Chronus: A Novel Deadline-aware Scheduler for Deep Learning Training Jobs

Wei Gao[1,2], Zhisheng Ye[3], Peng Sun[4], Yonggang Wen[1], Tianwei Zhang[1*]
[1]School of Computer Science and Engineering, Nanyang Technological University
[2]S-Lab, Nanyang Technological University [3] Peking University [4]SenseTime
gaow0007@e.ntu.edu.sg, yezhisheng@pku.edu.cn, sunpeng1@sensetime.com, {ygwen, tianwei.zhang}@ntu.edu.sg

ABSTRACT

Modern GPU clusters support Deep Learning training (DLT) jobs in a distributed manner. Job scheduling is the key to improve the training performance, resource utilization and fairness across users. Different training jobs may require various objectives and demands in terms of completion time. How to efficiently satisfy all these requirements is not extensively studied.

We present Chronus, an end-to-end scheduling system to provide deadline guarantee for SLO jobs and maximize the performance of best-effort jobs. Chronus is designed based on the unique features of DLT jobs. (1) It leverages the *intra-job predictability* of DLT processes to efficiently profile jobs and estimate their runtime speed with dynamic resource scaling. (2) It takes advantages of the *DLT preemption* feature to select jobs with a lease-based training scheme. (3) It considers the *placement sensitivity* of DLT jobs to allocate resources with new consolidation and local-search strategies. Large-scale simulations on real-world job traces show that Chronus can reduce the deadline miss rate of SLO jobs by up to 14.7×, and the completion time of best-effort jobs by up to 19.9×, compared to existing schedulers. We also implement a prototype of Chronus atop Kubernents in a cluster of 120 GPUs to validate its practicability.

*Corresponding author.

SoCC '21, November 1–4, 2021, Seattle, WA, USA

ACM ISBN 978-1-4503-8638-8/21/11.
https://doi.org/10.1145/3472883.3486978

CCS CONCEPTS

• **Computing methodologies** → **Distributed computing methodologies**.

KEYWORDS

GPU Datacenter, Deep Learning Training, Cluster Management System, Deadline-aware Scheduler

ACM Reference Format:
Wei Gao, Zhisheng Ye, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2021. Chronus: A Novel Deadline-aware Scheduler for Deep Learning Training Jobs. In *ACM Symposium on Cloud Computing (SoCC '21), November 1–4, 2021, Seattle, WA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3472883.3486978

1 INTRODUCTION

Past years have witnessed the tremendous progress of Deep Learning (DL) in many artificial intelligence tasks. The high performance of state-of-the-art DL models is attributed to the sophisticated algorithms, complex network structures with huge numbers of parameters. Training such a model requires vast amounts of GPU resources. Consequently, IT corporations, research institutes and cloud providers build large-scale GPU clusters to ease the development of DL training (DLT) jobs. A scheduler is necessary to manage DLT jobs and allocate resources in a GPU cluster.

DLT jobs in GPU clusters have various demands, based on which they can be coarsely classified into two categories. (1) *SLO jobs* [7, 30]: job execution time is one common Service Level Objective (SLO) for GPU users. With the successful commercialization of DL technology, DLT jobs related to product development raise high SLO requirements for the completion time. DL competitions and research paper submissions also call for such SLO demand. (2) *Best-effort jobs* [53]: These are exploratory jobs for debugging and testing purposes. They do not have deadline requirements but are expected to complete as soon as possible. As common GPU clusters support the mixture of SLO jobs and best-effort jobs, the problem we attempt to address is: *how can a DL scheduler*

satisfy various demands from different types of jobs, i.e., guaranteeing the completion time of SLO jobs, while maximizing the performance of best-effort jobs?

Unfortunately, existing DL schedulers lack explicit supports for SLO requirements. They are mainly designed for the improvement of job performance [16, 21, 53, 54, 56] or fairness [4, 28, 34], thus incapable of guaranteeing the job completion before the deadlines. To our best knowledge, there are only two works considering the scheduling of SLO jobs for DL training. HyperSched [30] mainly focuses on the performance improvement of Hyper-Parameter Optimization jobs with deadlines. It cannot be applied to other general DLT jobs. GENIE [7] automatically identifies the optimal resource allocation for SLO jobs. It requires the modifications of the underlying DL framework (i.e., tensorflow [1]). It makes decisions about the number of GPUs for each job, while ignoring the resource requirements of users. Moreover, these two works do not consider the mixture of SLO and best-effort jobs, which matches the realistic scenario.

Prior works also proposed deadline-aware schedulers for traditional big data jobs [9, 29, 39, 48]. They estimate the completion time of each job from online profiling [11] or offline prediction [22]. Then they formalize the resource and SLO requirements as an optimization problem, and leverage the Mixed Integer Linear Programming (MILP) solver [51] to make scheduling decisions. These designs are not tailored to but can be extended to the scheduling of DLT jobs. Additionally, DLT jobs exhibit certain unique features distinct from big data workloads.

Consideration of these features could bring new opportunities to further improve the efficiency and effectiveness of deadline-aware schedulers specifically for DLT jobs. This is what we aim to explore in this paper.

We present *Chronus*, a novel DL scheduler to support the deadline guarantee of SLO jobs, while minimizing the average latency of best-effort jobs. Chronus achieves these goals via three key mechanisms, based on three unique characteristics of DLT jobs. First, *DLT jobs exhibit very high intra-job predictability* [41, 53]. They perform repetitive iterations with constant behaviors and duration. Besides, the runtime speed of distributed training workloads with arbitrary GPUs can be accurately modeled and estimated [7]. Similar to [38, 41], we leverage some profiling techniques to estimate the completion time of a DLT job with any amount of GPU resources using at most two GPU nodes. We adopt a dynamic scaling mechanism to strike a balance between profiling latency and resource utilization.

Second, *DLT jobs naturally support the preemption scheduling with acceptable overhead compared to the execution time* [16, 34, 35, 43]. In contrast, there are no general mechanisms to preempt arbitrary big data jobs seamlessly and efficiently [5, 31, 40]. Inspired by this feature, we adopt a lease-based training scheme. A lease term refers to a fixed period of time that a DLT job can exclusively occupy the requested resources. Then the completion of a DLT job requires a number of lease terms. We design a *job selection* technique based on this scheme. It discretizes the execution time of a SLO job into multiple lease terms, and utilizes the MILP solver to decide when each lease term should be assigned. Compared to [39, 48] for big data scheduling, our adoption of the lease-based training with preemption increases the scheduling elasticity and enables the satisfaction of more SLO requirements. Besides, our method also supports the specification and fulfillment of soft deadline requirements in addition to strict ones, giving users more flexibility.

Third, *DLT jobs are more placement-sensitive* [34, 53]. The performance of a job highly depends on the affinity of the allocated GPU resources. Based on this feature, we propose a *GPU allocation* mechanism, to determine where each job should run after the lease term assignment. This strategy considers the impact of the GPU resource topology on our scheduling goals. We design a round-up method to ensure the selected SLO jobs run on consolidated resources, which can decrease the possibility of deadline violations. We also propose a local-search placement algorithm to discover an effective solution for the latency reduction of best-effort jobs.

To evaluate Chronus, we perform large-scale simulations on two real-world DL job traces (Philly-trace from a Microsoft cluster [21] and Helios-trace from a SenseTime cluster [19]). Experimental results show that Chronus can reduce the deadline miss rate by up to 14.7×. Compared to other deadline-aware solutions, Chronus reduces the latency of best-effort jobs by up to 19.9×. To further demonstrate the practicality of our design, we implement *Chronus* as a custom scheduler in Kubernetes [26], and deploy it on a cluster of 120 GPUs, running a wide range of common DL models (e.g., VGG16 [44], AlexNet [25], MobileNet [18], InceptionNet [46], ResNet [17], GAN [15], Bert [12], RL [24, 45]). Evaluations suggest that Chronus guarantees the deadline of SLO jobs and maintains the latency of best-effort jobs.

2 MOTIVATION

2.1 SLO Requirement in DL Training

Similar to conventional datacenters for big data jobs [9, 39, 48], a GPU cluster is also required to support the mixture of SLO jobs and best-effort jobs. SLO jobs are usually production and business-related. They are highly expected to be completed before certain deadlines, and the violations can incur huge financial loss. Best-effort jobs are mainly research-exploratory, which are more sensitive to job latency instead of SLO requirements.

To unveil the SLO requirement of DLT jobs, we conduct a user survey[1] collected from 103 participants (52% students, 16% researchers, and 32% engineers). According to this survey, we identify the following observations.

Observation 1: *SLO jobs and best-effort jobs coexist in modern GPU clusters.*

We observe that more than 60% participants mainly submit DLT jobs to their GPU clusters with explicit expectation of the job completion time. Additionally, 96% participants have the experience of submitting non-emergent/best-effort jobs for trial-and-error.

Observation 2: *Users can tolerate SLO violations of DLT jobs to certain extent.*

About 3%, 12%, 21% and 22% participants can accept 0%, 5%, 10% and 20% deadline delay respectively when the cluster supports the SLO guarantee for DLT jobs. This suggests the scheduler can manage the deadlines of SLO jobs at the granularity of minutes or even hours.

Observation 3: *It is not acceptable that best-effort jobs deprive SLO jobs of their GPU resources, and cause deadline violations.*

Over 32% participants cannot accept that best-effort jobs occupy the GPU resources originally reserved for SLO jobs. About 55% participants allow best-effort jobs to run simultaneously when they have no impacts on the completion time of the SLO jobs.

Observation 4: *Users have difficulties in estimating the execution time of their DLT jobs.*

Nearly 80% participants predict the job completion time with at least 10% error, and 40% participants have at least 25% prediction error. Notably, 5% participants suffer from 100% prediction error. As the duration of a DLT job can be up to several days, a small prediction error could result in long time deviations, significantly affecting the system operations. Also, the completion time of a DLT job varies under different scheduling algorithms and cluster states. Hence, it is infeasible for users to accurately estimate job completion time before scheduling.

2.2 Challenges of SLO Enforcement

Existing deadline-aware schedulers [9, 29, 39, 48] are not tailored to the characteristics of DLT jobs. There are still some unsolved problems when applying them to DLT job scheduling, as described below.

First, job completion time is prerequisite for deadline-aware scheduling. Some big data schedulers [10, 23, 39, 48] adopt offline methods (e.g., history trace, user specification, analytical model prediction) to obtain such information. This is not effective for DLT jobs because the runtime of DLT jobs is correlated with more factors, e.g., resource topology, model feature, batch size, iterations. History trace and user specifications fail to reflect the impact of these factors (Observation 4). Analytical models cannot estimate the runtime speed of models with unknown network structures or algorithms. Some other big data schedulers [11, 20] perform online profiling to estimate the job completion time. However, it requires to reserve large amounts of GPU resources for profiling, especially for large-size jobs. This can cause huge resource waste.

Second, job selection is a critical step in guaranteeing deadlines of SLO jobs and reducing the latency of best-effort jobs. Prior solutions [9, 29, 48] adopt the MILP solver to discover the best decision for a batch of SLO jobs. They can be enhanced from two perspectives. First, they do not consider the operation of job preemption, which can actually improve the possibilities of SLO satisfaction. However, frequent preemption can bring large overhead to the job performance. How to appropriately leverage this feature to improve the scheduling efficiency is not explored yet. Second, these solutions only consider strict deadlines for SLO jobs. According to our Observation 2, some users are tolerant with proper violations of deadlines. How to manage and enforce such "soft" deadlines for certain jobs is challenging.

Third, the runtime speed of a distributed DLT job highly depends on the topology of allocated GPUs. The job generally runs faster on consolidated GPUs due to the low cost of local communications. However, existing deadline-aware schedulers [9, 29, 39, 48] only consider the amount of available resources while ignoring their topology. Hence, if a job is justified by the MILP solver to meet the SLO requirement in the consolidated manner, it can still possibly miss the deadline when the placement is actually not consolidated. It is non-trivial to consider the impact of resource allocation on the scheduling decision.

3 SYSTEM DESIGN

We propose Chronus, a novel DL scheduler to enforce the deadlines of SLO jobs and reduce the latency of best-effort jobs. We give assumptions and overview in Sec 3.1, followed by the description of each component in Sec 3.2 – 3.4.

3.1 Overview

We make several assumptions about DLT jobs and GPU cluster in our system. (1) A DLT job is considered as completed when it finishes a fixed number of training iterations specified by the user. (2) Each GPU has enough memory to host the entire model of the DLT job. (3) Training with the model parallelism technique is not considered in Chronus (4) We focus on the *homogeneous* GPU resources and physical networking connections. Our design can be extended to heterogeneous clusters as well (discussed in Sec. 7).

[1] More details about the survey are available at https://github.com/S-Lab-System-Group/ChronusArtifact

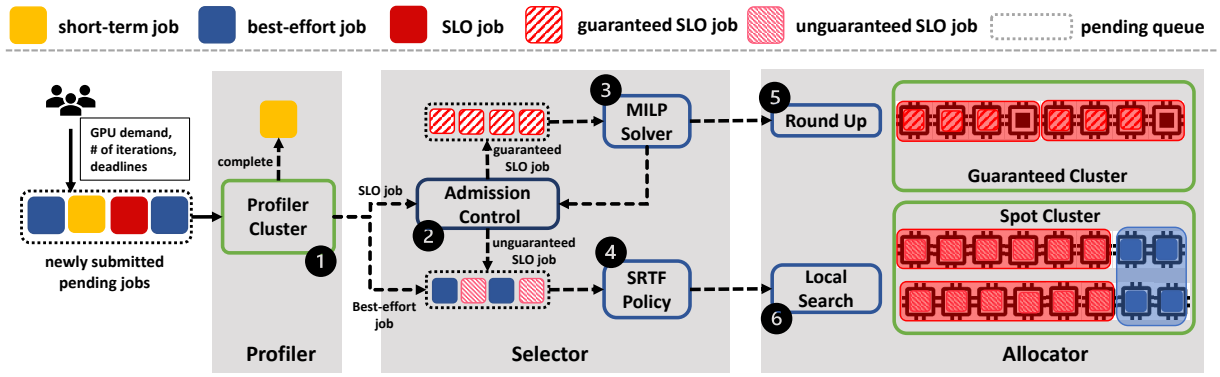


Figure 1: Workflow of Chronus.

Fig. 1 shows the workflow of Chronus. It consists of three main components. (1) A `Profiler` filters out short-term or buggy DLT jobs, and estimates the completion time of long-term jobs submitted by cluster users (❶). (2) After the profiling is completed, a `Selector` performs admission control to check and label each SLO job as guaranteed (the user-specified deadline is achievable), or unguaranteed (the deadline is hard to be satisfied) (❷). For guaranteed SLO jobs, an MILP solver is utilized to identify the jobs that need to be scheduled (❸). For unguaranteed SLO jobs and best-effort jobs, the Shortest-Remaining-Time-First (SRTF) algorithm is used to select jobs (❹). (3) An `Allocator` distributes GPUs to the selected jobs. For the guaranteed SLO jobs, it adopts a round-up technique (❺) to discover a consolidation solution for GPU application. For other jobs selected by SRTF, it performs a local search algorithm to identify an effective placement solution (❻).

Chronus can be deployed in existing GPU clusters. It logically partitions the cluster into three parts: a Profiling cluster is used by the `Profiler` to collect runtime information of DLT jobs; a Guaranteed cluster hosts the guaranteed SLO jobs and enforces their deadlines; a Spot cluster improves the latency of best-effort jobs and unguaranteed SLO jobs in an opportunistic manner. The capacities of these clusters are dynamically tuned based on the job density.

3.2 Profiler

Users submit DLT jobs to the cluster with relevant information (GPU demands, deadlines, numbers of training iterations). Then the `Profiler` runs these jobs in the Profiling cluster for a fixed time $T_{profile}$. It adopts the First Come First Serve (FCFS) policy. Short-term jobs and buggy jobs can be completed within $T_{profile}$ without the need for scheduling. For long-term jobs, the `Profiler` obtains the duration of one iteration, and then estimates the total completion time.

Such a profiling mechanism is also adopted in prior DL schedulers [16, 34]. However, some critical problems remain unsolved, e.g., how many resources should be allocated for profiling, how to handle the shortage of GPU resources. We leverage several approaches to address these problems and achieve more efficient profiling.

3.2.1 Submission control. It is common that users may submit a large number of DLT jobs concurrently or within a very short time (Fig 5(b)). This can impose heavy burdens to the Profiling cluster, causing much longer queuing delays for pending jobs. Previous works never consider such scenarios. To handle the issue of bursty job submission, our `Profiler` adopts a submission control policy to restrict the maximum amount of resources requested by each user with a fixed time interval. In Sec 5.1, we will show this mechanism can remarkably reduce the time cost of profiling DLT jobs.

3.2.2 Reducing profiling resources. Previous schedulers [16, 34] profile DLT jobs using the same amount of requested GPUs, which requires a large size of Profiling cluster to handle large-scale DLT jobs. This will decrease the size of the main cluster, and the overall resource utilization. To overcome this limitation, we propose to convert every multi-node distributed job into a two-node and single-node jobs. Then we can use at most two nodes to profile each job with arbitrary GPU demands and estimate its runtime speed [55].

Specifically, a DLT job is composed of many iterations. Fig. 2 shows the average duration with the standard deviation of one iteration for different DL models and GPU demands. We observe the runtime speed of the same DLT task is relatively stable. Hence, we only need to measure the duration for a small number of iterations, and then estimate the overall execution time based on the number of training iterations provided by the user. According to the analytic model in [7], for a distributed DLT job with n GPU nodes, the execution time of each iteration can be formulated as $t_n = t_c + log_2(n)t_o$, where t_c is the computation time and t_o is the communication time. Hence, we can measure the iteration time of the job on one node as $t_1 = t_c$, and on two nodes as $t_2 = t_c + t_o$. From these two results, we can derive the

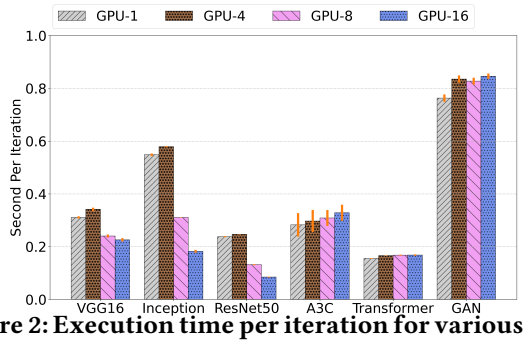


Figure 2: Execution time per iteration for various models and requested GPUs.

iteration time on n nodes: $t_n = t_1 + log_2(n)(t_2 - t_1)$ without actually running it.

3.2.3 Dynamic scaling of Profiling cluster capacity. Past works adopted fixed sizes of the Profiling cluster. However, a smaller cluster can lead to the increased pending time of submitted jobs, while a larger profiling cluster can decrease the size of the main cluster, and affect the performance of long-term jobs. To balance the trade-off between stability and resource utilization, we propose to dynamically adjusts the capacity of the Profiling cluster based on the density and requested resources of submitted DLT jobs. Specifically, we model the Profiler cluster as a queuing system, where the arrival rate of submitted jobs is λ. To guarantee the stability of this queuing system, we need to ensure the cluster capacity $C_{profile} \geq \lambda T_{profile}$. In our design, $C_{profile}$ is updated every hour to adapt to the job submission trend. For each update, we collect the submitted jobs over the past one hour and calculate the average GPU numbers to estimate λ. Then we can identify the lower bound of $C_{profile}$.

3.3 Job Selector

The primary function of the `Selector` is to produce resource-time scheduling decisions for SLO jobs and best-effort jobs. It adopts a lease-based training scheme, and uses an MILP solver and SRTF policy to select jobs.

3.3.1 Formulation of SLO requirements. Previous deadline-aware schedulers [9, 29, 39, 48] only consider the strict deadline scenario, i.e., the job must be completed before the specific moment. Based on our Observation 2 in Sec. 2.1, it is necessary to enable the soft deadlines, so the DLT jobs are allowed to complete after the deadlines with certain penalty.

We introduce a unified reward function to formulate different types of requirements (strict SLO, soft SLO and best-effort). Users can specify such functions to the GPU cluster when submitting the jobs. The reward is modeled as a step function (ranging between 1 and 100), as shown in Fig. 3. For best-effort jobs, we expect them to be completed as soon as possible without any deadlines. So the reward value is always constant (= 1) regardless of the completion time. For

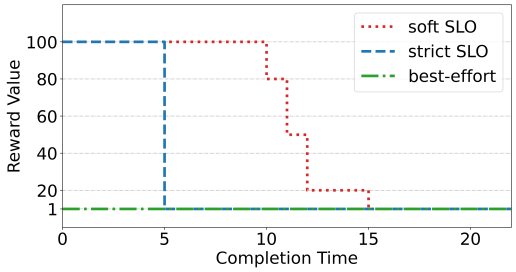


Figure 3: Reward functions for different types of jobs.

strict SLO jobs, they must be completed before the deadlines (= 100). Otherwise, the reward drops to the smallest value (= 1) immediately. For soft SLO jobs, the reward drops gradually with longer delays of completion time[2].

3.3.2 Admission control. It is possible that some users specify unreasonable deadline requirements for their jobs such that the cluster can never satisfy them. Malicious users may also intentionally abuse the SLO service to affect the scheduling operation and other jobs' performance. We introduce an admission control mechanism to handle these cases. It is triggered right after the profiling phase. For all the profiled SLO jobs, the admission control calls the MILP solver in Sec. 3.3.4 to check if there are any solutions to meet the SLO requirements. A job with a feasible solution will be labeled as a guaranteed SLO job, and placed in the SLO queue. Otherwise, it will be labeled as an unguaranteed SLO job, and placed in the best-effort queue mixed with the best-effort jobs[3]. Meanwhile, the user will receive the notification that the deadline cannot be satisfied, and the job will be treated in a best-effort manner.

3.3.3 Lease-based training. We divide each DLT job into multiple time periods (i.e., lease terms) with the same length. A job can be executed only when it obtains a lease term from the scheduler. A renewal is required when the lease expires. Upon a successful renewal, the job can continue its execution. Otherwise, it will be suspended and yield the resources.

We introduce two types of leases for the `Selector`: the SLO lease is adopted in the Guaranteed cluster for guaranteed SLO jobs; the BE lease is used in the Spot cluster for unguaranteed SLO jobs and best-effort jobs. The `Selector` distributes SLO and BE leases to the corresponding jobs at each scheduling cycle, when the leases expire. For ease of management, we set the SLO lease length as an integral multiple of the BE lease length. So BE lease expiration does not necessarily lead to the suspension of jobs in SLO lease terms,

[2]The reward function for soft SLO jobs can have other expressions. We can always approximate any function to the step function in our scheduler.
[3]It is possible to use binary search to obtain reasonable completion time for these jobs and then label them as guaranteed SLO jobs. However, this will incur a large overhead for calling the MILP solver multiple times. Hence, it is not adopted in our design.

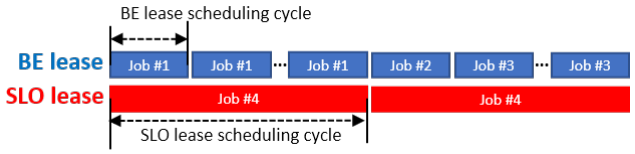


Figure 4: Illustration of two types of lease terms.

while SLO lease expiration happens concurrently with BE lease expiration. Fig. 4 shows these two types of leases.

3.3.4 Selecting guaranteed SLO jobs. We model the job selection task as an MILP problem. Then the MILP solver can be used to perform admission control (Sec. 3.3.2) and manage SLO lease terms for all guaranteed SLO jobs. At each SLO scheduling cycle, the `Selector` aggregates all guaranteed SLO jobs and makes decisions to update job status and assign cluster resources.

Formally, we consider a set of N guaranteed SLO jobs: $\mathcal{J} = \{j_1, j_2, j_3 ... j_N\}$ and M available GPUs at one scheduling cycle. Each job j_i requires G_i GPUs, with the runtime R_i estimated by the `Profiler`. We denote the reward function of the job j_i as a step function $f(\langle D_{1,i}, V_{1,i}\rangle, ..., \langle D_{P_i,i}, V_{P_i,i}\rangle)$, where the reward value is $V_{p,i}$ when the job is completed right before $D_{p,i}$. Note the strict SLO is a special case of soft SLO, where the step function has only two possible reward values. Assuming the SLO lease length is T_l, then the number of required lease terms to finish job j_i is $RL_i = \lceil R_i/T_l \rceil$. The number of required lease terms to complete this job before each deadline is $DL_{p,i} = \lfloor D_{p,i}/T_l \rfloor$, where $p \in \{1, 2, ...P_i\}$.

We use a binary variable $x_{k,i}$ to denote whether j_i obtains the k_{th} lease, and a binary variable $s_{p,i}$ to denote whether j_i hits the corresponding p_{th} deadline. The MILP solver can yield a solution for the following objective and constraints.

$$\max \sum_{i=1}^{N} \sum_{p=1}^{P_i} s_{p,i} * V_{p,i} \tag{1}$$

subject to:

$$x_k^i, s_{p,i} \in \{0, 1\}, \forall p, k, \forall i \in [1, N] \tag{2}$$

$$\sum_{i=1}^{N} x_k^i * R_i \leq M, \forall k \tag{3}$$

$$\sum_{k=1}^{DL_{p,i}} s_{p,i} * x_k^i = s_{p,i} * RL_i, \forall i \in [1, N] \tag{4}$$

$$\sum_{p=1}^{P_i} s_{p,i} = 1, \forall i \in [1, N] \tag{5}$$

Objective (1) is to maximize the total reward values of all guaranteed SLO jobs. Constraint (3) ensures the number of occupied GPUs does not exceed the cluster capacity. Constraint (4) ensures that all guaranteed SLO jobs should be completed before (soft) deadlines. Constraint (5) ensures each guaranteed SLO job is assigned with one feasible solution to meet the (soft) deadline.

Based on the solution from the solver, the `Selector` identifies the SLO jobs that need to be scheduled at this cycle. Meanwhile, it also identifies the necessary GPU nodes to host these selected jobs, which form a Guaranteed cluster. The rest jobs will be placed in a SLO queue for consideration at the next SLO scheduling cycle. The length of an SLO lease is of vital importance to the efficiency of the MILP solver. A shorter SLO lease introduces too frequent preemption operations and longer MILP solver latency, while a longer SLO lease can reduce the scheduling elasticity. We empirically set the length of a SLO lease term as 20 minutes.

The MILP solver introduces certain latency when solving the above problem, which can delay the job execution. To minimize the impact of such delays, we cache the solution of the last scheduling cycle. If the MILP solver fails to get a new solution for this cycle within a fixed time limit, we call the MILP solver from the cached solution to reduce the search space and computation overhead.

3.3.5 Selecting best-effort and unguaranteed SLO jobs. The `Selector` adopts the Shortest-Remaining-Time-First (SRTF) algorithm to select jobs from the best-effort queue and allocate BE leases to them for execution in the Spot cluster. We empirically set the length of a BE lease term the same as $T_{profile} = 300s$, which is one fourth of a SLO lease term.

The Spot cluster is built from the remaining resources after the establishment of the Profiling cluster and Guaranteed cluster. It is worth noting that when a guaranteed SLO job completes its execution, there is still a amount of time left in the current SLO lease. To avoid the resource waste, the resources yielded from this completed SLO job will be adjusted to the Spot cluster, and considered at the next BE scheduling cycle.

3.4 Allocator

The `Allocator` is responsible for allocating resources to the jobs identified from the `Selector`. To improve the job performance, an optimal strategy always follows the consolidation principle, i.e., deploying the job on as few nodes as possible. We utilize this strategy with a round-up solution to allocate resources for guaranteed SLO jobs (Sec. 3.4.1).

However, when more jobs are deployed in the cluster, there will be more GPU fragmentation, making it harder to achieve consolidation for newly submitted jobs. To deal with this issue, we propose a local search algorithm to place jobs in the best-effort queue (Sec. 3.4.2).

3.4.1 Placing guaranteed SLO jobs. Chronus performs placement for a batch of SLO jobs at the end of each SLO scheduling cycle. Consider a homogeneous GPU cluster with 8-GPU

compute nodes. We denote 1-GPU, 2-GPU, 4-GPU and $8n$-GPU jobs ($n \in N^+$) are consolidation-friendly, and jobs with other numbers of GPUs are consolidation-hostile. We say a placement solution has the consolidation feature, if each job with n_i GPUs is deployed on $\lceil n_i/8 \rceil$ nodes. Then we have the following proposition:

Proposition 1. *Assume the GPU cluster has m free nodes and each node has exact 8 GPUs. The pending queue only contains consolidation-friendly jobs. The total number of requested GPUs from all these jobs is no greater than 8m. Then there must exist a feasible solution that achieves consolidation placement.*

Proof. We construct a solution to fulfill the requirement. We first allocate the GPU resources to $8n$-GPU jobs in a consolidation manner such that there is no GPU fragmentation on the allocated nodes. Then we split the remaining m' nodes into $2m'$ 4-GPU nodes, and use some nodes to satisfy the consolidation placement of 4-GPU jobs. Next, we split the remaining m'' 4-GPU nodes into $2m''$ 2-GPU nodes to host 2-GPU jobs with consolidation. Finally, the remaining resources can be allocated to 1-GPU jobs. □

When all the guaranteed SLO jobs are consolidation-friendly, we can find a consolidation placement solution from Proposition 1. Due to the existence of consolidation-hostile jobs, it is possible that there are enough resources for all the jobs but a consolidation solution does not exist. To handle this case, we convert each consolidation-hostile job to a consolidation-friendly one by rounding up the number of its requested GPUs to $\{1, 2, 4, 8n\}$. Hence, Constraint 3 can be changed to:

$$\sum_{i=1}^{N} x_k^i * \text{RoundUp}(R_i) \leq M, \forall k \tag{6}$$

Although this round-up operation may increase the total of demanded resources slightly (consolidation-hostile jobs are not common), it ensures the existence of a consolidation solution at each SLO scheduling cycle, and significantly improve the allocation efficiency. With the solution from Proposition 1, each guaranteed SLO job can run at very fast speed as in the Profiling cluster. Note that this round-up technique is general to other cluster configurations: a node with an arbitrary number of GPUs can always be decomposed into some 2^n-GPU nodes (e.g., 6 = 4 + 2), and then this technique can be applied.

3.4.2 Placing best-effort and unguaranteed SLO jobs. For large jobs in the best-effort queue which request 16 or more GPUs, the `Allocator` also uses the round-up-based consolidation placement (Sec. 3.4.1). For small jobs in the best-effort queue, we design a novel local search algorithm to reduce their latency. It can effectively handle the placement of consolidation-hostile jobs.

Consider a job j_i with G_i GPUs. Its placement solution set is denoted as $\mathbb{A}_i$, containing all the possible solutions. A_i^* is denoted as the optimal solution that meets the consolidation requirement. We use $RTS(j_i, A_i)$ to denote the runtime speed of j_i under a solution $A_i \in \mathbb{A}_i$. Then we define an allocation reward function RW (Eq. 7) to quantify the correlation between job performance and placement topology. A higher $RW(j_i, A_i)$ indicates job j_i runs faster under the solution A_i.

$$RW(j_i, A_i) = G_i * \frac{RTS(j_i, A_i)}{RTS(j_i, A_i^*)} \tag{7}$$

We also define the potential of a job j_i, to denote how much it prefers the consolidation solution:

$$pot(j_i) = \max_{A_i \in \mathbb{A}_i} RW(j_i, A_i) - \min_{A_i \in \mathbb{A}_i} RW(j_i, A_i) \tag{8}$$

Given a set $\mathcal{J}$ of jobs, we exhaustively search for the optimal placement solution of K jobs with higher potential, which are more placement-sensitive. Then we place the rest jobs in a quasi-consolidation manner. Alg. 1 describes the search process. Specifically, we profile different placement solutions of each job and calculate the corresponding potential. Then we sort the jobs by their potential (Line 1). We select top-K jobs J^s to ensure the entire search space of these K jobs is smaller than a predefined threshold $|S|$ (Line 2). Next, we consider all possible combined placement solutions $\mathbb{A}^s$ of these K jobs (Line 6). For each $A^s \in \mathbb{A}^s$, we allocate the rest jobs J^q with a quasi-consolidation solution A^q (Line 8): we first try to find a feasible consolidation solution for each job. If no solution exists, we will allocate this job to as few nodes as possible. Finally we compute the sum of the RW values of all the jobs under $A = A^s U A^q$ (Line 10). The optimal solution $\mathcal{A}_\mathcal{J}$ is the one with the largest RW value (Lines 11-12).

Algorithm 1: Local Search Placement Strategy.

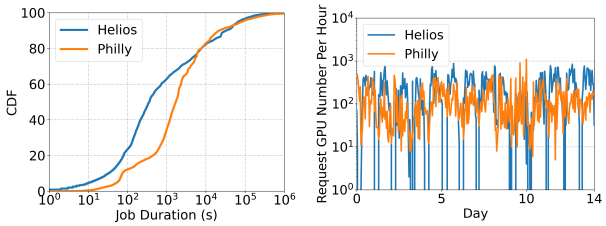
Input : Job set $\mathcal{J}$, job potential P,
allocation set $\mathcal{A}$, search space $\mathcal{S}$
Output: Optimal solution set $\mathcal{A}_\mathcal{J}$
Sort jobs in $\mathcal{J}$ in descending order by their $pot(\cdot)$;
$K \leftarrow \arg\max_k (\prod_{i=1}^{k} |\mathbb{A}_i| \leq |S|)$;
$J^s \leftarrow j_1, j_2, ...j_K$;
$J^q \leftarrow j_{K+1}, j_{K+2}, ...j_N$;
$\mathcal{R}* \leftarrow 0, \mathcal{A}_\mathcal{J} \leftarrow \{\}$;
$\mathbb{A}^s \leftarrow \{(j_1, A_1), (j_2, A_2), ..., (j_K, A_K)\} | A_1 \in \mathbb{A}_1, A_2 \in \mathbb{A}_2, ..., A_K \in \mathbb{A}_K\}$
for $A^s \in \mathbb{A}^s$ **do**
  $A^q \leftarrow$ Quasi-Consolidation(J^q);
  $A \leftarrow A^s \bigcup A^q$;
  $R \leftarrow \sum_{(j_i, A_i) \in A} RW(j_i, A_i)$;
  if $R \geq \mathcal{R}*$ **then**
    $\mathcal{R}*, \mathcal{A}_\mathcal{J} \leftarrow R, A$;
return $\mathcal{A}_\mathcal{J}$

4 EXPERIMENTAL SETUP

We implement a trace-driven simulator to simulate GPU clusters with different schedulers. It has 10,865 lines of python code[4]. We implement Chronus in our simulator with 2,431 lines of python code. It adopts Gurobi 9.1 [8] as the backend MILP solver. We also implement Chronus on the real Kubernetes scheduling system, as detailed in Sec. 5.5.

4.1 Testbed

We simulate two homogeneous GPU clusters: a 120-node cluster (C120) and 96-node cluster (C96). Each node contains 8 GPUs. We adopt two real-world DLT job traces for simulation. The Helios trace is from a production cluster in SenseTime [19]. It is collected over two weeks from 14th - 27th April 2020. The Philly trace is from Microsoft datacenter [21]. We select a 14-day trace from 12th - 25th October 2017. Fig. 5(a) shows the cumulative distributions of the job duration in the two traces. We observe that Helios has a higher ratio of short-term jobs than Philly. Fig. 5(b) shows the amount of requested GPUs per hour in the two traces. They exhibit an obvious bursty submission feature.



(a) Duration distribution (b) Submission rate

Figure 5: Trace characterization.

For each job, our traces contain the information of submit time, duration, GPU demand, user name, job type (Strict SLO/Soft SLO/BE), and model type (VGG, ResNet, etc). We obtain the runtime speed of jobs with different GPU topologies, and their preemption overhead via running the corresponding type of model on actual GPUs. Since the jobs in these two traces do not have explicit deadline information, we adopt the following method to generate deadlines for SLO jobs. Inspired by [23, 39], for a strict SLO job with the duration of R_i, we choose a random value in $[1.2R_i, 2R_i]$ as its deadline. For a soft SLO job, we follow the same method to generate the first deadline $D_{1,i}$. Based on the user survey, we specify another three soft deadlines as $1.1D_{1,i}$, $1.2D_{1,i}$, $1.5D_{1,i}$ with reward values of 80, 50, 20, respectively.

We synthesize seven workloads from the two traces, as summarized in Table 1. First, we filter out the short-term jobs which can be completed in the Profiling cluster and are never scheduled in the main cluster (Short). Second, we consider the workloads with all strict SLO jobs (H_SLO and P_SLO). Third, we construct workloads mixed with strict SLO and best-effort jobs (H_MIX1 and P_MIX1). Finally, we build workloads with strict SLO, soft SLO and best-effort jobs (H_MIX2 and P_MIX2).

[4]The implementation code is available at https://github.com/S-Lab-System-Group/ChronusArtifact

Table 1: Summary of workloads in our experiments

Workload	Strict/Soft/BE (%)	Trace	Cluster	# of jobs
Short	0 / 0 / 100	Helios	–	5,735
H_SLO	100 / 0 / 0	Helios	C96	6,599
H_MIX1	70 / 0 / 30	Helios	C96	6,599
H_MIX2	30 / 60 / 10	Helios	C96	6,599
P_SLO	100 / 0 / 0	Philly	C120	30,940
P_MIX1	70 / 0 / 30	Philly	C120	30,940
P_MIX2	30 / 60 / 10	Philly	C120	30,940

4.2 Evaluation Metrics.

We use the following metrics to quantify the performance and efficiency of DLT scheduling.

Weighted Deadline Miss Rate (wDMR). This is a standard metric to measure the enforcement of SLO requirements by a scheduler. Consider a set J_{slo} of SLO jobs and each job j_i obtains a reward value $RW(j_i)$ based on its SLO requirement (Fig. 3). Then wDMR is defined in Eq. 9, where $RW_{min} = 1$ and $RW_{max} = 100$ are the bounds of the reward values.

$$wDMR = \frac{1}{J_{slo}} \sum_{j_i \in J_{slo}} \frac{RW(j_i) - RW_{min}}{RW_{max} - RW_{min}} \tag{9}$$

Job Completion Time (JCT). We measure the average completion time of best-effort jobs for their performance. A small JCT indicates the cluster has higher efficiency.

4.3 Baselines

To fully demonstrate the advantages of Chronus, we select six popular scheduling systems from prior works for comparisons. They can be classified into three categories.

General scheduler: (1) Yarn-CS [49] adopts the static quota and FCFS algorithm to manage jobs and resources. In our implementation, we configure the static quota according to the ratio between SLO and best-effort jobs.

Deadline-aware scheduler: (2) The Earliest-Deadline-First (EDF) algorithm [3] is a representative solution to maintain SLO requirements in real-time systems. In our implementation, SLO jobs are allowed to preempt best-effort jobs. To prevent the abuse of the SLO service, we disable the preemption of SLO jobs. (3) 3Sigma [39] leverages the MILP solver to schedule SLO and best-effort jobs in big data clusters. It is not effective in supporting the preemption between SLO jobs, which significantly restricts the search space of MILP. Considering the time scale of DLT jobs in our traces, we set the scheduling cycle as 60 seconds. (4) GENIE [7] proposes an offline prediction model to estimate the processing

rate and response latency for diverse DL workloads. It enables DLT jobs to run on various GPU resources in an elastic manner and identifies the best placement for DLT jobs. It prioritizes SLO jobs with the smallest laxity without considering best-effort jobs. We assign best-effort jobs with the lowest priority.

Deep Learning scheduler: (5) Tiresias [16] measures the GPU time of the received jobs, and adopts the Least Attained Service and Gittins Index algorithm to increase the job throughput and decrease the average job completion time. We implement it following the same system setting in its released code. (6) Themis [34] proposes the finish time fairness as a new metric to evaluate scheduling fairness. We follow the same implementation in [38].

5 EVALUATION

We first validate the design of each system component and identify the optimal parameters (Sec. 5.1 – 5.3). Then we study the performance of the end-to-end system and comparisons with prior solutions (Sec. 5.4). Finally we present our prototyping results on real systems (Sec. 5.5).

5.1 Profiler Evaluation

We mix the jobs from the Short and H_MIX2 workloads, and deploy them on C96 to evaluate the `Profiler`.

First, we consider the impact of the Profiling cluster capacity. Fig. 6(a) shows the profiling pending overhead (red line) and wDMR (blue bars) of these jobs under different fixed sizes of the Profiling cluster. We also show the results when the dynamic scaling mechanism (Sec. 3.2.3) is adopted. We observe that (1) if the cluster is too large, the wDMR of SLO jobs will be increased, as the resources of the guaranteed cluster is reduced. (2) If the cluster is too small, there are not enough resources for profiling, and the profiling pending overhead is increased. (3) The dynamic scaling mechanism can perfectly balance such trade-off, giving the satisfactory pending overhead (32 seconds) and lowest wDMR (5.0%).

Second, we evaluate our submission control mechanism (Sec. 3.2.1). It limits the number of requested GPUs per user below 24 within each interval $T_{profile}$. Fig. 6(b) shows the distributions of the profiling pending time without and with submission control respectively. We observe this mechanism can effectively reduce the longest pending time from 2,105 seconds to 960 seconds. It enables the `Profiler` to respond to the jobs promptly.

Third, our `Profiler` can effectively filter and complete short-term jobs without scheduling them. Fig. 6(c) shows the JCT of these jobs in the Profiling cluster with different sizes as well as dynamic scaling. For comparisons, we also show the ideal result when the SRTF algorithm is applied (red dashed line)[5]. We see the average JCT is smaller with more profiling resources. With dynamic scaling, the JCT is slightly higher than the ideal one, which is still acceptable. This concludes the `Profiler` can significantly benefit the short-term jobs.

Fourth, the prediction accuracy of the `Profiler` can affect the scheduling performance. We perform a sensitivity analysis by perturbing the profiled job runtime with random Gaussian noise. Fig. 6(d) shows the scheduling result for different workloads, where x-axis is the standard deviation of the added noise and y-axis is the wDMR of SLO jobs. We observe Chronus exhibits strong robustness when the noise scale is smaller than 40%. This can be easily achieved by the `Profiler` in practical scenarios.

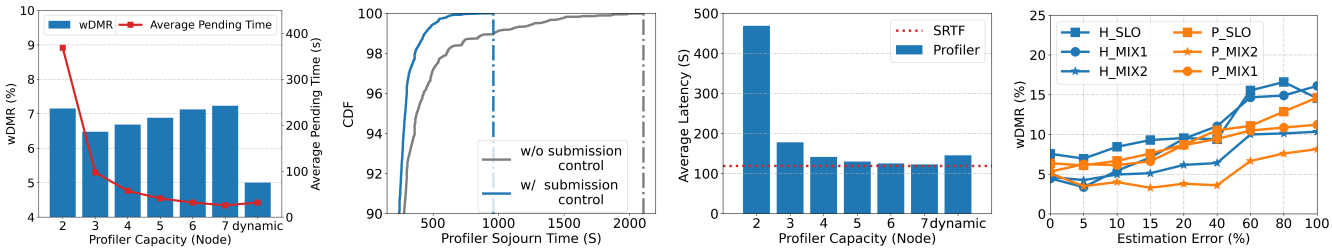
5.2 Job Selector Evaluation

We first measure the impact of the SLO lease length on the deadline enforcement. We run the H_MIX2 workload, and measure the JCT of best-effort jobs and wDMR of SLO jobs, as shown in Fig. 7(a). When the lease term is too short (<10 minutes), there will be more frequent preemption operations with large overhead, causing high wDMR for SLO jobs. When the lease term is too long, wDMR is also increased due to the restricted scheduling opportunities. A lease term between 15 – 30 minutes gives the best results. Besides, the SLO lease length does not affect the performance of best-effort jobs when it is longer than 15 minutes. In the following experiments, we will set the SLO lease length as 20 minutes.

Second, the MILP solver can effectively support the soft deadlines of SLO jobs by maximizing the total reward value (Eq. 1) and rejecting unguaranteed SLO jobs. Fig. 7(b) shows the wDMR of the SLO jobs in two cases: (1) the MILP solver tries to maximize the objective under the constraints. (2) The MILP solver only finds a feasible solution to obey the constraints. We observe the consideration of the objective can significantly reduce the wDMR, mainly for the improvement of soft SLO jobs. Without this objective, the wDMR of soft SLO jobs will be terribly affected. Fig. 7(c) shows the wDMR of the unguaranteed SLO jobs rejected by the admission control mechanism in various workloads. We observe that wDMR is very high for these jobs, indicating this mechanism can effectively identify the unguaranteed jobs which cannot be completed before the deadlines.

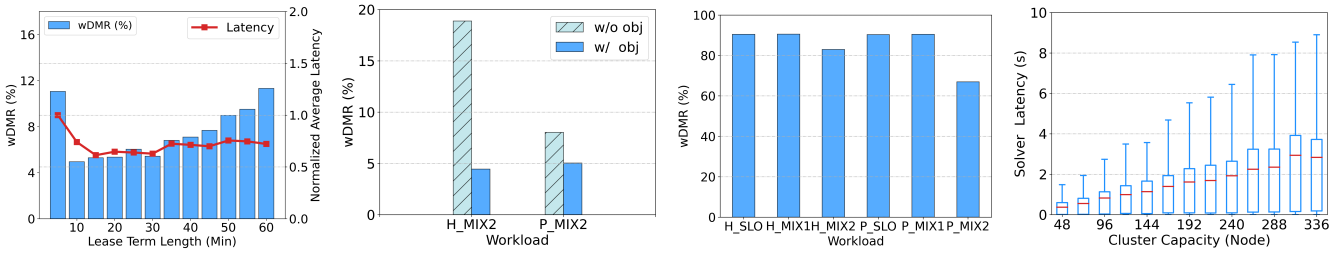
Third, the latency of the MILP solver can affect the scalability of Chronus. For a larger cluster with a higher job submission rate, the MILP solver needs to take more time to identify the solutions, which might decrease the scheduling efficiency and incur larger pending overhead. We adopt the

[5]This ideal result cannot be achieved in practice as the remaining time is unknown during profiling.



(a) Impact of Profiling cluster size (b) Impact of submission control (c) JCT of short-term jobs (d) Sensitivity analysis

Figure 6: Performance of the `Profiler`.



(a) Impact of the SLO lease length (b) Impact of the objective (c) Impact of admission control (d) Scabaility analysis

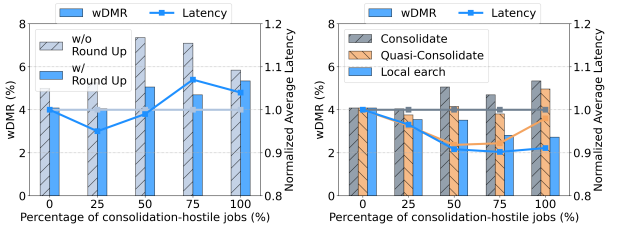
Figure 7: Performance of the `Selector`.

H_MIX2 workload, and adjust the number of jobs proportional to the cluster capacity. Fig. 7(d) shows the corresponding solver latency under different cluster and job scales. The maximal latency from the MILP solver is less than 10 seconds, which can be ignored compared to the duration of DLT jobs. This suggests that Chronus can handle large-scale GPU clusters and heavy workloads with high efficiency.

5.3 Allocator Evaluation

The `Allocator` adopts two placement strategies for different types of jobs. They are mainly optimizing the consolidation-hostile jobs. To show the impact of these jobs on the placement strategies, we modify the GPU demands of some jobs in H_MIX2 to get different ratios of consolidation-hostile jobs. We set the threshold $|S|$ as 100,000 in the implementation.

First, we check the consolidation placement for guaranteed SLO jobs (Sec. 3.4.1). Fig. 8(a) shows the average JCT of best-effort jobs (lines) and wDMR of SLO jobs (bars), without and with the round-up technique respectively. We get two observations. (1) The round-up technique can effectively reduce the wDMR of SLO jobs. Without it, the `Allocator` cannot find a consolidation solution for some guaranteed SLO jobs, and has to put them in the pending state, which can cause deadline violations. When round-up is applied (Eq. 6), the MILP solver will request more compute nodes to satisfy the consolidation principle for the selected SLO jobs. Then the wDMR becomes smaller. (2) When the ratio of consolidation-hostile jobs is higher, the benefit of round-up is smaller. This is because more consolidation-hostile jobs can cause more GPU fragmentation in the Guaranteed cluster. This lowers the resource utilization and leads more jobs to miss the deadlines. Round-up cannot mitigate this issue.



(a) Round-up technique (b) Local search

Figure 8: Performance of the `Allocator`.

Next, we explore the effectiveness of our proposed local search algorithm (Sec. 3.4.2). Fig. 8(b) shows the wDMR of SLO jobs (bars), and the average JCT of best-effort jobs[6] (lines). We consider three placement strategies for the best-effort jobs: consolidation, quasi-consolidation, and local search. We observe that local search beats the other two strategies in reducing the JCT of best-effort jobs. More interestingly, we find local search also reduces the wDMR of SLO jobs, even it is used for placing jobs in the best-effort queue. The reason is that round-up increases the GPU demands of consolidation-hostile jobs. Hence, some of these jobs will be judged by the admission control as unguaranteed SLO jobs, even they can meet the deadlines with the actual GPUs without rounding

[6]Normalized to the value under the consolidation policy

up. They will be placed in the best-effort queue. With the advanced local search algorithm, these jobs satisfy the deadline requirements in the Spot cluster, thus reducing the wDMR.

5.4 End-to-end Evaluation

SLO Enforcement. Fig. 9(a) shows the wDMR of our system for the six workloads, and comparisons with other baseline schedulers (Sec. 4.3). Chronus always gives the best results. In contrast, DL schedulers, especially Yarn-CS, are really poor in guaranteeing the deadlines, as they do not consider SLO in their design.

Deadline-aware schedulers perform better than DL schedulers. (1) For SLO workloads, GENIE outperforms 3Sigma and EDF. However, they are still not as good as Chronus, which utilizes the preemption feature. (2) For MIX1 workloads, EDF and 3Sigma achieve comparable performance as Chronus, as they can sacrifice best-effort jobs to free more GPUs for SLO jobs. (3) For MIX2 workloads, these deadline-aware schedulers only consider the strict deadlines while missing the opportunities of achieving better overall rewards. In contrast, Chronus leverages soft deadlines to significantly decrease the wDMR of SLO jobs.

Best-effort job performance. Fig. 9(b) shows the average JCT of best-effort jobs, normalized to the value of Chronus. We observe that Chronus still gives the best performance compared to other six schedulers. It can beat DL schedulers because Chronus can have enough GPU resources to reduce the latency of best-effort jobs without compromising SLO enforcement. Deadline-aware schedulers perform rather bad for best-effort jobs, as they seriously sacrifice them to satisfy the requirements of more SLO jobs.

Impact of the job density. Next, we measure the performance of different scheduling systems with different job densities in the cluster. We choose the H_MIX2 and P_MIX2 workloads. We scale down the job density to 80% by randomly removing 20% DLT jobs. We also scale up the job densities to 120%, 140% and 160% by randomly selecting certain numbers of jobs and inject them into the workloads [52]. Figs. 9(c) and 9(d) show the results of SLO enforcement in the two workloads. We observe a higher job density can increase the wDMR of all the schedulers. However, Chronus always performs the best given a fixed density.

Figs. 9(e) and 9(f) demonstrate the average JCT of best-effort jobs, normalized to that of Chronus. Similarly, Chronus gives the lowest JCT given one job density and workload. Compared to other deadline-aware schedulers, Chronus frees enough GPU resources for best-effort jobs without sacrificing the enforcement of SLO jobs. Compared to other DL schedulers, Chronus benefits from the runtime information during profiling to better schedule the best-effort jobs.

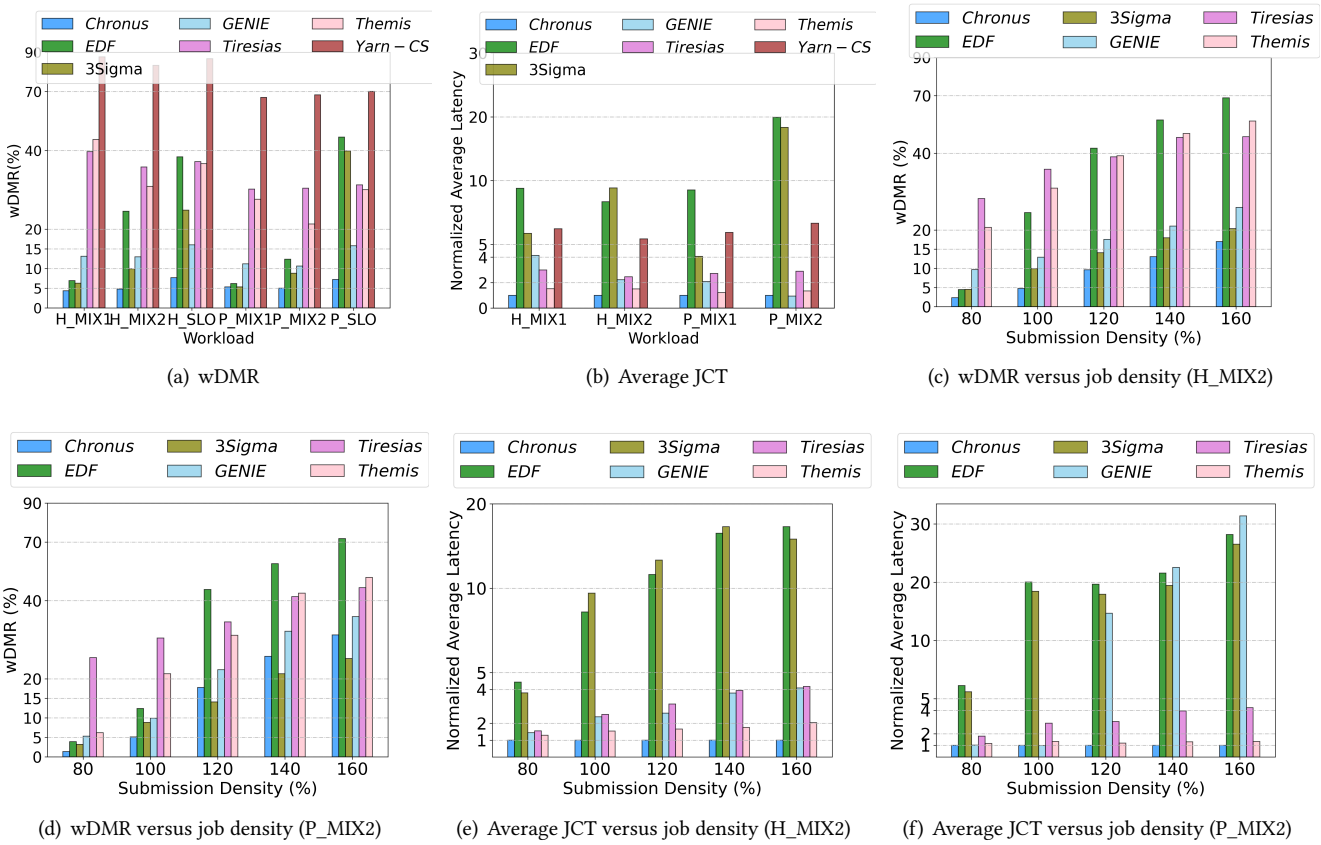
5.5 Prototype Implementation and Evaluation

To comprehensively validate the practicability of our design, we implement a prototype of Chronus on top of the Kubernetes [2]. Our implementation consists of a scheduler, controller and client-side watcher. (1) The client-side watcher is responsible for monitoring the execution progress of DLT jobs, receiving notifications of the checkpoint from the controller, and making checkpoints when the lease expires. (2) The controller notifies the scheduler when the lease of a DLT job expires and triggers job checkpoint by communicating with the watcher. It communicates with the MILP solver, an open-source goop library [27], to make scheduling decisions. (3) The scheduler receives the scheduling-related events and information from the controller (e.g., lease renewal, estimated remaining time), and manages the jobs (e.g., execution, termination, preemption, assigning resources). The implementation of the scheduler and controller contains a total of 4,293 lines of Go code. The client-side watcher only covers hundreds of lines of python code.

Our Chronus prototype can successfully schedule and host general DLT jobs and models. To compare the results from the trace-driven simulations and Kubernetes prototype, we sample some DLT jobs from H_MIX2, and randomly assign common DL models (VGG16, AlexNet, MobileNet, InceptionNet, ResNet, GAN, Bert, RL) to them. We restrict the number of requested GPUs in each job below 16, and set the duration of these jobs between 5 minutes to 180 minutes. The submission process lasts for ten hours. Fig. 10 shows the wDMR of SLO jobs and average JCT of DLT jobs from simulation and Kubernetes implementation. We consider configurations ($G[n]/T[m]$) with different job densities and cluster capacities: $G[n]$ denotes the cluster has n GPUs and $T[m]$ denotes m jobs are submitted per hour. For wDMR, the gap between simulation and Kubernetes prototype is small in consideration of real-world preemption overhead. For average JCT, the gap between simulation and Kubernetes is relatively larger when the GPU cluster is small. A possible reason is that Chronus mainly allocates GPUs to SLO jobs first. A small cluster has limited free GPUs for best-effort jobs, which can enlarge the performance gap. However, the difference is still acceptable and does not affect the conclusions from simulations.

6 RELATED WORKS

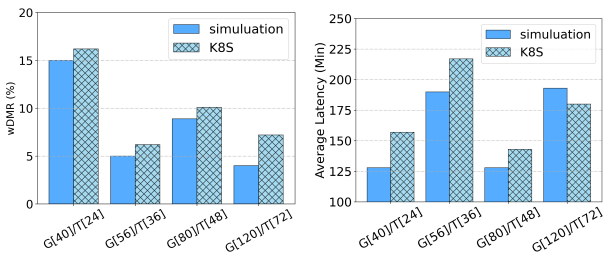
Deadline-aware scheduling. This classic problem was thoroughly studied in the context of network communication [6, 33]. Priority-based methods (e.g. Earliest Deadline First [32]) and rate control methods (e.g. RCP [13]) were adopted to satisfy the deadlines of network packets. Then researchers explored deadline-aware scheduling of big data jobs in cloud



(a) wDMR (b) Average JCT (c) wDMR versus job density (H_MIX2)

(d) wDMR versus job density (P_MIX2) (e) Average JCT versus job density (H_MIX2) (f) Average JCT versus job density (P_MIX2)

Figure 9: End-to-end comparisons between different schedulers.



(a) wDMR (b) Average JCT

Figure 10: Comparisons between simulation and kubernetes implementation.

computing. Some works [39, 48] modelled this scheduling task as an MILP problem. However, these algorithms are not tailored to the DLT jobs, and exhibit less effectiveness in GPU cluster scheduling. For instance, these methods cannot accurately and efficiently estimate the execution time of DLT jobs. They do not consider the impacts of GPU affinity and job preemption on the scheduling efficiency.

Recently, researchers started to focus on the SLO requirements of DLT jobs. HyperSched [30] maximizes the performance of hyper-parameter optimization jobs to meet the given deadlines. It cannot be applied to general DL tasks as we do in this paper. GENIE [7] develops an offline runtime estimation model to identify the best placement solutions for DL jobs. However, it does not allow user-specified SLOs. More importantly, these solutions do not consider the mixture of SLO and best-effort jobs in real-world GPU clusters.

Different from these works, we provide a scheduling system to satisfy the deadline requirements of SLO jobs and maximize the performance of best-effort jobs. It can be readily deployed in existing GPU clusters for general DLT jobs.

Job duration estimation. Job duration time is critical information for scheduling in datacenters. A variety of works propose to leverage the historic data [23, 47] and task structures [36, 37, 50] to predict the runtime of big data analytic jobs in an offline manner. For DLT jobs, several works [7, 14] leverage the DL model information to predict the completion time. Unfortunately, such solutions are not applicable to training jobs with unknown model types. Some works [16, 34] adopt online profiling to address this drawback. Chronus also follows this strategy. We introduce several techniques to enhance the profiling performance and efficiency over existing solutions, e.g., dynamical scaling of profiling resources.

Deep learning schedulers. A variety of scheduling systems were designed to optimize the execution of DLT jobs in GPU clusters from different perspectives. To maximize the resource utilization, Gandiva [53] designs a primitive to support DLT job packing, sharing and migration. Antman [54] introduces a fine-grained elastic mechanism to enable the co-execution of multiple jobs on a shared GPU. To improve the job performance, Tiresias [16] proposes a Discretized Two-Dimensional LAS to reduce the job completion time. Optimus [41] uses an online fitting model to predict the model training convergence and then minimize the training time. Pollux [42] dynamically adjusts the batch size and learning rate for each job to improve the cluster-wide throughput. To maintain the fairness of resource allocation, Themis [34] follows a finish-time fair manner to enable sharing incentive. $Gandiva_{fair}$ [4] leverages a novel automated trading scheme to incentivize users to release GPUs for cluster efficiency improvement and fairness guaranteeing. Unfortunately, they are not very effective in guaranteeing SLO requirements. This motivates us to design a new deadline-aware scheduler specifically for DLT jobs.

7 DISCUSSIONS AND FUTURE WORKS

Extension to heterogeneous resources. In this paper, we focus on homogeneous GPU clusters. Our system can be easily extended to heterogeneous clusters. Consider a cluster with various types of compute nodes and GPUs. For the `Selector`, we can introduce a new binary variable to represent the kind of GPU resources, and embed it into the constraint and objective of MILP. The `Profiler` and `Allocator` are also applicable to the new clusters. We will implement Chronus on heterogeneous clusters in the future.

Extension to auto-scaling DLT jobs. In the auto-scaling mechanism, a user specifies the range of GPUs for his DLT job. To handle this, we can introduce multiple binary variables to denote the selection of every value in that range, and adjust the constraint and objective of MILP optimization subsequently. This will bring larger search costs due to the increased search space.

Scheduling directed acyclic graph DLT jobs. A directed acyclic graph (DAG) DLT job is a collection of multiple tasks with high execution dependencies. Some tasks will be executed in sequence and others in parallel. It will be costly for our `Profiler` to estimate the runtime speed of each task: more GPUs allow parallel execution of many tasks at the cost of GPU resources, while fewer GPUs delay the progress of a DAG DLT job. A possible solution is to combine offline prediction and online profiling to balance this trade-off. For the `Selector`, we can use binary variables to indicate which tasks are executed in parallel, and design the corresponding constraints and objective of the MILP solver. The `Allocator` also needs to be redesigned to adapt to the DAG DLT jobs based on the relationships between performance and GPU affinity. We will consider this as another line of future work.

Handling rare cases in profiling. We make several assumptions about our system in Sec 3.1. In reality, there can be some rare cases that do not meet the assumptions. First, when a job adopts the loss-convergence stopping criteria instead of the fixed number of iterations, we can leverage the loss curve fitting technique in [34, 41] to estimate job runtime. Second, for some super-large models or model parallelism jobs, we can ask the users to reserve enough resources ahead of time for profiling.

Limitations of evaluations. In addition to the baseline schedulers in our evaluation, there are also some scheduling systems designed for elastic training, e.g., Optimus [41], Pollux [42]. We did not evaluate them as our traces do not contain enough information for simulation. We believe Chronus can outperform these elastic-aware DL schedulers since they are not designed for deadline guarantee. Besides, the elastic training techniques can also be easily integrated into Chronus. In the future, we will collect the required information from the physical environment and perform more extensive comparisons.

Reward function design. Our reward function has the range of [1, 100]. The cluster operator has the flexibility to adjust this range to adapt to the actual cluster environment. For instance, he can assign a larger reward value for more expensive DLT jobs, which can further increase the possibility of SLO guarantee.

8 CONCLUSION

This paper presents Chronus, a novel DLT scheduling system to satisfy the SLO requirements and maximize the performance of DLT jobs. We make innovations in the designs of job profiling, selection and resource allocation to improve the scheduling efficiency and effectiveness. Extensive simulations indicate that Chronus outperforms six state-of-the-art scheduling algorithms in reducing the deadline miss rate and job completion time. We also implement Chronus on the Kubernetes system in our production cluster, to demonstrate its practicability and validate the fidelity of simulation results. We expect our system can benefit existing GPU clusters in managing time-constraint DLT jobs.

9 ACKNOWLEDGEMENT

We thank our Shepherd Dr. Bailu Ding and anonymous reviewers for their valuable comments. This study is 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] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes. *ACM Queue* (2016).

[3] Giorgio C Buttazzo. 2011. *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media.

[4] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. 2020. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In *European Conference on Computer Systems*.

[5] Wei Chen, Jia Rao, and Xiaobo Zhou. 2017. Preemptive, low latency datacenter scheduling via lightweight virtualization. In *USENIX Annual Technical Conference*.

[6] Xiangwen Chen, Minghua Chen, Baochun Li, Yao Zhao, Yunnan Wu, and Jin Li. 2011. Celerity: A low-delay multi-party conferencing solution. In *ACM international conference on Multimedia*.

[7] Zhaoyun Chen, Wei Quan, Mei Wen, Jianbin Fang, Jie Yu, Chunyuan Zhang, and Lei Luo. 2019. Deep Learning Research and Development Platform: Characterizing and Scheduling with QoS Guarantees on GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems* (2019).

[8] Gurobi Company. 2021. Gurobi Optimization: https://https://www.gurobi.com/. https://www.gurobi.com/

[9] Carlo Curino, Djellel E Difallah, Chris Douglas, Subru Krishnan, Raghu Ramakrishnan, and Sriram Rao. 2014. Reservation-based scheduling: If you're late don't blame us!. In *ACM Symposium on Cloud Computing*.

[10] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGPLAN Notices* (2013).

[11] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and qos-aware cluster management. *ACM SIGPLAN Notices* (2014).

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

[13] Nandita Dukkipati, Nick McKeown, and Alexander G Fraser. 2006. RCP-AC: Congestion control to make flows complete quickly in any environment. In *International Conference on Computer Communications*.

[14] Yanjie Gao, Xianyu Gu, Hongyu Zhang, Haoxiang Lin, and Mao Yang. 2021. *Runtime Performance Prediction for Deep Learning Models with Graph Neural Network*. Technical Report MSR-TR-2021-3. Microsoft.

[15] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks. *arXiv preprint arXiv:1406.2661* (2014).

[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 Symposium on Networked Systems Design and Implementation*.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.

[18] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[19] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. Characterization and Prediction of Deep Learning Workloads i nLarge-Scale GPU Datacenters. In *International Conference for High Performance Computing, Networking, Storage, and Analysis*.

[20] Virajith Jalaparti, Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. 2012. Bridging the tenant-provider gap in cloud services. In *ACM Symposium on Cloud Computing*.

[21] 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 Annual Technical Conference*.

[22] Ru Jia, Yun Yang, John Grundy, Jacky Keung, and Hao Li. 2019. A deadline constrained preemptive scheduler using queuing systems for multi-tenancy clouds. In *International Conference on Cloud Computing*.

[23] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, et al. 2016. Morpheus: Towards automated slos for enterprise clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation*.

[24] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*. Citeseer.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* (2017).

[26] Kubernetes contributors. 2021. Kubernetes: https://kubernetes.io/. https://kubernetes.io/

[27] MIT Distributed Robotics Laboratory. [n.d.]. Github repository https://github/com/mit-drl/goop: Generalized Mixed Integer Optimization in Go. https://github.com/mit-drl/goop

[28] Tan N. Le, Xiao Sun, Mosharaf Chowdhury, and Zhenhua Liu. 2020. AlloX: Compute Allocation in Hybrid Clusters. In *Proceedings of the Fifteenth European Conference on Computer Systems*.

[29] Dan Li, Congjie Chen, Junjie Guan, Ying Zhang, Jing Zhu, and Ruozhou Yu. 2015. DCloud: deadline-aware resource allocation for cloud computing jobs. *IEEE transactions on parallel and distributed systems* (2015).

[30] Richard Liaw, Romil Bhardwaj, Lisa Dunlap, Yitian Zou, Joseph E Gonzalez, Ion Stoica, and Alexey Tumanov. 2019. Hypersched: Dynamic resource reallocation for model development on a deadline. In *ACM Symposium on Cloud Computing*.

[31] Jimmy Lin and Chris Dyer. 2010. Data-intensive text processing with MapReduce. *Synthesis Lectures on Human Language Technologies* (2010).

[32] Chung Laung Liu and James W Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* (1973).

[33] Shiyao Ma, Jingjie Jiang, Bo Li, and Baochun Li. 2016. Chronos: Meeting coflow deadlines in data center networks. In *International Conference on Communications*.

[34] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and Efficient GPU Cluster Scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation*.

[35] Jayashree Mohan, Amar Phanishayee, and Vijay Chidambaram. 2021. CheckFreq: Frequent, Fine-Grained {DNN} Checkpointing. In *19th USENIX Conference on File and Storage Technologies*.

[36] Kristi Morton, Magdalena Balazinska, and Dan Grossman. 2010. ParaTimer: a progress indicator for MapReduce DAGs. In *ACM SIGMOD International Conference on Management of data*.

[37] Kristi Morton, Abram Friesen, Magdalena Balazinska, and Dan Grossman. 2010. Estimating the progress of MapReduce pipelines. In *nternational Conference on Data Engineering*.

[38] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation*.

[39] Jun Woo Park, Alexey Tumanov, Angela Jiang, Michael A Kozuch, and Gregory R Ganger. 2018. 3sigma: distribution-based cluster scheduling for runtime uncertainty. In *Proceedings of the Thirteenth EuroSys Conference*.

[40] Mario Pastorelli. 2014. *Size-based disciplines for job scheduling in data-intensive scalable computing systems*. Ph.D. Dissertation. Télécom ParisTech.

[41] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*.

[42] Aurick Qiao, Keun Choe Sang, Jayaram Subramanya Suhas, Neiswanger Willie, Qirong Ho, Hao Zhang, Gregory R Ganger, and Eric P Xing. 2021. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. In *15th USENIX Symposium on Operating Systems Design and Implementation*.

[43] Elvis Rojas, Albert Njoroge Kahira, Esteban Meneses, Leonardo Bautista Gomez, and Rosa M Badia. 2020. A Study of Checkpointing in Large Scale Training of Deep Neural Networks. *arXiv preprint arXiv:2012.00825* (2020).

[44] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *ICLR*.

[45] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. 1999. Policy gradient methods for reinforcement learning with function approximation.. In *NIPS*.

[46] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *IEEE conference on computer vision and pattern recognition*.

[47] Alexey Tumanov, Angela Jiang, Jun Woo Park, Michael A Kozuch, and Gregory R Ganger. 2016. Jamaisvu: Robust scheduling with auto-estimated job runtimes. *Parallel Data Laboratory, Carnegie Mellon University, Tech. Rep.* (2016).

[48] Alexey Tumanov, Timothy Zhu, Jun Woo Park, Michael A Kozuch, Mor Harchol-Balter, and Gregory R Ganger. 2016. TetriSched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *European Conference on Computer Systems*.

[49] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Annual Symposium on Cloud Computing*.

[50] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*.

[51] Juan Pablo Vielma. 2015. Mixed integer linear programming formulation techniques. *Siam Review* (2015).

[52] Haoyu Wang, Zetian Liu, and Haiying Shen. 2020. Job scheduling for large-scale machine learning clusters. In *International Conference on emerging Networking EXperiments and Technologies*.

[53] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. 2018. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation*.

[54] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. AntMan: Dynamic Scaling on {GPU} Clusters for Deep Learning. In *USENIX Symposium on Operating Systems Design and Implementation*.

[55] Yang You. 2020. *Fast and Accurate Machine Learning on Distributed Systems and Supercomputers*. Ph.D. Dissertation. UC Berkeley.

[56] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Fan Yang, Lidong Zhou, Mao Yang, Francis C.M. Lau, Yuqi Wang, Yifan Xiong, and Bin Wang. 2020. HiveD: Sharing a GPU Cluster for Deep Learning with Guarantees. In *USENIX Symposium on Operating Systems Design and Implementation*.