Characterization and Prediction of Deep Learning Workloads in Large-Scale GPU Datacenters

Qinghao Hu[1,2] Peng Sun[3] Shengen Yan[3] Yonggang Wen[1] Tianwei Zhang[1*]
[1]School of Computer Science and Engineering, Nanyang Technological University
[2]S-Lab, Nanyang Technological University [3]SenseTime
{qinghao.hu, ygwen, tianwei.zhang}@ntu.edu.sg {sunpeng1, yanshengen}@sensetime.com

ABSTRACT

Modern GPU datacenters are critical for delivering Deep Learning (DL) models and services in both the research community and industry. When operating a datacenter, optimization of resource scheduling and management can bring significant financial benefits. Achieving this goal requires a deep understanding of the job features and user behaviors. We present a comprehensive study about the characteristics of DL jobs and resource management. First, we perform a large-scale analysis of real-world job traces from SenseTime. We uncover some interesting conclusions from the perspectives of clusters, jobs and users, which can facilitate the cluster system designs. Second, we introduce a general-purpose framework, which manages resources based on historical data. As case studies, we design (1) a *Quasi-Shortest-Service-First* scheduling service, which can minimize the cluster-wide average job completion time by up to 6.5×; (2) a *Cluster Energy Saving* service, which improves overall cluster utilization by up to 13%.

CCS CONCEPTS

• **Computing methodologies** → **Distributed computing methodologies**; **Machine learning approaches**; **Simulation evaluation**.

KEYWORDS

GPU Datacenter, Cluster Statistical Analysis, Deep Learning Training, Cluster Management System, Workload Scheduling, Energy Conservation, Time-series Prediction

ACM Reference Format:
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 *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3458817.3476223

1 INTRODUCTION

Over the years, we have witnessed the remarkable impact of Deep Learning (DL) technology and applications on every aspect of our daily life, e.g., face recognition [65], language translation [64], advertisement recommendation [21], etc. The outstanding performance of DL models comes from the complex neural network structures and may contain trillions of parameters [25]. Training a production model may require large amounts of GPU resources to support thousands of petaflops operations [15]. Hence, it is a common practice for research institutes, AI companies and cloud providers to build large-scale GPU clusters to facilitate DL model development. These clusters are managed in a multi-tenancy fashion, offering services to different groups and users based on their demands, with resource regulation and access controls.

A job scheduler is necessary to manage resources and schedule jobs. It determines the resource utilization of the entire cluster, and job performance, which further affects the operation cost and user experience. Understanding the characteristics of DL workloads is indispensable for managing and operating GPU clusters. DL jobs share some similar features as conventional HPC workloads, which are generally different from big data jobs in cloud computing (e.g., MapReduce). (1) *Iterative process* [46, 56]. Similar to some numerical simulation and analysis jobs in HPC, typical DL training jobs are also iterative computation. The target model is obtained through the gradient descent update iteratively and the long-term training process can be suspended and resumed via checkpoints. (2) *Gang scheduling* [26]. DL training jobs require all the GPUs to be allocated simultaneously in an all-or-nothing manner [18, 78]. This may cause resource fragmentation in GPU clusters. (3) *Exclusive allocation* [31, 39]. The GPU resources are allocated exclusively in DL datacenters. Although the advanced NVIDIA Multi-Instance GPU (MIG) technology [5] provides intrinsic hardware-level support for fine-grained sharing on NVIDIA A100 GPUs, existing GPU datacenters built with previous-generation GPUs typically only support coarse-grained GPU allocation [81].

Furthermore, DL training jobs also exhibit some unique features different from most HPC or big data workloads. (1) *Inherent heterogeneity* [77]. DL training jobs typically require a number of GPUs as the dominant resources, as well as associated resources such as CPUs and memory. (2) *Placement sensitivity* [46]. Multi-GPU jobs usually perform gradient synchronization to update model parameters at the end of each iteration. Better interconnect topology can achieve lower communication overhead and faster training speed for multi-GPU jobs. Meanwhile, colocating multiple jobs in one server could cause performance degradation since the interference of system resources like PCIe bandwidth [39]. (3) *Feedback-driven*

[*]Corresponding author.

SC '21, November 14–19, 2021, St. Louis, MO, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8442-1/21/11…$15.00
https://doi.org/10.1145/3458817.3476223

exploration [77]. To train a model, users typically try several configurations, and use early feedback from these jobs to decide which one should be kept or killed. This gives higher cancellation rates in GPU clusters [39].

A variety of works proposed new schedulers specifically for GPU clusters to achieve higher performance, resource utilization and fairness [31, 46, 52, 78, 84]. To adapt to the characteristics of DL workloads, these works adopt the Philly trace [39] from Microsoft GPU cluster for design and evaluation. To the best of our knowledge, the Philly trace is the only publicly available DL workload dataset so far. However, analysis of only this trace may be inadequate to prove the generality of the designs and can cause the overfitting issue, as pointed in [9, 55]. In addition, this trace was collected in 2017. Due to the rapid development of DL technology and demands, this trace may not be able to reflect the latest characteristics of DL jobs anymore.

In this paper, we present an in-depth study about the characterization of DL workloads and scheduler designs. Our study is based on a new set of DL job traces collected from a datacenter in SenseTime, named Helios. These traces have the following benefits. (1) They were collected over six months in 2020, which can represent the emerging DL algorithms and models. (2) They incorporate more than 3 million jobs among four independent GPU clusters. These jobs exhibit high variety, covering different types (training, inference, data preprocessing, etc.), applications (computer vision, natural language processing, etc.), and purposes (product development, research etc.). (3) These traces include rich information, which enables thorough analysis from different aspects, and diverse evaluations of scheduling systems. It is worth noting that we do not aim to compare our traces with Philly and show the advantages. Instead, we release our traces and expect they can be an alternative to Philly. Our traces together with Philly can increase the diversity of job workloads to enable more general design and evaluations (as we will do for our prediction framework in this paper). We also hope this can inspire other organizations to release their job traces to benefit the community of deep learning systems.

Based on the traces, this paper makes two key contributions. First, we perform a large-scale analysis of the DL jobs in the four clusters, from the perspectives of jobs, clusters and users. Prior works [39, 73] only focused on the job-level characterization. Our study provides more extensive analysis about the behaviors of GPU clusters and users. Through our analysis, we identify seven new implications, which can shed light on the design of GPU clusters.

Second, we introduce a novel prediction-based framework to manage compute resources and schedule DL jobs. This framework is inspired by some implications from our trace analysis: the characteristics of clusters and jobs exhibit obvious predictable patterns, which gives us the opportunities to forecast those behaviors in advance and then optimize the management of jobs and resources. This framework can be integrated with different services. Each service builds a machine learning model from the historical data, and predicts the future states of the cluster, or upcoming job information. Based on the prediction results, the service can make optimized actions for resource and job management. To maintain prominent performance, the prediction model is also kept updated with new data. We present two services as case studies: (1) A *Quasi-Shortest-Service-First* scheduling service can assign each new job a priority score based on the historical information. It then schedules the jobs based on these priority scores, which can reduce the queuing delay by up to 20.2× and improve the overall job completion time (JCT) by up to 6.5×. (2) A *Cluster Energy Saving* service can proactively predict the demanded compute nodes in advance, and then leverages the Dynamic Resource Sleep (DRS) technique to efficiently power off the unnecessary nodes. It can improve up to 13% of node utilization rate and conserve millions of kilowatt hours of electricity annually across the four clusters.

Table 1: Configurations of four clusters in Helios (All data are collected on September 1st, 2020, except # of jobs and VCs, which cover the period of April-September, 2020).

	Venus	Earth	Saturn	Uranus	Total
CPU	Intel, 48 threads/node		Intel, 64 threads/node		-
RAM	376GB per node		256GB per node		-
Network	IB EDR		IB FDR		-
GPU model	Volta	Volta	Pascal & Volta	Pascal	-
# of VCs	27	25	28	25	105
# of Nodes	133	143	262	264	802
# of GPUs	1,064	1,144	2,096	2,112	6,416
# of Jobs	247k	873k	1,753k	490k	3,363k

2 BACKGROUND

In this section, we first introduce our GPU datacenter, dubbed Helios (§2.1). Then we describe the DL workloads running in this datacenter (§2.2) and the corresponding job traces (§2.3).

2.1 Helios Datacenter

Helios is a private datacenter dedicated to developing DL models for research and production in SenseTime, containing multiple multi-tenant clusters. In this paper, we select 4 representative clusters: Venus, Earth, Saturn, and Uranus. Table 1 shows the configurations of each cluster. Note that Saturn is a heterogeneous cluster with mixed NVIDIA Pascal and Volta GPUs, while the other three clusters are composed of identical Volta or Pascal GPUs. Our conclusions from the analysis of these 4 clusters are general for other clusters in Helios as well.

Each cluster in Helios serves multiple groups in SenseTime concurrently. To support resource isolation and management for multi-tenancy, a cluster is further divided into several Virtual Clusters (VCs), and each VC is dedicated to one group with its demanded resources. All GPUs within one VC are homogeneous. Each node is exclusively allocated to one VC, and over-subscription of GPU resource quota is not enabled in Helios. Configuration of a VC can be dynamically changed in three situations: (1) when the demand of a group is increased, new servers will be purchased and allocated to its VC, or form a new VC; (2) when a group is less busy, the size of its VC may be scaled down; (3) when groups are combined or split, their VCs are also merged or split correspondingly.

To reduce the communication overhead in distributed workloads, GPUs are interconnected to each other via a hierarchic network: (1) intra-node communication is achieved via the high-bandwidth and low-latency PCIe (for Pascal GPUs) or NVLink (for Volta GPUs) [7] ; (2) inter-node communication within the same RDMA domain is achieved via the high-speed InfiniBand. To improve the workload performance and reduce network interference, cross-RDMA-domain distributed training (communication through TCP/IP) are not allowed in Helios.

The distributed storage system is also critical for workload performance. To support the massive data throughput in DL jobs, Helios adopts Lustre [2] as the file system, and the input data for the jobs are stored in Ceph [75] as the object storage. In addition, Memcached [3] is used to accelerate data access.

The *Slurm* workload manager [79] is adopted to regulate the resources and job execution. Specifically, it dynamically adjusts the configurations of VCs, including the resource quota in each VC, total number of VCs, job time limit, etc. Meanwhile, it is also responsible for the online scheduling of DL jobs, following three steps. (1) A user submits his or her job to a VC with specified job resource demands (e.g., numbers of GPUs and CPUs). If the CPU requirement is not specified, the scheduler will allocate CPU cores proportional to the requested GPU counts. (2) *Slurm* maintains a separate allocation queue for each VC (*VCQueue*), and selects jobs for scheduling. All jobs in one *VCQueue* have the same priority setting so the scheduling order is only determined by the job's submission time. (3) *Slurm* allocates jobs in a consolidated paradigm by packing jobs into as few nodes as possible. A user can also select specific nodes if he or she has special topology requirements. For instance, some exploratory jobs are placed across specific numbers of nodes for testing the performance impact of GPU affinity. Only 0.15% of the jobs are placed in a way customized by the users. After the job is scheduled, it keeps running until completion or being terminated by the user. Preemption is not supported in Helios.

2.2 Workloads in Helios

Helios supports various types of jobs in the DL development pipeline, e.g., data preprocessing, model training, inference, quantization, etc. These workloads are submitted by product groups for developing commercial products, as well as research groups for exploring new technologies. They range over different DL domains, including computer vision, natural language processing, reinforcement learning, etc.

A majority of the GPU jobs are DL training, which mainly follow an iterative fashion [31]: the training task consists of many iterations, where gradient descent is applied to update model parameters based on the mini-batch data in each iteration. To scale with complex models and large datasets, many jobs adopt the distributed data-parallel training scheme across multiple GPUs. In each iteration, every GPU processes a subset of data in parallel and then performs gradient synchronization to update model parameters. This synchronization typically adopts the parameter sever [44] or all-reduce strategy for high-speed multi-node/multi-GPU communication (e.g., NCCL [4] as backend). Users mainly adopt the built-in libraries (e.g., DistributedDataParallel in Pytorch, MultiWorkerMirroredStrategy in Tensorflow) to implement their jobs.

In addition, there are also a quantity of jobs for data/model preprocessing and postprocessing. For instance, some CPU jobs generate large-scale training datasets by extracting frames from videos; some jobs rescale the images according to the model's requirements. To speed up model inference, it is common to perform post-training quantization to reduce model size before deployment.

Table 2: Comparisons between Helios and Philly traces.

	Helios	Philly		Helios	Philly
# of clusters	4	1	Duration	6 months	83 days
# of VCs	105	14	Average # of GPUs	3.72	1.75
# of Jobs	3.36M	103k	Average Duration	6,652s	28,329s
# of GPU Jobs	1.58M	103k	Maximum # of GPUs	2,048	128
# of CPU Jobs	1.78M	0	Maximum Duration	50 days	60 days

2.3 DL Job Traces from Helios

We collect jobs from each of the 4 clusters in Helios, which serves as the basis of our analysis and system design in this paper. These four traces span 6 months from April 2020 to September 2020, covering a total of 3.36 million jobs over 802 compute nodes with 6416 GPUs. Each trace contains two parts: (1) We collect the job logs through the *Slurm* sacct command, which provides the rich information for each job. (2) The daily VC configurations of each cluster from *Slurm*. Besides, we leverage node allocation details from the job logs to infer the timing information for each cluster.

To the best of our knowledge, this is the largest set of DL job traces, and also the first one with comprehensive types of jobs in addition to DL training. We release these traces to the research community, and expect they can benefit researchers for DL workload analysis and design of GPU datacenter systems.

2.3.1 *Terminology.*

We emphasize some terminologies in the job traces, which will be widely mentioned in our following analysis.

Job status: a job can end up with one of five statuses: (1) completed: it is finished successfully; (2) canceled: it is terminated by the user; (3) failed: it is terminated due to internal or external errors; (4) timeout: the execution time is out of limit; (5) node fail: it is terminated due to the node crash. Timeout and node fail are very rare in our traces, and will be regarded as failed in this study.

CPU job: this job is executed without any GPUs (e.g., image preprocessing, file decompression).

GPU job: the job needs to be executed on GPUs for acceleration (e.g., DL model training, model evaluation).

GPU time: this metric is used to quantify the amount of GPU resources required by the job. It is calculated as the product of total execution time and the number of GPUs.

CPU time: this is the product of total execution time and the number of CPUs. It is only considered for CPU job analysis.

Cluster utilization: this metric is used to characterize the resource utilization of a cluster. Since GPUs are the dominant resources in DL jobs, we calculate the cluster utilization as the ratio of active GPUs among the total GPUs in the cluster.

2.3.2 *Comparisons with the Philly Trace.*

Microsoft released a trace of DL training jobs from its internal cluster Philly [39]. It is currently the most popular public trace containing rich information about production-level DL training jobs. A quantity of works on GPU resource management and optimization leveraged this Philly trace for analysis and evaluation [31, 34, 46, 52, 58, 78, 84].

DL has experienced rapid development over the years. New models and algorithms are emerging with increased resource demands [15, 25]. There is a 10.5× year-by-year increase in the number of DL jobs in Microsoft datacenters [31]. Hence, the Philly trace collected in 2017 may not be able to accurately reflect the characteristics of

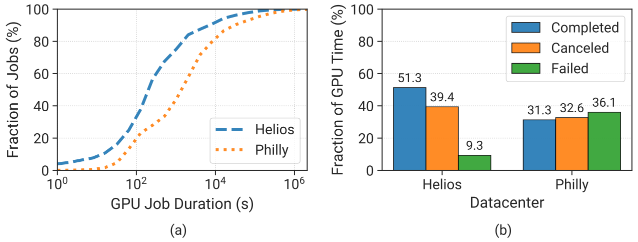


Figure 1: Comparisons of job characteristics between Helios and Philly. (a) The CDFs of the GPU job duration. (b) Distribution of GPU time by the final job status.

modern GPU clusters and DL jobs. We expect our trace can fill this gap with a larger number of jobs and the latest DL models and algorithms. We present detailed comparisons between our Helios trace and Microsoft Philly trace. Although the Philly trace ranges from August 2017 to December 2017, jobs in the first two months exhibit abnormal behaviors (much less density and different features). So we select the period of October 2017 to December 2017 with 103,467 jobs, which was also adopted in the original paper [39].

Table 2 summarizes the comparisons between Helios and Philly traces. Our traces are collected from 4 independent clusters, while the Philly trace only describes one cluster. Helios contains 32.6 times more jobs than Philly and around half of jobs from Helios are CPU jobs. We will reveal some interesting insights through analyzing different characteristics of CPU & GPU jobs in §3, which cannot be learned from Philly. Moreover, the average number of GPUs required by our GPU jobs is over twice that of Philly. The maximum number of requested GPUs from our traces is 2048, which is an order of magnitude higher than Philly. These indicate that our traces provides new features of GPU clusters and jobs, which can increase the plurality and generality in DL system research.

We calculate the average duration of GPU jobs from Helios, which is much lower than Philly. There are three possible reasons: (1) Philly adopted *A YARN* [70] to schedule jobs, where failed jobs would be retried for a fixed number of times. Such retrials were counted into the entire job duration, statistically resulting in longer execution time. For instance, the longest job took about 60 days for a total of 3 attempts due to job failures, although the successful execution only took around 25 days. If we profile the Philly trace by regarding each attempt as an individual job, the average duration will drop to 17,398 seconds. (2) We keep all GPU jobs in Helios to reflect the realistic characteristics of a GPU datacenter. These include debugging and testing jobs which are much shorter than normal training jobs, causing a lower average duration. (3) Our datacenter provides more computing resources. Users usually request more resources (two more GPUs on average) for training, which can also significantly accelerate the training process.

Figure 1(a) illustrates the Cumulative Distribution Function (CDF) of job duration. We observe that Philly jobs statistically took more time than Helios. This is consistent with our analysis from Table 2. Figure 1(b) shows the percentages of GPU time occupied by jobs with different statuses. We can see a significant fraction of GPU time contributed to the jobs which were finally ended up with the failure or canceled status. Particularly, over one-third of GPU time was wasted for the failed jobs in Philly and 9.3% in Helios.

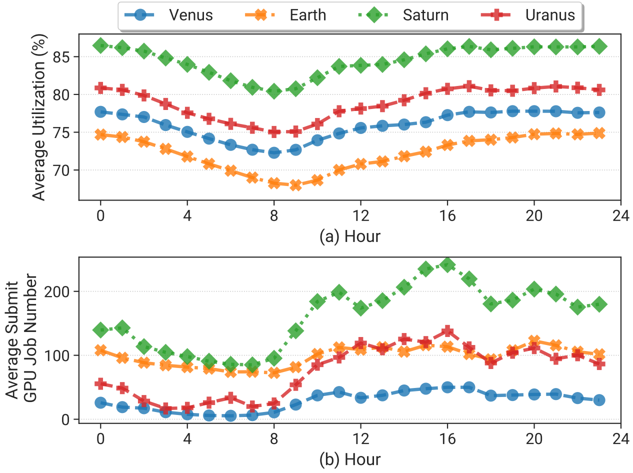


Figure 2: Daily pattern of the cluster usage in Helios. (a) Hourly average cluster utilization over six months. (b) Hourly average GPU job submission rates over six months.

3 CHARACTERIZATION OF DL JOBS

In this section, we perform a thorough analysis of our job traces. Some prior works analyzed the traditional big data traces from real-world datacenters [19, 59, 61, 68]. In contrast, very few studies focused on the analysis of DL jobs. [39, 73] performed an empirical study of the characteristics of their clusters. We give more comprehensive analysis from the perspectives of clusters (§3.1), jobs (§3.2) and users (§3.3).

Our traces cover different characteristics of DL jobs and behaviors of AI developers and researchers in SenseTime. For instance, users can submit long-term production jobs, as well as short-term exploratory jobs for debugging purposes. Users might early stop their jobs when they can (not) reach the expected performance. We perform our assessment statistically, and believe the conclusions are general for other organizations and clusters as well.

3.1 Cluster Characterization

3.1.1 Daily Trends of Cluster Usage. Figure 2(a) shows the average cluster utilization for every hour in one day. All the clusters and users are in the same timezone, and hence exhibit similar patterns for the daily cluster usage. The utilization of all the clusters ranges from 65% to 90%. Saturn has the highest utilization, while Venus and Earth are relatively underutilized. The standard deviation of hourly utilization in Saturn is 7% and ranging from 10% to 12% in other clusters. Besides, we observe a 5~8% decrease at night (0 am – 8 am) for all the clusters, which is not very significant. This is because the workloads in Helios are mainly DL training jobs, which can take hours or days to complete. It is common that some jobs are submitted in the daytime but still keep running overnight.

Figure 2(b) shows the average GPU job submission rate for each hour during the six months. All the clusters have similar patterns of the job submission in one day: the number drops to the lowest point at night (sleep), and experiences a slight drop around 12pm (lunch) and 6pm (dinner). It is also interesting to note that Earth has a stable and high submission rate (~100 jobs) per hour, but

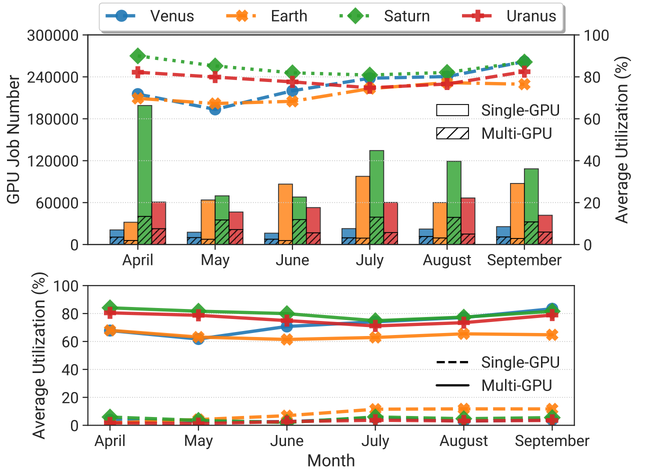


Figure 3: Monthly trends of cluster activities in Helios. Top: number of submitted (single- and multi-GPU) jobs (*bars*) and average cluster utilization (*dashed lines*). Bottom: average cluster utilization from multi-GPU jobs (*solid lines*) and single-GPU jobs (*dashed lines*).

has the lowest utilization among the four clusters. This is because the GPU jobs in Earth are overall shorter than the other clusters. The cluster utilization depends on both the number of jobs as well as their running time. Further, the frequency of job submission is much lower than big data clusters [9, 68]. This implies some time-consuming scheduling optimization algorithms would apply for scheduling DL training jobs.

Implication #1: Both the cluster utilization and the job submission rate exhibit obvious daily patterns. This provides us opportunities to predict those behaviors in advance and then perform the optimal resource management and job scheduling.

3.1.2 *Monthly Trends of Cluster Usage.*

We further analyze the monthly trends of GPU resources and job behaviors. Figure 3 (top) shows the monthly statistics of GPU job submission[1] (bars) and average cluster utilization (dashed lines). All the clusters have stable submissions of multi-GPU jobs each month, while the numbers of single-GPU jobs fluctuate dramatically. The utilization of Saturn and Earth remains stable under varied numbers of GPU jobs per month. Surprisingly, Saturn executed almost twice GPU jobs in July compared with May or June, whereas the utilization in May (85.21%) and June (81.92%) is even higher than July (80.87%). Furthermore, we find the distribution of multi-GPU jobs within the same cluster is similar each month. The average number of requested GPUs is very close with a standard deviation of 2.9. Therefore, we can accurately predict the monthly submissions of multi-GPU jobs from the previous months' data. Figure 3 (bottom) presents the cluster utilization contributed by single-GPU and multi-GPU jobs. It is evident that single-GPU jobs have little influence on the overall cluster utilization (less than 6% except for Earth). Conversely, multi-GPU jobs are dominant to cluster utilization.

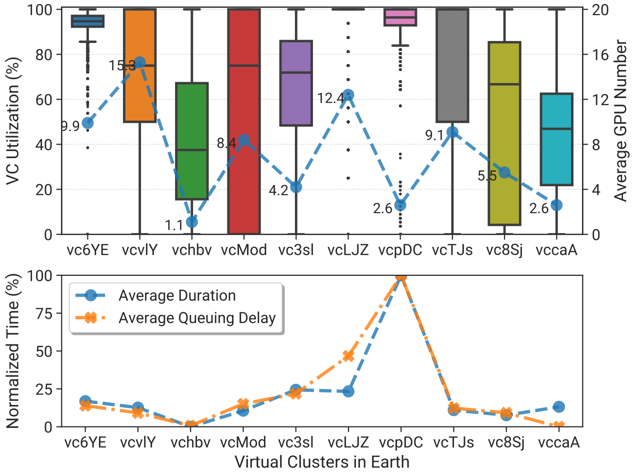


Figure 4: VC behaviors in Earth. Top: The boxplot of utilization distributions for the top 10 largest VCs and the job's average number of requested GPUs in each VC (*dashed line*). Bottom: Min-Max normalized average job duration (blue dashed line) and queuing delay (orange dashed line).

Implication #2: For monthly trends, it is infeasible and unnecessary to predict the submissions of single-GPU jobs due to their weak impact on the cluster usage. In contrast, multi-GPU jobs exhibit more stable monthly patterns, and are critical to cluster utilization, which we can predict for better scheduling efficiency.

3.1.3 *Virtual Cluster Behaviors.*

In addition to the entire physical clusters, investigation of VCs is also indispensable. We select a period when the VC configuration remains stable (May in Earth). Figure 4 (top) shows the utilization distributions of the 10 largest VCs (in descending order) averaged per minute. Specifically, there are 208 GPUs in *vc6YE* and 32~96 GPUs in other VCs. Each box is framed by the first and third quartiles, while the black line inside the box represents the median value. Both whiskers are defined at 1.5 times the InterQuartile Range (IQR). We plot the job's average number of requested GPUs for each VC above the corresponding box. Figure 4 (bottom) shows the average queuing delay and job duration for each VC.

We find the behaviors of each VC vary significantly, as they run different types of GPU jobs in terms of resource demands and duration. First, we observe that the VC utilization is positively correlated with the average GPU demands. *vc6YE* and *vcLJZ* keep over 90% utilization most of the time as they generally run large jobs. In contrast, the utilization of *vchbv* and *vccaA* is basically below 65% for hosting small jobs. One exception is *vcpDC*, which has high utilization but small average numbers of GPUs. Second, the job queuing delay is approximately proportional to the average job duration. Busy VCs (e.g., *vcLJZ* and *vcpDC*) typically have much longer queuing delay. These prove that job queuing and resource underutilization co-exist in our clusters due to imbalanced VCs.

The key reason is the adoption of static partitioning with VCs. This simple and mature solution is widely used in production GPU clusters (e.g., Microsoft [39, 84], Alibaba [73, 78]) for fairness among multi tenants. However, it also causes long queuing delay

[1] Our traces end on September 27th. So the reported numbers of September are around 10% lower than the actual one.

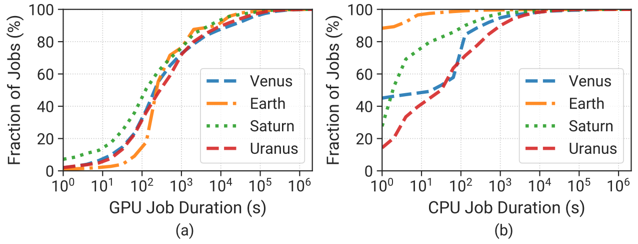


Figure 5: The CDFs of (a) GPU and (b) CPU job duration.

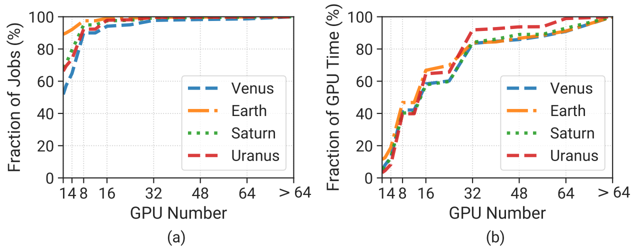


Figure 6: The CDFs of job sizes (in GPU number) with the number of (a) jobs and (b) GPU time.

and resource underutilization. Researchers also designed advanced scheduling algorithms to address these issues with high fairness [18, 46, 84]. How to implement them into production clusters with better reliability and robustness will be an important future work.

Implication #3: Different groups submit DL jobs to their VCs with distinct GPU demands and duration. Hence, the imbalanced resource allocation across VCs can lead to low resource utilization and severe job queuing delay [51]. It is critical to consider fairness when designing schedulers for shared clusters [18, 46, 84].

3.2 Job Characterization

Beyond the cluster statistics, our traces also contain rich information at the job level. We draw interesting conclusions from such information, as discussed below.

3.2.1 Job Execution Time. As shown in Table 2, the number of GPU jobs is close to the number of CPU jobs in the Helios traces. However, the average execution time of GPU jobs (6,652s) is 10.6× longer than CPU jobs (629s). More than 50% of CPU jobs run for less than 2s. In contrast, the median execution time of GPU jobs is 206s. More specifically, Figure 5 compares the duration distributions of GPU and CPU jobs in each cluster. The duration of GPU jobs ranges from seconds to weeks, and is generally over an order of magnitude longer than CPU jobs. Interestingly, these four clusters have diverse duration distributions of CPU jobs, while similar distributions for GPU jobs. In Earth, short-term CPU jobs account for a larger portion compared with other clusters: nearly 90% of CPU jobs run for only one second in Earth. Most of them are related to training progress and node state queries. As for GPU jobs, roughly three-quarters of jobs last for less than 1000 seconds as they are mainly for model evaluation and program debugging.

To dive deeper into the characteristics of GPU jobs in each cluster, we investigate the relationship between the GPU demands, GPU time and number of GPU jobs. We show the CDFs of requested GPU demands with the number of jobs (Figure 6(a)) and GPU time (Figure 6(b)). We observe there are over 50% single-GPU jobs in each cluster, and the largest ratio is 90% in Earth. However, they only occupy 3~12% of the total GPU time. In contrast, although the proportion of large-size jobs (≥8 GPUs) is smaller than 10%, they account for around 60% of computing resources.

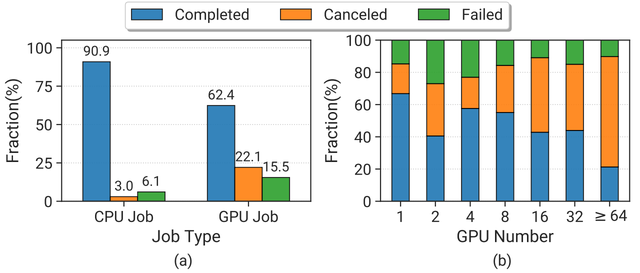


Figure 7: Distribution of jobs by their final statuses. (a) Comparisons of CPU and GPU jobs for their final statuses. (b) Percentages of final job statuses w.r.t. different GPU demands.

Implication #4: Despite the number of single-GPU jobs is predominant, GPU resources are mainly consumed by multi-GPU jobs. Hence, optimization of multi-GPU jobs is more important to improve cluster efficiency. This characteristic resembles traditional HPC workloads [9, 55, 60] and implies some optimization techniques in HPC can also be applied to GPU clusters.

3.2.2 Job Final Statuses. Figure 7(a) summarizes the distributions of final statuses for CPU and GPU jobs of these 4 clusters. The ratio of unsuccessful GPU jobs (37.6%) is significantly higher than CPU jobs (9.1%). One reason is that some users prefer to terminate their DL training jobs in advance as the model accuracy already converges to the satisfactory value. These canceled jobs are still successful. Another reason is that users can inspect the training states and kill the poor-performing jobs earlier. This reflects the *feedback-driven exploration* feature of DL training jobs. This early-stopping feature can be leveraged to optimize the scheduling efficiency. For instance, [56, 77] help users automatically make the early-stopping decision through recording training feedback and predicting future training performance. This can bring a huge benefit to datacenters.

There are also other causes for failed jobs, including timeout, node failure, incorrect inputs, runtime failure, etc. Microsoft [39, 82] presented a detailed analysis of the reasons for training job failures, so we do not conduct similar investigations in this paper.

Implication #5: Since many DL training jobs can reach the convergence earlier than expected, the scheduler can automatically detect this condition and stop the jobs for resource efficiency [56, 77]. Users can use different metrics (e.g., loss, accuracy) and authorize the scheduler to monitor and manage their jobs.

In Figure 7(b), we quantify the ratios of different job final statuses in terms of GPU demands. We only consider the GPU numbers of $2^k (k \in \mathbb{N})$ as they are mostly requested in Helios. We observe that the ratio of job completion keeps decreasing as the number of GPUs increases, with an exception of 2-GPU jobs. For large jobs with 64 or

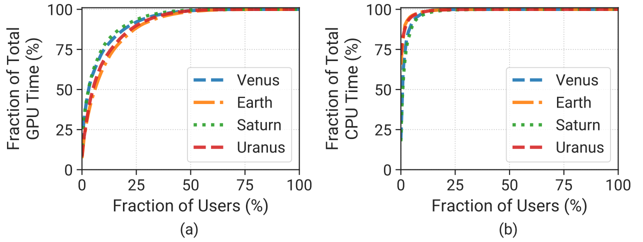


Figure 8: The CDFs of users that consume the cluster resources in terms of (a) GPU Time (b) CPU Time.

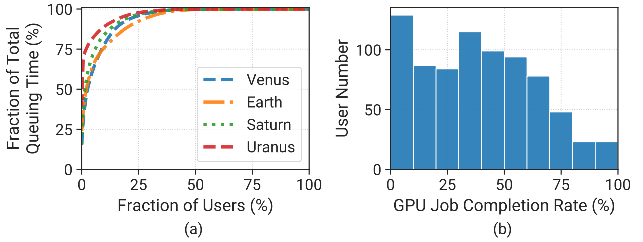


Figure 9: (a) The CDFs of users w.r.t. GPU job queuing delay. (b) The distribution of user GPU job completion ratios.

more GPUs, only fewer than a quarter of jobs complete successfully while the canceled ratio even reaches roughly 70%. This is because these jobs typically run for very long time, and users have higher chances to early stop them to save time and resources.

Additionally, we find most failed jobs are terminated within a short time, which matches the conclusions in prior works [39, 82]. The majority of failures are incurred by user errors, such as script configuration, syntax/semantic errors in the program. However, plenty of short-term debugging jobs suffer from severe queuing delays. Users usually fail to get the code debugging feedback timely, which considerably affects their experience.

Implication #6: A lot of failed jobs are for debugging purposes, and last for a very short time. However, they are mixed with the long-term production jobs in the queue and possibly suffer from much longer waiting time than execution. A possible solution is to allocate a special VC for debugging jobs and enforce a short-term limit (e.g., 2 minutes). This can help users obtain error messages timely and filter most failed jobs for normal clusters.

3.3 User Characterization

We analyze the traces from the users' perspective, to discover methods for user experience enhancement. This has never been considered in prior works about DL job analysis. Each cluster has 200~400 users. Some users can submit jobs to multiple clusters concurrently.

Similar to our analysis of job trends, we first investigate the consumption of CPU and GPU resources at the user level, as shown in Figure 8. We find the trends are similar across all the clusters. Compared with GPU time, the CDF curves of CPU time are much steeper, indicating CPU jobs are more concentrated within a small portion of users. This is because only 25% of users on average need to conduct CPU tasks (e.g., video frames extraction), and the top 5% of users occupy over 90% CPU time. In contrast, almost every user has GPU training jobs, and the top 5% of users consume 45~60% GPU time.

Next, we study the distributions of GPU job queuing delay among users, as shown in Figure 9(a). We observe that most users do not suffer from severe job queuing, whereas a few users have jobs blocked for a long time. In Uranus, the top 1% of users (only 3) bear over 70% queuing time, even they are not among the top 10 resource-consumption users. We name them "marquee users" [76], and their experiences need to be ameliorated.

Figure 9(b) shows the distributions of users for different GPU job completion rates. It is obvious that the users' GPU job completion rates are generally low, which proves that the high fraction of unsuccessful GPU jobs (shown in Figure 7) reflects the users' overall behaviors instead of some individual ones.

Implication #7: To alleviate the problem of unfair cluster queuing, it is recommended that the scheduler should consider our user-level analysis to perform the corresponding optimization. For instance, the scheduler can dynamically adjust temporary priorities to users, especially to the marquee ones, based on their current job queuing statuses. The VC configuration can also be regulated appropriately according to users' behaviors.

4 A PREDICTION-BASED FRAMEWORK

From §3, we find the feasibility of predicting clusters' behaviors (e.g., job duration, node states) from the history. Inspired by this observation, we design a novel prediction-based GPU resource management framework, which leverages the historical data to improve the resource usage efficiency and workload performance.

4.1 Framework Overview

Figure 10 illustrates the overview of our framework. It is designed as a centralized manager built atop each GPU cluster. It adopts the "plug-and-play" fashion, where different resource management services can be integrated into this framework. Each service is independent and targets a different perspective of optimization. They share common design philosophy, and follow the same workflow. The cluster operators can select services based on their demands.

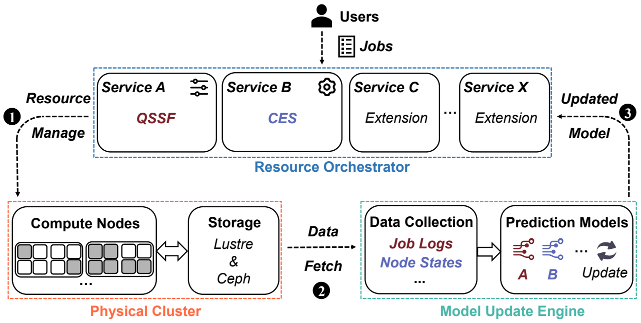


Figure 10: Overview of our prediction-based framework.

The framework consists of two components: *Model Update Engine* and *Resource Orchestrator*. For each service, a machine learning model is trained to predict the job behaviors or cluster states. During the operation, the *Resource Orchestrator* uses the model to predict the upcoming *events* and determines the optimal *resource management* operation (❶), which will be executed in the cluster. Meanwhile, the *Model Update Engine* fetches the run-time data (❷) regularly (e.g., every minute) or triggered by events, and fine-tunes the model periodically to adapt to the changes in the cluster (❸).

Our framework has several benefits. (1) *Extensibility*: we provide an abstract pipeline for resource management. As case studies, we design two novel services: *Quasi-Shortest-Service-First Scheduling* to minimize the cluster-wide average JCT (§4.2), and *Cluster Energy Saving* to improve the cluster energy efficiency (§4.3). Other services based on machine learning prediction can also be integrated into our framework, e.g., burstiness-aware resource manager [80, 83], network-aware job scheduler [12, 38], etc. (2) *High usability*: our framework can be deployed into arbitrary GPU clusters. The services work as plugins atop the current management systems (e.g. *Slurm*, *YARN*, *Kube-scheduler*) with minimal or no modifications to them. Users do not need to provide extra information or specifications. (3) *Low overhead*: the prediction and operation latency for each service typically takes milliseconds, which is negligible for DL workloads.

4.2 Quasi-Shortest-Service-First Scheduling

4.2.1 Motivation. All GPU clusters in Helios are managed by *Slurm*. Jobs are scheduled using the vanilla First-In-First-Out (FIFO) algorithm, similar to *YARN*'s Capacity Scheduler [70]. Due to such runtime-unaware scheduling style [69], users complain that even short-term jobs can suffer from long queuing delays.

Both Shortest-Job-First (SJF) and Shortest-Remaining-Time-First (SRTF) algorithms were proposed to reduce the average JCT [29–31, 63] with and without preemption respectively. However, they are too ideal and thus impractical in GPU clusters, due to the following two reasons: (1) some jobs in GPU clusters do not have iterative characteristics, and cannot be preempted and then restored from the checkpoints. Hence, the preemption-enabled scheduling algorithms (SRTF) are not applicable for these jobs. (2) These algorithms rely on the information of job duration or remaining execution time, which is uncertain and could be affected by many unexpected factors (e.g., termination in advance, training early-stopping [57], error and node crash, etc.). To address this issue, prior works try to obtain the job runtime information from the users [16, 33, 70], job profiling [13, 27, 37, 71, 72] or leveraging the job's periodic features [22, 24, 38, 41]. These approaches can either affect the usability or operation cost, or lack of generality for different types of workloads.

Driven by these limitations, we design a new Quasi-Shortest-Service-First (QSSF) scheduling service. This scheduler adopts the non-preemption mechanism, which can be applied to different types of jobs in our datacenter. Owing to the constraint of the gang scheduling for DL training jobs, large-size short-term jobs can occupy many GPUs, which can block multiple small-size jobs using the SJF scheduler [31]. We choose to rank jobs with GPU time instead of duration. Through the trace analysis, we find there exist strong correlations between the job duration and some attributes, such as job name, user, GPU demands, submission time, etc. Hence, our scheduler leverages these attributes to predict the jobs' priority orders (i.e., expected GPU time) for scheduling. Similar ideas were also applied for scheduling big data workloads [54, 69]. We consider more attributes with a machine learning model ensemble to enhance the scheduling performance for DL jobs.

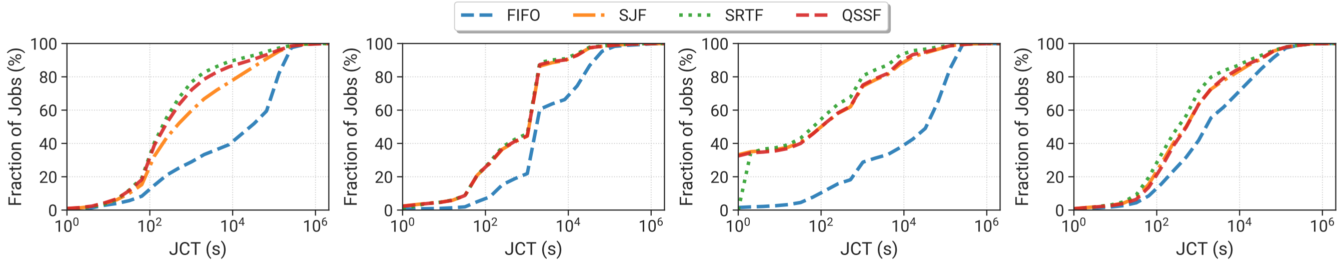
Algorithm 1 Quasi-Shortest-Service-First Scheduler

```
Input: New job: $\mathcal{J}$, VCQueue: $\mathbb{Q}$, Historical job trace: $\mathbb{J}$
 1: procedure QSSF Schedule($\mathcal{J}$, $\mathbb{Q}$, $\mathbb{J}$)
 2:     $\mathcal{J}.priority$ = Priority($\mathcal{J}$, $\mathbb{J}$)          ▷Assign priorty to the new job
 3:     Enqueue $\mathcal{J}$ to $\mathbb{Q}$
 4:     SortJobPriority($\mathbb{Q}$)                          ▷Sort by job priority
 5:     for all $Job \in \mathbb{Q}$ do
 6:         if Consolidate($Job$) is True then
 7:             ConsolidateAllocate($Job$)                 ▷Job placement
 8:             Dequeue $Job$ from $\mathbb{Q}$
 9:         else
10:             break
11:
12: function Priority($\mathcal{J}$, $\mathbb{J}$)
13:     if UserMatch($\mathcal{J}.user$, $\mathbb{J}.users$) is $\varnothing$ then
14:         $\mathcal{P}_R$ = Distribution($\mathbb{J}.durations$)            ▷New user
15:     else if SimilarName($\mathcal{J}.name$, $\mathbb{J}.user.names$) is $\varnothing$ then
16:         $\mathcal{P}_R$ = Distribution($\mathbb{J}.user.durations$)       ▷New job name
17:     else
18:         $\mathcal{P}_R$ = RollingEstimator($\mathcal{J}$, $\mathbb{J}$)
19:     $\mathcal{P}_M$ = MLEstimator($\mathcal{J}$)
20:     $\mathcal{P} = N(\lambda\mathcal{P}_R + (1-\lambda)\mathcal{P}_M)$     ▷$N$: GPU number of $\mathcal{J}$, $\lambda$: Merging
    coefficient
21:     return $\mathcal{P}$
```

4.2.2 Service Design. Our QSSF scheduler builds a machine learning model to predict the expected GPU time of incoming jobs. When a user submits a DL job to the cluster, the scheduler immediately retrieves the relevant attributes (e.g., GPU & CPU demands, job name, user id, target VC, etc) and infers the job's expected GPU time from the model. Then the scheduler selects and allocates jobs based on the predicted GPU time and cluster resource states. After the job termination, the job's final information will be collected by the *Model Update Engine* for fine-tuning the prediction model.

We train a Gradient Boosting Decision Tree (GBDT) [42] model to capture the overall trend of the relevant jobs. Specifically, we extract all features and actual duration from the traces to construct a training and validation set. We encode all the category features (e.g., user name, VC name, job name). For the extremely sparse and high-dimensional features of job names, we utilize the Levenshtein distance [53] to cluster the names and bucketize similar ones, which convert them into relatively dense numerical values. For the time-related features (e.g., job submission time), we parse them into several time attributes, such as month, day of the week, hour, minute. Finally, we train a GBDT model that can map these job attributes to the corresponding duration.

Algorithm 1 shows the pseudo-code of our QSSF algorithm. The core function is Priority (line 12), which returns a priority value ($\mathcal{P}$) for a given job ($\mathcal{J}$). $\mathcal{P}$ is calculated as the weighted sum of a rolling estimate $\mathcal{P}_R$ and machine learning estimate $\mathcal{P}_M$. The rolling estimate $\mathcal{P}_R$ is computed directly from the historical jobs with similar attributes. There are three cases for calculating $\mathcal{P}_R$: (1) if the job user cannot be found in the traces (new user), then $\mathcal{P}_R$ is the average duration of all the jobs with the same GPU demands in the traces (line 14). (2) If the traces have the records of this user, then we



(a) Venus (b) Earth (c) Saturn (d) Uranus

Figure 11: Comparisons of JCT distributions using different scheduling algorithms, based on the September job traces across 4 clusters in `Helios`. Note that SJF and SRTF are optimal baselines (with perfect job duration information).

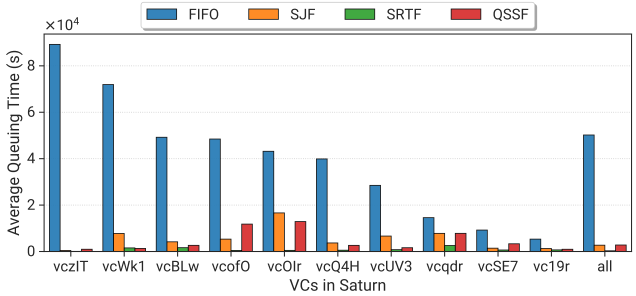


Figure 12: The average job queuing delay of the top 10 VCs in Saturn (September) with different scheduling algorithms. The column `all` represents the whole cluster.

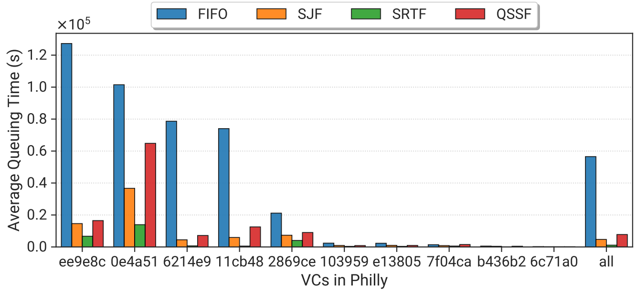


Figure 13: The average job queuing delay of the top 10 VCs in Philly (October and November) with different scheduling algorithms. The column `all` represents the whole cluster.

leverage the Levenshtein distance to find historical jobs from this user, which have similar names or formats as the incoming one. If no such historical jobs are found, then $\mathcal{P}_R$ is the average duration of all this user's jobs with the same GPU demands in the traces (line 16). (3) Otherwise, we compute $\mathcal{P}_R$ via exponentially weighted decay of duration of historical jobs with matched names (line 18). The machine learning estimate $\mathcal{P}_M$ is computed by the GBDT model (line 19), which considers the overall trend of the relevant jobs. Finally, we combine the two estimates $\mathcal{P}_R$ and $\mathcal{P}_M$, and multiply the requested GPU number ($\mathcal{N}$) to get the job's expected GPU time $\mathcal{P}$ as the priority value (line 20), which can reflect both spatial and temporal aspects of the job [31].

We select a job from the VC queue with the highest priority (lowest predicted GPU time) for scheduling. For job placement, unless the topology is specified by the user, the *ConsolidateAllocate* policy is adopted to allocate each job on as few nodes as possible to reduce the communication overhead. For instance, a 16-GPU job needs to wait for two compute nodes with 8 idle GPUs. Other placement solutions will violate the consolidation principle.

4.2.3 Evaluation. We develop a trace-driven simulator to evaluate our QSSF algorithm. It emulates a datacenter with the same configuration as `Helios` in Table 1, which operates with the real-world job workflow: job arrival – queuing – running – completion/canceled/failed. Since the GPU resources are the bottleneck in our clusters, we mainly consider the GPU jobs in our simulation. We train the GBDT model using the jobs from April to August in the traces, and evaluate the model with the jobs in September.

We consider three baseline algorithms for comparisons. (1) *FIFO*: the current scheduling policy in our clusters, which is simple but has poor performance. (2) *SJF*: the optimal policy to minimize the average JCT without preemption. (3) *SRTF*: the optimal preemption-enabled version of *SJF*, where we assume all the jobs can be preempted with negligible overhead. Both the SJF and SRTF algorithms are not practical, since they need perfect job duration information [31, 63], which cannot be obtained in reality. They serve as the upper bound of the scheduling performance. We assume the scheduler knows the exact job duration given in the trace. Also, we do not consider the backfill mechanism, as we want to explore how much benefit can be obtained solely from prediction. Integration of backfill with our QSSF service will be considered as future work.

Figure 11 shows CDF curves of JCT in each cluster with different scheduling algorithms. We observe that our QSSF algorithm can significantly outperform the naive FIFO policy, and perform comparably with SRTF and SJF, without making unrealistic assumptions. The advantage of QSSF over FIFO is relatively smaller in `Uranus`, as the job queuing delay in this cluster is not as severe as the other three clusters: the queuing period takes 70~90% of the total job completion time in other clusters while 42% in `Uranus` (Table 3).

From Table 3, QSSF reduces 37~82% of queued jobs, which is even better than SJF. Interestingly, even though QSSF prioritizes short-term jobs to alleviate the head-blocking problem, the queuing delays of large jobs are also improved because of fewer queuing jobs. Table 4 shows the improvement of QSSF over SJF in different groups of jobs. Short-term jobs achieve at least 9.2× improvement while long-term jobs can also obtain 2.0~4.8× improvement in `Helios`.

Table 3: Performance comparison of different schedulers.

		Venus	Earth	Saturn	Uranus	Philly
Average JCT (s)	FIFO	64,702	19,754	55,984	19,758	86,072
	SJF	21,095	6,892	8,501	13,226	34,272
	QSSF	18,349	6,732	8,581	13,123	37,324
Average Queuing Time (s)	FIFO	52,933	13,699	50,202	8,394	56,531
	SJF	9,325	837	2,719	1,861	4,731
	QSSF	6,580	677	2,798	1,759	7,783
# of Queuing Jobs	FIFO	15,336	30,030	65,991	16,917	30,282
	SJF	8,353	8,848	21,808	11,320	21,437
	QSSF	3,713	5,462	15,311	10,581	22,026

Table 4: The ratio of queuing delay between FIFO and QSSF in different job groups. A higher ratio indicates shorter delay and better efficiency in QSSF.

	Venus	Earth	Saturn	Uranus	Philly
short-term (<15 mins)	11.44	33.51	22.88	9.24	26.95
middle-term (15 mins~6 hours)	4.13	13.39	7.39	2.49	5.54
long-term (>6 hours)	3.22	4.77	3.56	2.00	1.70

This justifies that QSSF will not sacrifice the interest of long-term jobs and all kinds of jobs can benefit from our service.

To evaluate the performance differences caused by the scheduler in each VC, we depict the average job queuing delay of VCs in Saturn, as shown in Figure 12. We select the top-10 VCs with the highest average queuing time, while the other VCs have little delay. We observe the average queuing time of QSSF is almost identical to SJF, and remain stable in each VC. Furthermore, QSSF even outperforms SRTF in *vcWk1*. This confirms the importance of GPU demands to enhance the scheduling efficiency, since large-size short-term jobs may block many small-size jobs. To summarize, compared with FIFO, QSSF achieves 1.5~6.5× improvement in average JCT, and 4.8~20.2× improvement in average queuing delay among 4 clusters in Helios.

In addition, we also evaluate the applicability of our QSSF service on the Philly trace. Since Microsoft did not release sufficient trace information (e.g., job names and VC configurations) which is necessary for our service, we make two reasonable assumptions. (1). The VC configurations are static during our evaluation period (from 1st October to 30th November, 2017) and the size of each VC is set corresponding to its workloads. (2). The priority values are generated randomly with a similar error distribution as Helios estimation. As shown in Table 3, QSSF obtains comparable performance with the optimal SJF in Philly, even without precise estimates. It achieves 2.3× improvement in average JCT, 7.3× improvement in average queuing delay, and reduces 27% of queued jobs. Figure 13 presents the VC-level analysis of QSSF performance. We observe QSSF brings large improvement in each VC with regard to the average queuing time.

4.3 Cluster Energy Saving

4.3.1 Motivation. Electricity dominates the operation cost of modern GPU datacenters, even surpassing the manufacturing cost during the datacenters' lifetime [47]. How to reduce the energy consumption in GPU clusters becomes an important research direction. In reality, tremendous energy is wasted on idle compute nodes.

Algorithm 2 Cluster Energy Saving Node Control

```
Input: Nodes #: Current Active $\mathbf{C_A}$, Current Running $\mathbf{C_R}$, Request $\mathbf{R}$, History Series $\mathcal{H}$, Prediction Series $\mathcal{P}$
procedure JobArrivalCheck($\mathbf{C_A}$, $\mathbf{R}$)
    if $\mathbf{C_A} < \mathbf{R}$ then                                   ▷Lack nodes
        NodesWakeUp($\mathbf{R} - \mathbf{C_A} + \sigma$)                 ▷$\sigma$: Buffer nodes

procedure PeriodicCheck($\mathcal{H}$, $\mathbf{C_A}$, $\mathbf{C_R}$, $\mathcal{P}$)
    $\mathcal{T}_H$ = RecentNodesTrend($\mathcal{H}$)
    $\mathcal{T}_P$ = FutureNodesTrend($\mathcal{P}$)
    if $\mathcal{T}_H \geq \xi_H$ and $\mathcal{T}_P \geq \xi_P$ then     ▷$\xi_H, \xi_P$: Thresholds
        $\mathbf{C_A} \leftarrow$ DynamicResourceSleep($\mathbf{C_R} + \sigma$ )
```

Then the critical goal is to reduce energy consumption from those idle nodes while satisfying users' demands. There are generally two techniques to conserve energy in datacenters. (1) Dynamic Voltage and Frequency Scaling (DVFS) adjusts the CPU & GPU voltage and frequency to save power. (2) Dynamic Resource Sleep (DRS) puts idle servers into deep sleep states (or simply turning them off) [47, 50]. *Slurm* also provides an integrated power saving mechanism [79] to switch the nodes between power saving and normal operation modes based on their workloads.

We introduce a novel Cluster Energy Saving (CES) service to achieve energy conservation in GPU datacenters. Existing DRS-based technique [47] simply turns off and on the nodes based on *recent* and *current* workloads. However, frequent server boot-up can introduce extra energy overhead and delay. In order to reduce the unnecessary mode switch operations, we build a prediction model to estimate the *future* utilization trend in our clusters based on the historical node state logs. With the prediction, we can select and power off the optimal number of servers. This saves energy consumption while maintaining the cluster usability. As described in §3.1, the average utilization rate of our clusters ranges from 65% to 90%, and partial VCs are underutilized all the time. Hence, this strategy can bring huge financial gains for the datacenter.

4.3.2 Service Design. The core of our service is the prediction model, which performs a time-series forecasting task. The biggest challenge of building this model is the lack of accurate information about the future behaviors of the cluster. Therefore, we extract time-related data such as trend, seasonality, and any deterministic event as the features for prediction. Specifically, we encode repetitive patterns (e.g., hour, day of the week, date) of running nodes to explore the periodic variations. Node trends are calculated as the average values and standard deviations of active nodes under different rolling window sizes. Moreover, binary holiday indicators and various time scale lags are also crucial for prediction. We use these features to build a model which can predict the number of running nodes in the future. We try different machine learning algorithms, and find the GBDT [42] model performs the best over other classical or deep learning models, e.g., ARIMA [32], Prophet [67], and LSTM [11]. So we choose GBDT in our CES service, which can achieve around 3.6% error rate (measured in Symmetric Mean Absolute Percentage Error (SMAPE) [35]) in the Earth cluster. This can give reliable and accurate advice for powering off nodes.

With the prediction results, we can select the idle nodes and apply DRS to them. Algorithm 2 illustrates the procedure, where

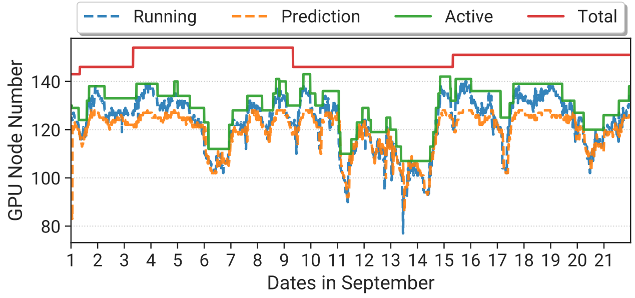


Figure 14: Numbers of compute nodes at different states in Earth from 1st September to 21st September (3 weeks).

two steps are conducted to guarantee compute resources and conserve energy. (1) When a new job is submitted to the cluster, the service performs JobArrivalCheck (line 1) to inspect whether the requested resources ($\mathbf{R}$) surpass the currently available ones ($\mathbf{C_A}$) in the cluster. If so, the service needs to wake up some nodes immediately via the Intelligent Platform Management Interface (IPMI). The quantity of nodes is determined by the resource gap ($\mathbf{R} - \mathbf{C_A}$) with an additional number σ to buffer unexpected future jobs. (2) Our service also calls PeriodicCheck regularly (e.g., every 10 minutes) to check if extra nodes need to be powered off. The decision is made from both the historical and predicted future trends, to circumvent the incorrect DRS operations caused by the prediction error. Specifically, we call the function *RecentNodesTrend* to calculate the reduced number of active nodes during a fixed past period (e.g., one hour) from the historical data. We also call *FutureNodesTrend* to calculate the expected reduced number of active nodes in a fixed future period (typically 3 hours) based on the GBDT model prediction. If these two trends are larger than the thresholds (ξ_H, ξ_P), *DynamicResourceSleep* is called to reduce the active nodes ($\mathbf{C_A}$) to the current running number ($\mathbf{C_R}$) with a buffer σ.

In this paper, we exploit the DRS technique on the selected nodes. Alternatively, we can also utilize *CPUfreq governor* and *nvidia-smi* [6] to adjust the frequency and voltage of CPUs & NVIDIA GPUs. According to [66], DVFS can not only improve the DL training performance by up to 33% but also save up to 23% energy consumption. Evaluations of these techniques will be our future work.

4.3.3 Evaluation. We perform similar simulations as §4.2.3 based on the real-world traces from Helios. We select a period of 3 weeks (from 1st September to 21st September) in each cluster for evaluation, and the previous records are all used for training the prediction model. Figure 14 shows the GPU node behaviors in the Earth cluster. Two dashed lines denote the numbers of actual active nodes (blue) and our predictions (orange) respectively. We observe that our prediction can precisely reflect the actual trend with small estimate errors. The red solid line depicts the total number of GPU nodes. It is obvious to see a huge gap between the red and blue lines, representing a large number of energy-wasting idle nodes. With our CES service, a lot of idle nodes are powered off. The green solid line denotes the number of remaining active nodes. Qualitatively, we observe these nodes are just enough to meet users' demands while significantly reducing the energy waste.

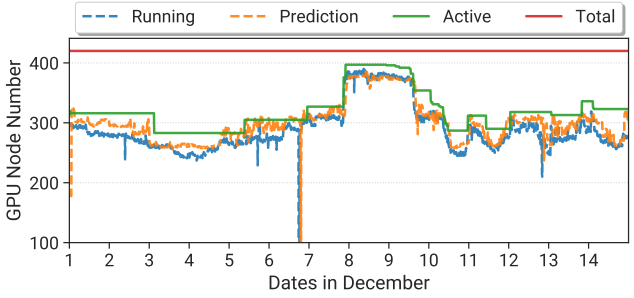


Figure 15: Numbers of compute nodes at different states in Philly from 1st December to 14th December (2 weeks).

Table 5: Performance of CES service in each cluster of Helios and Philly.

	Venus	Earth	Saturn	Uranus	Philly
Average # of DRS nodes	5.0	20.5	20.0	34.0	100.1
Average times of daily wake up	1.1	1.3	2.0	2.6	0.5
Average # of woken up nodes per time	6.7	6.7	7.1	9.7	27.1
Node utilization (Original)	92.7%	82.1%	90.2%	83.8%	69.0%
Node utilization (CES)	96.2%	95.1%	97.6%	96.4%	90.4%

Table 5 presents a quantitative analysis for each cluster. Our service can remarkably improve the node utilization, especially in Earth (13.0%) and Uranus (12.6%), as these clusters are relatively underutilized. Besides, our service only calls the function *NodesWakeUp* 1.1~2.6 times a day in each cluster. In contrast, the vanilla DRS without considering nodes' future trends can incur an average of 34.1 *NodesWakeUp* operations a day, which causes much more turn-on energy overhead and job queue delay. Specifically, assuming each node takes 5 minutes to reboot, our service only affects 251 out of 198k jobs during 21 days. Nevertheless, vanilla DRS leads to nearly 6k jobs affected in the same traces.

Furthermore, we make a rough estimation about the reduction of energy consumption with CES using the data from Table 5. The power consumption of one single idle DGX-1 server is around 800 watts (obtained from NVIDIA BMC [1] by adding the input values of all PSUs). In addition, the cooling infrastructure typically consumes twice the energy as the servers in datacenter [23]. Hence, we can save over 1.65 million kilowatt hours of electricity annually across these 4 clusters. This can significantly reduce the operation cost.

We also evaluate our CSE service on the Philly trace. Microsoft provides per-minute statistics about GPU utilization on every node from the Ganglia monitoring system [39], from which we obtain a time sequence about the numbers of total and running GPU nodes. We select a period of 2 weeks (from 1st December to 14th December) for evaluation, and the previous data are all used for training the GBDT model. As shown in Figure 15, it is obvious that the change frequency of running nodes in Philly is lower than Earth and its cluster scale is over twice than Earth. Hence, the CES service only needs to take 0.5 times of nodes wake up action on average (Table 5), which has a negligible impact on the average JCT. Besides, more than 100 idle nodes can be powered off on average and the cluster node utilization rate is increased from 69% to 90% (Table 5). This shows the CES service has strong applicability and generality for different clusters and workloads.

5 RELATED WORK

Cluster characterization. A number of prior works conducted trace analysis for traditional CPU workloads, e.g., HPC systems [9, 10, 40, 55, 60, 62, 76], private clouds [20, 59, 61, 68]. Amvrosiadis et al. [9] presented an analysis of the private and HPC cluster traces about job characteristics, workload heterogeneity, and cluster utilization. They disclosed the correlations among job duration, behaviors, logical names and user behaviors. We further demonstrate the predictability of cluster resources and workloads information in GPU datacenters.

On the other hand, fewer studies focused on the analysis of characteristics of DL workloads in large-scale clusters. Jeon et al. [39] studied the DL job trace from the Microsoft cluster to identify the impact of the DL training job locality on the GPU utilization as well as job failure reasons. Wang et al. [73] characterized various resource requirements and identified the performance bottlenecks of DL training jobs in Alibaba datacenter.

In comparison, our work provides a more comprehensive analysis about clusters, jobs, and user behaviors. Besides, wec provide more diversity to reflect the latest features of DL algorithms and technologies. These can deepen our understanding about the characteristics of DL clusters and jobs. We uncover seven interesting observations to inspire the new designs of efficient GPU schedulers.

Prediction-based scheduling. Prior knowledge of jobs can significantly facilitate the management of cluster resources. Modern cluster systems [16, 33, 70] expect users to provide estimates about their job duration, which may not be accurate as the job completion time is unexpected. Several methods were proposed to obtain more precise job information automatically for efficient scheduling. For instance, some schedulers predict the duration of new jobs based on the recurrent jobs [22, 24, 38, 41], or job structure knowledge [13, 27, 37, 71, 72]. These require the jobs to have explicit periodic patterns or known structures. For more general cases, some systems [20, 54, 69] predict the estimate from the history of relevant jobs.

For DL training job schedulers, Gandiva [77] leverages intra-job predictability to split GPU-time efficiently across multiple jobs to achieve lower latency. Optimus [56] adopts online fitting to predict model convergence during training. Tiresias [31] calculates the Gittins index as the job priority according to the duration distribution of previous jobs. AlloX [43] designs an estimator and profiles jobs in an online manner to determine the job duration. Different from these schedulers only for DL training, our QSSF scheduling service supports all types of DL jobs in the model developing pipeline. This better fits the industry requirements for production GPU clusters.

Energy efficiency for GPU clusters. Prior studies [36, 50] demonstrated that DRS does not affect the performance and efficiency of the cluster, which is consistent with our simulation results. Furthermore, a series of works [8, 14, 28, 45, 48, 66, 74] indicated that GPU DVFS has huge benefits to save the energy consumption in GPU clusters. Some job scheduling algorithms [17, 47, 49] were then proposed based on these techniques to achieve energy efficiency in the clusters. However, they do not consider the impact of cluster future workloads, which can lead to more unnecessary server boot-up delays and extra energy overhead. In our CES service, we further enhance the benefits of the DRS by predicting the cluster usage in advance, which can get better efficiency.

6 DISCUSSIONS

6.1 Extension to Small-scale Clusters.

In this paper, we mainly conduct the analysis and characterization of large-scale GPU clusters. Our framework can also be applied to small clusters, for two reasons. (1) In a small cluster, the number of active jobs is also scaled down with the available resources. This does not affect the inherent behaviors and characteristics of jobs and users. (2) We focus on the analysis of VCs (with tens to hundreds of GPUs), which are independent without sharing. They can be regarded as small clusters.

Specifically for our prediction framework, the prediction accuracy does not depend on the cluster size. It is mainly determined by two factors: (1) the size of the training set: more historical data can bring more comprehensive and generalized information, which leads to higher model accuracy; (2) the characteristics of the cluster. QSSF relies on the diverse attributes and distributions of jobs, while CES relies on the seasonal utilization trends of the cluster. So we believe our system can be deployed to small clusters similarly.

6.2 Future Works

We identify the following several directions as future work. (1) We aim to design and integrate more services into our framework to make it more comprehensive. (2) Some attributes in our services may not be available in other clusters. We aim to design new qualified models with limited job information for our services. (3) Our services mainly rely on historical job data, and coarse-grained cluster information. We aim to collect and leverage more fine-grained resource information (e.g., GPU memory usage and computation unit utilization, CPU utilization) as features to build more accurate models for better cluster management performance. (4) We are planning to implement our prediction framework in our production clusters, and evaluate its effectiveness at scale.

7 CONCLUSION

In this paper, we perform a large-scale analysis of the real-world DL job traces from four clusters in our datacenter. We present the characterizations of clusters, jobs and users, and identify seven implications, to guide us to design more efficient GPU cluster systems. Justified by the implication that the behaviors of jobs and clusters are predictable, we introduce a general-purpose GPU cluster management framework, which predicts the future behaviors of jobs and clusters to improve the resource utilization and job performance. As two case studies, we design a QSSF service to improve the average JCT by up to 6.5×, and a CES service to conserve annual power consumption of over 1.65 million kilowatt hours.

`Helios` traces are publicly available at https://github.com/S-Lab-System-Group/HeliosData. We expect they can benefit researchers in the design of GPU datacenter systems.

ACKNOWLEDGMENTS

We thank the 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] 2021. DGX-1 BMC. https://docs.nvidia.com/dgx/dgx1-user-guide.

[2] 2021. Lustre. https://www.lustre.org/.

[3] 2021. Memcached. https://memcached.org/.

[4] 2021. NCCL. https://developer.nvidia.com/nccl.

[5] 2021. NVIDIA Multi-Instance GPU. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/.

[6] 2021. NVIDIA-smi. https://developer.nvidia.com/nvidia-system-management-interface.

[7] 2021. NVLink. https://www.nvidia.com/en-us/data-center/nvlink/.

[8] Yuki Abe, Hiroshi Sasaki, Shinpei Kato, Koji Inoue, Masato Edahiro, and Martin Peres. 2014. Power and Performance Characterization and Modeling of GPU-Accelerated Systems. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS '14)*.

[9] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman, and Nathan DeBardeleben. 2018. On the diversity of cluster workloads and its impact on research results. In *2018 USENIX Annual Technical Conference (USENIX ATC '18)*.

[10] Norbert Attig, Paul Gibbon, and Thomas Lippert. 2011. Trends in supercomputing: The European path to exascale. *Computer Physics Communications* 182 (2011), 2041–2046.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations (ICLR '15)*.

[12] Marcel Blöcher, Lin Wang, Patrick Eugster, and Max Schmidt. 2021. Switches for HIRE: Resource Scheduling for Data Center in-Network Computing. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*.

[13] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. 2014. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*.

[14] Robert A. Bridges, Neena Imam, and Tiffany M. Mintz. 2016. Understanding GPU Power: A Survey of Profiling, Modeling, and Simulation Methods. *Comput. Surveys* 49 (2016).

[15] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS '20)*.

[16] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems over a Decade. *Queue* 14 (2016), 70–93.

[17] Vincent Chau, Xiaowen Chu, Hai Liu, and Yiu-Wing Leung. 2017. Energy Efficient Job Scheduling with DVFS for CPU-GPU Heterogeneous Systems. In *Proceedings of the Eighth International Conference on Future Energy Systems (e-Energy '17)*.

[18] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. 2020. Balancing Efficiency and Fairness in Heterogeneous GPU Clusters for Deep Learning. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*.

[19] Yanpei Chen, Sara Alspaugh, and Randy Katz. 2012. Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. *Proceedings of the VLDB Endowment* 5 (2012), 1802–1813.

[20] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*.

[21] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*.

[22] 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 *Proceedings of the ACM Symposium on Cloud Computing (SoCC '14)*.

[23] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. 2016. Data center energy consumption modeling: A survey. *IEEE Communications Surveys and Tutorials* 18 (2016), 732–794.

[24] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. 2015. Hawk: Hybrid Datacenter Scheduling. In *2015 USENIX Annual Technical Conference (USENIX ATC '15)*.

[25] William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *CoRR* abs/2101.03961 (2021).

[26] Dror G. Feitelson. 1996. Packing schemes for gang scheduling. In *Job Scheduling Strategies for Parallel Processing*.

[27] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. 2012. Jockey: Guaranteed Job Latency in Data Parallel Clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*.

[28] Rong Ge, Ryan Vogt, Jahangir Majumder, Arif Alam, Martin Burtscher, and Ziliang Zong. 2013. Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU. In *42nd International Conference on Parallel Processing (ICPP '13)*.

[29] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. 2016. Altruistic Scheduling in Multi-Resource Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.

[30] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. 2016. GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.

[31] 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 *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19)*.

[32] James Douglas Hamilton. 2020. *Time Series Analysis*. Princeton University Press.

[33] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)*.

[34] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. 2021. Elastic Resource Sharing for Distributed Deep Learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21)*.

[35] Rob J. Hyndman and Anne B. Koehler. 2006. Another look at measures of forecast accuracy. *International Journal of Forecasting* 22 (2006), 679–688.

[36] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. 2007. Algorithms for Power Savings. *ACM Transactions on Algorithms* 3 (2007), 41–es.

[37] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '07)*.

[38] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. 2015. Network-Aware Scheduling for Data-Parallel Jobs: Plan When You Can. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*.

[39] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC '19)*.

[40] Wayne Joubert and Shi-Quan Su. 2012. An Analysis of Computational Workloads for the ORNL Jaguar System. In *Proceedings of the 26th ACM International Conference on Supercomputing (ICS '12)*.

[41] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, and Sriram Rao. 2016. Morpheus: Towards Automated SLOs for Enterprise Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.

[42] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems (NeurIPS '17)*.

[43] 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 (EuroSys '20)*.

[44] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI '14)*.

[45] Wenjie Liu, Zhihui Du, Yu Xiao, David A. Bader, and Chen Xu. 2011. A Waterfall Model to Achieve Energy Efficient Tasks Mapping for Large Scale GPU Clusters. In *25th IEEE International Symposium on Parallel and Distributed Processing, Workshop Proceedings (IPDPS '11)*.

[46] 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 (NSDI '20)*.

[47] Xinxin Mei, Xiaowen Chu, Hai Liu, Yiu-Wing Leung, and Zongpeng Li. 2017. Energy efficient real-time task scheduling on CPU-GPU hybrid clusters. In *IEEE Conference on Computer Communications (INFOCOM '17)*.

[48] Xinxin Mei, Qiang Wang, and Xiaowen Chu. 2017. A survey and measurement study of GPU DVFS on energy conservation. *Digital Communications and Networks* 3 (2017), 89–100.

[49] Xinxin Mei, Qiang Wang, Xiaowen Chu, Hai Liu, Yiu-Wing Leung, and Zongpeng Li. 2021. Energy-aware Task Scheduling with Deadline Constraint in DVFS-enabled Heterogeneous Clusters. *CoRR* abs/2104.00486 (2021).

[50] David Meisner, Brian T. Gold, and Thomas F. Wenisch. 2009. PowerNap: Eliminating Server Idle Power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '09)*.

[51] Richard L. Moore, Adam Jundt, Leonard K. Carson, Kenneth Yoshimoto, Amin Ghadersohi, and William S. Young. 2012. Analyzing Throughput and Utilization on Trestles. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the EXtreme to the Campus and Beyond (XSEDE '12)*.

[52] 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 (OSDI '20)*.

[53] Gonzalo Navarro. 2001. A Guided Tour to Approximate String Matching. *Comput. Surveys* 33 (2001), 31–88.

[54] 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 (EuroSys '18)*.

[55] Tirthak Patel, Zhengchun Liu, Raj Kettimuthu, Paul Rich, William Allcock, and Devesh Tiwari. 2020. Job Characteristics on Large-Scale Systems: Long-Term Analysis, Quantification, and Implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20)*.

[56] 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 (EuroSys '18)*.

[57] Lutz Prechelt. 1998. *Early Stopping - But When?* Springer Berlin Heidelberg, 55–69.

[58] 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 *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI '21)*.

[59] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '12)*.

[60] Gonzalo P. Rodrigo, P.-O. Östberg, Erik Elmroth, Katie Antypas, Richard Gerber, and Lavanya Ramakrishnan. 2018. Towards understanding HPC users and systems: A NERSC case study. *J. Parallel and Distrib. Comput.* 111 (2018), 206–221.

[61] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX Annual Technical Conference (USENIX ATC '20)*.

[62] Nikolay A. Simakov, Joseph P. White, Robert L. DeLeon, Steven M. Gallo, Matthew D. Jones, Jeffrey T. Palmer, Benjamin Plessinger, and Thomas R. Furlani. 2018. A Workload Analysis of NSF's Innovative HPC Resources Using XDMoD. *CoRR* abs/1801.04306 (2018).

[63] 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 *49th International Conference on Parallel Processing (ICPP '20)*.

[64] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NeurIPS '14)*.

[65] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '14)*.

[66] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. 2019. The Impact of GPU DVFS on the Energy and Performance of Deep Learning: An Empirical Study. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems (e-Energy '19)*.

[67] Sean J. Taylor and Benjamin Letham. 2018. Forecasting at Scale. *The American Statistician* 72 (2018), 37–45.

[68] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. 2020. Borg: the Next Generation. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*.

[69] Alexey Tumanov, Angela Jiang, Jun Woo Park, Michael A. Kozuch, and Gregory R. Ganger. 2016. *JamaisVu: Robust Scheduling with Auto-Estimated Job Runtimes*. Technical Report. Carnegie Mellon University.

[70] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing (SoCC '13)*.

[71] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)*.

[72] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. 2011. ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC '11)*.

[73] Mengdi Wang, Chen Meng, Guoping Long, Chuan Wu, Jun Yang, Wei Lin, and Yangqing Jia. 2019. Characterizing Deep Learning Training Workloads on Alibaba-PAI. In *Proceedings of the 2019 IEEE International Symposium on Workload Characterization (IISWC '19)*.

[74] Qiang Wang and Xiaowen Chu. 2020. GPGPU Performance Estimation With Core and Memory Frequency Scaling. *IEEE Transactions on Parallel and Distributed Systems* 31 (2020), 2865–2881.

[75] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. 2006. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)*.

[76] Nicole Wolter, Michael O Mccracken, Allan Snavely, Lorin Hochstein, Taiga Nakamura, and Victor Basili. 2006. What ' s working in HPC : Investigating HPC User Behavior and Productivity. *CTWatch Quarterly* 2 (2006), 1–14.

[77] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*.

[78] 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 *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*.

[79] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*.

[80] Anas A. Youssef and Diwakar Krishnamurthy. 2017. Burstiness-aware service level planning for enterprise application clouds. *Journal of Cloud Computing* 6 (2017), 1–21.

[81] Peifeng Yu and Mosharaf Chowdhury. 2020. Fine-Grained GPU Sharing Primitives for Deep Learning Applications. In *Proceedings of Machine Learning and Systems (MLSys '20)*.

[82] Ru Zhang, Wencong Xiao, Hongyu Zhang, Yu Liu, Haoxiang Lin, and Mao Yang. 2020. An Empirical Study on Program Failures of Deep Learning Jobs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*.

[83] Sheng Zhang, Zhuzhong Qian, Zhaoyi Luo, Jie Wu, and Sanglu Lu. 2016. Burstiness-Aware Resource Reservation for Server Consolidation in Computing Clouds. *IEEE Transactions on Parallel and Distributed Systems* 27 (2016), 964–977.

[84] 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 *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

GPU Datacenter Description

Helios is a private datacenter dedicated to developing DL models for research and production in SenseTime. It contains thousands of compute nodes, and each node is equipped with 8 GPUs.

In this paper, we select 4 representative multi-tenant clusters from Helios, including Venus, Earth, Saturn, and Uranus. These four clusters consist of 802 compute nodes with 6416 GPUs. Saturn is a heterogeneous cluster with mixed NVIDIA Pascal and Volta GPUs, while the other three clusters are composed of identical Volta or Pascal GPUs. The inter-node communication within the same RDMA domain is achieved via the high-speed InfiniBand.

Analysis Platform Description

We collect deep learning jobs from each of the 4 clusters in Helios, which serves as the basis of our analysis and system design in this paper. These four traces span 6 months from April 2020 to September 2020, covering a total of 3.36 million jobs. We perform the analysis of these traces on Ubuntu 20.04 using JupyterLab 3.0. Required python libraries for characterization and visualization include pandas 1.2.3, numpy 1.19.2, matplotlib 3.3.4, and seaborn 0.11.1.

Our traces dataset and analysis codes are publicly available.

Author-Created or Modified Artifacts:

```
Persistent ID: https://github.com/S-Lab-System-Group⌋
↪  /HeliosArtifact
Artifact name: HeliosArtifact

Persistent ID:
↪  https://github.com/S-Lab-System-Group/HeliosData
Artifact name: HeliosData
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Datacenter: Each node is equipped with dual-sockets Intel Xeon Gold 6146, eight NVIDIA Tesla V100 SXM2 32GB, 376 GB of memory, and connected through Infiniband EDR in Venus and Earth. Analysis: The workstation is equipped with Intel Core i9-10900, NVIDIA GeForce RTX 2080 Ti and 32 GB of memory.

Operating systems and versions: Datacenter: CentOS 7, Analysis: Ubuntu 20.04

Compilers and versions: Analysis: Python 3.8

Applications and versions: Analysis: JupyterLab 3.0

Libraries and versions: Analysis: pandas 1.2, numpy 1.19, matplotlib 3.3, seaborn 0.11, lightgbm 3.1

Key algorithms: Gradient Boosting Decision Tree

Input datasets and versions: Helios trace, Philly trace

URL to output from scripts that gathers execution environment information.

```
https://github.com/S-Lab-System-Group/HeliosArtifact⌋
↪  /tree/master/enviornment
```