Chen, Hanyu Zhao, Zhi Yang, and Yafei Dai. 2019. SCHED^2: Scheduling Deep Learning Training via Deep Reinforcement Learning. In *IEEE GLOBECOM*.
[36] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and efficient GPU cluster scheduling. In *USENIX NSDI*.
[37] Dritan Nace and Michal Pióro. 2008. Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial. *IEEE Communications Surveys & Tutorials* 10, 4 (2008), 5–17.
[38] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized pipeline parallelism for DNN training. In *ACM SOSP*.
[39] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-Aware cluster scheduling policies for deep learning workloads. In *USENIX OSDI*.
[40] Abhay K Parekh and Robert G Gallager. 1993. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM transactions on networking* 1, 3 (1993), 344–357.
[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural*

information processing systems 32 (2019).
[42] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *ACM EuroSys*.
[43] 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 *USENIX OSDI*.
[44] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *IEEE SC*.
[45] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *IEEE SC*.
[46] 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).
[47] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).
[48] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. {Zero-offload}: Democratizing {billion-scale} model training. In *USENIX ATC*.
[49] Abeda Sultana, Li Chen, Fei Xu, and Xu Yuan. 2020. E-LAS: Design and analysis of completion-time agnostic scheduling for distributed deep learning cluster. In *IACC ICPP*.
[50] Haoyu Wang, Zetian Liu, and Haiying Shen. 2020. Job scheduling for large-scale machine learning clusters. In *ACM CoNEXT*.
[51] 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 *USENIX NSDI*.
[52] 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 *USENIX ATC*.
[53] Zhisheng Ye, Wei Gao, Qinghao Hu, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. 2024. Deep learning workload scheduling in gpu datacenters: A survey. *Comput. Surveys* 56, 6 (2024), 1–38.
[54] Peifeng Yu, Jiachen Liu, and Mosharaf Chowdhury. 2021. Fluid: Resource-aware hyperparameter tuning engine. *Proceedings of Machine Learning and Systems* 3 (2021), 502–516.
[55] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. 2022. Alpa: Automating inter-and Intra-Operator parallelism for distributed deep learning. In *USENIX OSDI*.
[56] Pengfei Zheng, Rui Pan, Tarannum Khan, Shivaram Venkataraman, and Aditya Akella. 2023. Shockwave: Fair and efficient cluster scheduling for dynamic adaptation in machine learning. In *USENIX NSDI*.

Received 16 June 2025; revised 16 June 2025; accepted 5 September 2025