Energy-Efficient GPU Clusters Scheduling for Deep Learning

Diandian Gu
Peking University

Xintong Xie
Peking Univerisity

Gang Huang
Peking Univerisity

Xin Jin
Peking Univerisity

Xuanzhe Liu
Peking Univerisity

Abstract

Training deep neural networks (DNNs) is a major workload in datacenters today, resulting in a tremendously fast growth of energy consumption. It is important to reduce the energy consumption while completing the DL training jobs early in data centers. In this paper, we propose PowerFlow, a GPU clusters scheduler that reduces the average Job Completion Time (JCT) under an energy budget. We first present performance models for DL training jobs to predict the throughput and energy consumption performance with different configurations. Based on the performance models, PowerFlow dynamically allocates GPUs and adjusts the GPU-level or job-level configurations of DL training jobs. PowerFlow applies network packing and buddy allocation to job placement, thus avoiding extra energy consumed by cluster fragmentations. Evaluation results show that under the same energy consumption, PowerFlow improves the average JCT by 1.57 - 3.39× at most, compared to competitive baselines.

1 Introduction

Deep learning (DL) powers many applications and services we use every day. Therefore, training deep neural networks (DNNs) has become an important workload in datacenters. To increase the accuracy of DNN models, a substantial volume of training data is required; and the DNN model architectures become more and more complex (e.g., BERT large [20] with 340 million parameters and GPT-3 [7] with 175 billion parameters). It is reported that the computation demand of DL training grows at a speed of 35× every 18 months [1].

However, this growing demand for computation leads to greater energy demand and carbon emissions. For example, the estimated CO_2 emissions of training a Transformer (large) model are 5 times the life cycle CO_2 emissions of the car [3]. Many service providers have made their goals to reduce their carbon footprints [4–6]. As an energy-consuming workload, reducing the total energy consumption of DL jobs is of vital importance to achieving such goals. Meanwhile, for the service providers' revenue, reducing the job completion time (JCT) of DL training jobs as much as possible is also substantial. It is beneficial for service providers to speed up the DL training jobs under a given energy budget, as this is not only energy-saving and environmentally friendly, but also leaves resources to serve more jobs.

As far as we know, the current practice for scheduling DL jobs is neither energy-aware nor energy-efficient. Most schedulers for DL jobs ignore the GPU-level configurations such as *GPU core frequency*, which has a significant impact on the throughput and energy consumption of DL jobs. Many schedulers execute DL jobs with user-defined job-level hyperparameters, which might not be energy-efficient. Also, existing GPU schedulers ignore the energy-consumption characteristics of different DL jobs, leading to a waste of energy in the cluster. Several efforts have studied the impact of GPU Dynamic Frequency and Voltage Scaling (DVFS) and power configuration on the energy consumption and the throughput of DNN training [12, 36, 60]. But directly applying DVFS or power configuration lacks the overall perspective from the whole GPU cluster and cannot leverage elasticity to further optimize the energy consumption and average JCT. There are virtual machine (VM) schedulers to reduce energy consumption in traditional clusters [21, 62]. These methods require fine-grained resource allocation. However, the common practice for DL training jobs dedicates the whole GPU to a single job. The fine-grained methods do not apply to this common practice.

In this paper, we present PowerFlow, an energy-aware scheduler for DL training jobs in GPU clusters. Under a given energy budget (i.e., the total consumed energy does not exceed a certain goal in a period of time), PowerFlow tries to tradeoff between the average JCT and the total energy consumption of the submitted DL training jobs. Compared with existing solutions, PowerFlow holds the view of the whole cluster and adjusts both GPU-level and job-level configurations dynamically to make the whole cluster energy-efficient.

Designing such a solution can be challenging. First, it is difficult to know how throughput and energy consumption

change with different configurations. Existing studies only summarize coarse-grained features of DL jobs' performance: some of them only consider a fixed configuration for a DL job [10], while others do not consider the GPU-level configurations such as GPU frequency [53]. Therefore, existing studies are not sufficient and accurate to predict the performance of DL training jobs whose configurations change dynamically. To accelerate DL jobs under a specified energy budget, we need to know how the throughput and energy consumption change with different configurations. Second, there is a trade-off between the energy consumption and the JCT of a DL job. In a cluster with limited GPU devices, it is difficult to decide whether a DL job should use less energy to achieve the energy budget goal or use more energy for a shorter JCT.

To address these challenges, we model how a DL job's performance (i.e., throughput and energy consumption) changes with different GPU-level and job-level configurations. Then, we propose *energy efficiency* to capture how much throughput improvement can be brought by each unit of energy consumption. Based on the performance model and the energy efficiency metric, we design a greedy algorithm to prioritize resource allocation and GPU frequency raise for the most energy-efficient job without violating a *power limit*. PowerFlow applies network packing and buddy allocation to job placement, avoiding extra energy consumption of cluster fragmentations.

In summary, we make the following contributions:

- We show that a model of a DL job's throughput and energy consumption per iteration can be learned by observing its step time and energy consumption during training and be used for predicting the performance given different configurations.
- We propose a formulation of energy efficiency for DL jobs in a cluster, which is a measure of training performance that takes into account both training throughput and energy consumption.
- We design and implement a scheduling algorithm that uses such performance models and the energy efficiency metric to schedule each submitted DL job.
- We evaluate PowerFlow with simulation experiments using a 1901-job trace. Experiment results show that PowerFlow improves the average JCT by 1.57 - 3.39× at most, compared to competitive baselines. This indicates that PowerFlow is energy efficient.

2 Background

2.1 DNN Training

A DL training job trains a DNN model with a dataset. The job includes many iterations, and each uses a batch of samples from the dataset to train the model with a forward propagation pass and a backward propagation pass. The batch size is a hyperparameter set by DL training developers, which affects not only the model accuracy but also the training throughput and energy consumption. Usually, the DNN models are trained on powerful accelerators such as GPUs.

As DNN models are widely used in various applications and services [11, 17], training DNN models has become an important workload in datacenters. To improve the accuracy of DNN models, the training datasets and the sizes of models are growing larger and larger. Therefore, it requires more and more computation resources for DL training. Since 2012, the amount of computation used in the largest AI training runs has been increasing exponentially with a 3.4-month doubling time [57]. It is time-consuming to train such large DNN models, so distributed training is widely used to speed up the training process. In the data parallelism [37, 39, 40, 49] strategy of distributed training, the batch size of each worker is called local batch size, and the sum of the batch size of all workers is called global batch size. There are also other strategies, such as model parallelism [18], hybrid parallelism [34], and pipeline parallelism [29]. In this paper, we leverage data parallelism with elastic training to adjust the throughput and energy consumption of DL training jobs.

The growth of large training datasets and large DNN models leads to enormous energy consumption. It is reported that most of the energy consumption in data centers is in servers [44]. Recent benchmarks show that among hardware devices such as GPU, CPU, and DRAM, GPUs are responsible for about 70% of the total energy consumption during DNN training [22, 28, 38]. Therefore, we focus on the energy consumed by GPU devices in this work.

2.2 Limitations of Existing Solutions

Some schedulers for VM have been proposed to reduce energy consumption in traditional clusters [21, 62]. These schedulers require fine-grained resource allocation such as allocating the CPU cores to different VMs. However, these efforts cannot be directly applied to a GPU cluster. This is because GPU sharing is still undeveloped: it brings many problems [41] such as performance degrade, memory corruption, error propagation, etc. The common practice for DL training is to dedicate a GPU to a single job.

Considering the characteristics of GPU devices and DL jobs, schedulers for DL training jobs in GPU clusters are proposed [24, 30, 53, 65]. These schedulers often optimize for the goal of minimizing average JCT by adjusting the execution order of training jobs or allocating different numbers of GPUs to each job. However, none of these schedulers consider the total energy consumption of DL training jobs. Zeus [66] navigates the tradeoff between energy consumption and performance optimization for a single DL training job, but lacks the consideration of running multiple jobs from the perspective of

the whole GPU cluster.

3 Motivation

3.1 Opportunities

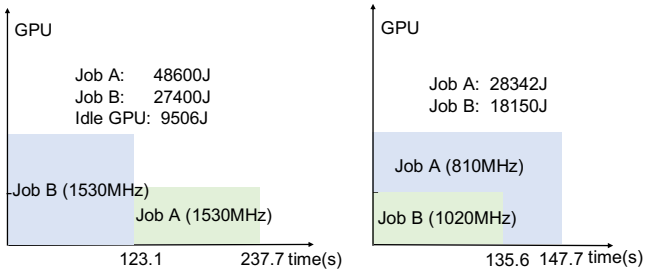
There are opportunities for reducing the average JCT under an energy budget from three aspects: GPU level, job level, and cluster level.

From the perspective of a single GPU, the default GPU core frequency is usually the largest supported frequency, which is not energy-efficient [66]. The frequency of a GPU has a great influence on the power of the GPU, thus influencing the training throughput and energy consumption of a training job. DVFS tunes the frequency and voltage of hardware devices and can be adopted by laptop computers, servers, and mobile devices to conserve energy [45]. The CPU DVFS technology is well-developed and has been adopted in both personal computing devices and large-scale clusters [8]. Despite the maturity of CPU DVFS, the study of GPU DVFS is still at an early stage. Therefore, the GPU cluster scheduler can set the GPUs at a more energy-efficient frequency than the default settings according to the energy budget of the whole cluster.

From the perspective of a DL training job, the job-level hyperparameters may not be energy-efficient. DL training developers are likely to be familiar with DL algorithms (e.g., designing the architecture of DNN models, setting the global batch size, etc.) but have limited expertise in systems (e.g., choosing a local batch size that fits in the GPU memory, setting the number of GPUs for training, etc.). Compared to the hyperparameters set by developers, there might be another set of hyperparameters that achieves a smaller JCT or less energy consumption. For example, for a DL training job that trains the VGG16 model with a global batch size of 64 on NVIDIA A100 GPUs, it takes 0.112 s and 39 J to train one iteration on two GPUs, but it only takes 0.114 s and 27 J on one GPU[1]. Compared to the two-GPU setting, the one-GPU setting consumes 31% less energy with a step time loss of only 2%. Therefore, we can leave the configuration of system-related configurations to the scheduler (i.e., adjust the number of GPUs dynamically and adjust the local batch size accordingly).

From the perspective of a GPU cluster, existing solutions do not consider the energy impact of their scheduling decisions. On one hand, for schedulers that do not leverage the elasticity of DL jobs (e.g., the number of GPUs does not change during job execution) [24, 65], there might be fragmentations in the cluster, and the idle GPUs consume non-negligible energy. On the other hand, the schedulers with elastic resource allocation [30, 53] ignore the different energy consumption patterns of different jobs, thus resulting in energy inefficiency. An energy-efficient scheduler should try to avoid fragmentations in the cluster while being aware of the different characteristics of different jobs.

[1] We measured the GPU energy consumption with NVML [2]



(a) GPU allocation with Tiresias over time. (b) Optimal schedule.

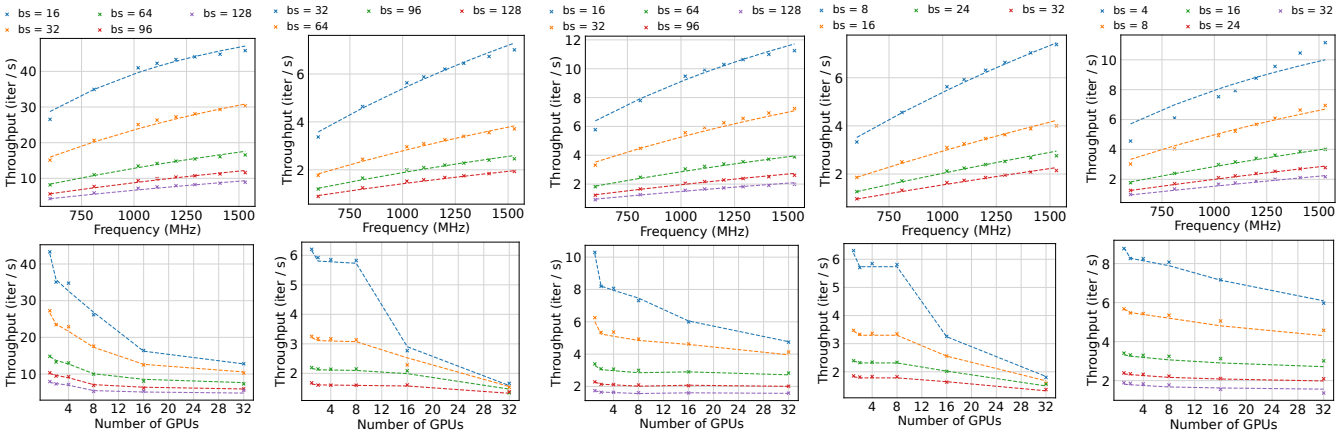
Figure 1: Motivating example to show that existing DL schedulers are not energy-efficient.

Combining the above three opportunities, there is still room for improvement in reducing JCT with an energy consumption budget. Consider two DL training jobs in a cluster with two A100 GPUs. Job A trains a VGG16 model with global batch size 64. The developer sets the number of GPUs to 1. Job B trains a GPT2 model with global batch size 32. The developer sets the number of GPUs to 2. Both jobs are submitted at the same time and are set to train for 1000 iterations. The scheduling result with Tiresias [24] scheduler is shown in Figure 1(a). The average JCT is 180.4 s and the total energy consumption (including the energy consumption of the idle GPU) is 85 546 J. Figure Figure 1(b) shows a scheduling result with smaller JCT and less energy consumption: the JCT is reduced by 21% and the energy consumption is reduced by 46%. The improvement is achieved by adjusting the GPU frequency and the number of GPUs dynamically, and the specific configurations depend on the performance characteristics of the jobs.

4 Performance Modeling

In this section, we illustrate that the performance (including the training throughput and energy consumption per iteration) of DL training jobs can be measured during training and used as predictive models. PowerFlow leverages these performance models to tradeoff between JCT and energy consumption under an energy budget. PowerFlow maintains the same algorithm-related hyperparameters (i.e., global batch size, etc.) and adjusts the other system-related configurations. We focus on three system-related configurations of DL jobs:

- n: the number of GPUs allocated for the DL training job.
- bs: the local (i.e., per-GPU) batch size, which is defined as $bs = BS/n$, where BS is the global batch size specified by the DL developer.
- f: the core frequency of GPUs.



(a) ResNet18 (b) VGG16 (c) Inception V3 (d) GPT2 (e) Deep Speech 2

Figure 2: Throughput of different DNN models with different configurations. The scattered data points are the throughput values measured on real NVIDIA V100 GPUs and the lines show the values predicted by the models. The figures on the top show how throughput changes with GPU frequency on one GPU. The figures at the bottom show how throughput changes with the number of GPUs with 1200 MHz frequency. Note that the local batch size is kept the same and the global batch size scales linearly with the number of GPUs.

4.1 Modeling Throughput

First, we model and predict the throughputs of training a DNN model under different configurations. The throughput is defined as the number of iterations that can be executed in each unit of time:

$$tpt = 1/T_{iter}, \tag{1}$$

where T_{iter} is the step time (i.e., training time per iteration) for a DL training job. DL training requires multiple types of resources such as GPU, CPU, network, etc [68]. Therefore, we separately model T_{IO} for the time in each iteration spent reading the dataset from disk, T_{grad} for the time in each iteration computing gradients locally, and T_{sync} for the time in each iteration synchronizing model parameters between different GPU devices.

Modeling T_{IO}. In DL training, reading training data into workers requires storage IO from local or remote storage. It is common to prefetch the training samples of the next batch while training the current batch (i.e., pipeline storage IO and GPU computation). We model T_{IO} as a linear function of the local batch size. GPUs co-located on the same node might compete for the IO bandwidth of the node. Therefore, we also include a linear factor to model T_{IO} with the number of allocated GPUs on each node. Thus, T_{IO} is modeled as:

$$T_{IO} = \alpha_{IO} + \beta_{IO} * bs * r, \tag{2}$$

where r is the number of allocated GPUs on each node and α_{IO} and β_{IO} are fittable parameters. Note that we assume the number of GPUs on every node is the same.

Modeling T_{grad}. The time spent on computing local gradients with forward and backward propagation is relevant to the GPU core frequency f and the local batch size bs. On one hand, by definition, the amount of computation executed in a certain time slot scales linearly with the device frequency. Therefore, we also model T_{grad} as inversely proportional to f. On the other hand, T_{grad} scales linearly with bs. The final T_{grad} is modeled as:

$$T_{grad} = \alpha_{grad} + (\beta_{grad} + \kappa_{grad}/f) * bs, \tag{3}$$

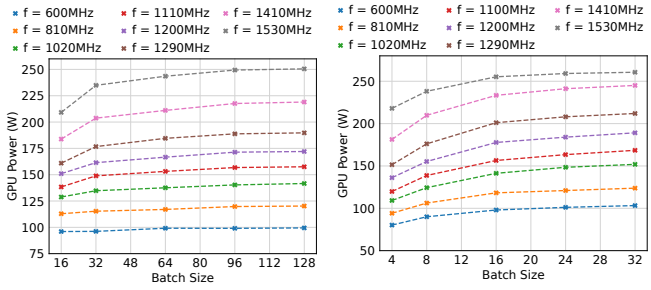
where α_{grad}, β_{grad}, and κ_{grad} are fittable parameters.

Modeling T_{sync}. Following previous work [53], we separately model T_{sync} when the job is allocated on a single GPU (no synchronization is required), on multiple GPUs of the same node (synchronize with PCIe or NVLink), and on multiple nodes (synchronize with Ethernet or InfiniBand). Similar to T_{grad}, we estimate the communication time between GPUs as inversely proportional to f:

$$T_{sync} = \begin{cases} 0 & if\ n = 1 \\ \frac{\alpha_{sync}^{local}}{f} + (\frac{\kappa_{sync}^{local}}{f} + \beta_{sync}^{local}) * (n-2) + \theta_{sync}^{local} & if\ n = r, n \geq 2 \\ \frac{\alpha_{sync}^{node}}{f} + (\frac{\kappa_{sync}^{node}}{f} + \beta_{sync}^{local}) * (n-2) + \theta_{sync}^{node} & otherwise. \end{cases} \tag{4}$$

$n = r$ means only one node is allocated for the job. α_{sync}^{local}, β_{sync}^{local}, θ_{sync}^{local}, and κ_{sync}^{local} are the parameters for the case when the job uses multiple GPUs of the same node. α_{sync}^{node}, β_{sync}^{node}, θ_{sync}^{node}, and κ_{sync}^{node} are the parameters for the case when the job uses multiple nodes.

Combining T_{IO}, T_{grad}, and T_{sync}. It is common for DL frameworks to overlap T_{IO}, T_{grad}, and T_{sync} by pipelining storage IO,



(a) ResNet18. (b) Deep Speech 2.

Figure 3: GPU power of training DNN models on one GPU.

GPU computation, and network IO [35, 46, 49, 52]. We follow previous work [53] to estimate T_{iter} as somewhere between no overlap (i.e., $T_{IO} + T_{grad} + T_{sync}$) and fully overlap (i.e., $max(T_{IO}, T_{grad}, T_{sync})$). Therefore, T_{iter} is modeled as:

$$T_{iter} = ((T_{IO}^{\gamma_1} + T_{grad}^{\gamma_1})^{\gamma_2/\gamma_1} + T_{sync}^{\gamma_2})^{\gamma_2}, \gamma_1 \geq 1, \gamma_2 \geq 1. \quad (5)$$

γ_1 and γ_2 are fittable parameters. When $\gamma_1 = 1$ and $\gamma_2 = 1$, $T_{iter} = T_{IO} + T_{grad} + T_{sync}$ and when $\gamma_1 \to \infty$ and $\gamma_2 \to \infty$, T_{IO}, T_{grad}, and T_{sync} tend to be fully overlapped.

Figure 2 shows how our throughput model is fitted to measured throughput results on different DNN models and different configurations. We measured the throughputs on real NVIDIA V100 GPUs. As is shown in the figures, The fitted models represent the real throughput performance well. We will evaluate the fitted models in detail in Section 6.3.

4.2 Modeling Energy Consumption

The energy consumption of a GPU device can be decomposed into the sum of the dynamic and static components [25]. GPU devices take part in both the computation of gradients and the synchronization between GPUs, which are dynamic processes. Therefore, we model E_{iter} as:

$$E_{iter} = E_{grad} + E_{sync} + E_{static}. \quad (6)$$

Then, we model the GPU energy consumption of different components in each iteration by modeling the average power of executing one iteration:

$$E_{grad} = P_{grad} * T_{grad} * n; \quad (7)$$
$$E_{sync} = P_{sync} * T_{sync} * n; \quad (8)$$
$$E_{static} = P_{static} * T_{iter} * n, \quad (9)$$

where P_{grad}, P_{sync}, and P_{static} are the power (i.e., the consumed energy per unit of time) of different components on a single GPU. Previous studies [14, 23] proposed that the power of static and dynamic components can be modeled as:

$$P_{dynamic} = a * m * V^2 * f; \quad (10)$$
$$P_{static} = V * N * K_{design} * \tilde{I}_{leak}, \quad (11)$$

where a denotes the average utilization ratio, C is the total capacitance, V is the supply voltage. N, K_{design}, and $\tilde{I}_{leak}$ are constant parameters determined by the hardware. Therefore, in these two models, the power of dynamic and static components scales with the frequency f and the voltage V.

Although it is common for GPU manufacturers to provide tools to get the frequencies of GPU devices, there is no easy way to know the voltage directly, nor how voltage scales with frequency [25]. Therefore, to model the energy consumption of DL training jobs, we first model how the voltage of a GPU scales with frequency.

GPU Voltage and frequency. The voltage of a GPU scales with GPU frequency in a piecewise function: the voltage is a constant value when the frequency is low; after a specific frequency, the voltage increases linearly with the frequency [25]. This specific frequency depends on the hardware device. Therefore, according to Equation 10, P_{grad} and P_{sync} scale linearly with f at lower frequencies and is a cubic complements of f at higher frequencies; according to Equation 11, P_{static} is constant at lower frequencies and is a linear function of f at higher frequencies. Next, we separately model P_{grad} , P_{sync}, and P_{static} for low frequencies and high frequencies.

Modeling P_{grad}. Except for GPU frequency, P_{grad} is also related to the local batch size on each GPU. The larger the local batch size, the larger the power of a GPU. However, the power of a GPU does not scale linearly with local batch size. As is shown in Figure 3, we observe that the power of the GPU scales sublinearly with local batch size. Therefore, we model P_{grad} as a logarithmic function of bs:

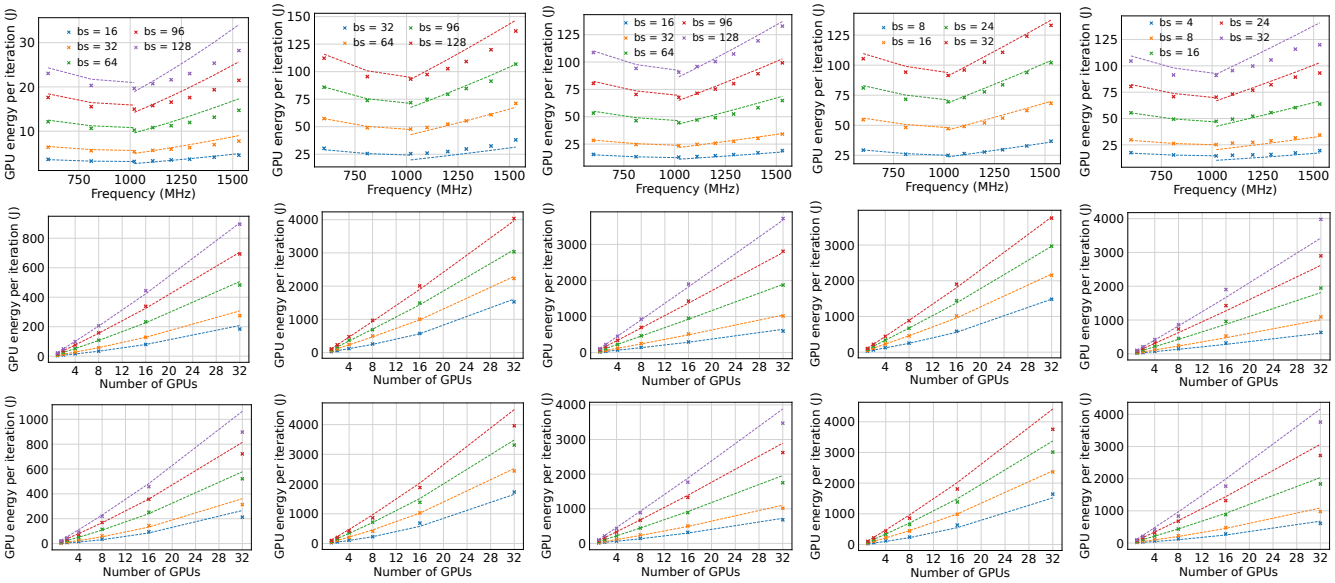
$$P_{grad} = \begin{cases} \kappa_{grad} * (\alpha_{l_g} * log(bs + \theta_{l_g} + \beta_{l_g})) & if\ f < f_0 \\ \kappa_{grad} * (\alpha_{h_g} * log(bs + \theta_{h_g} + \beta_{h_g})) & otherwise; \end{cases} \quad (12)$$

$$\kappa_{grad} = \begin{cases} a_{l_g} * f + b_{l_g} & if\ f < f_0 \\ a_{h_g} * f^3 + b_{h_g} * f^2 + c_{h_g} * f + d_{h_g} & otherwise, \end{cases} \quad (13)$$

where α, β, θ, a, b, c, d are learnable parameters, f_0 is the hardware-dependent breaking point between low frequencies and high frequencies.

Modeling P_{sync}. Similar to P_{grad}, we model P_{sync} at both low and high frequencies. The time spent on model parameters synchronization is only relevant to the DNN model and the bandwidth between GPU devices. So P_{sync} is modeled as:

$$P_{sync} = \begin{cases} a_{l_s} * f + b_{l_s} & if\ f < f_0 \\ a_{h_s} * f^3 + b_{h_s} * f^2 + c_{h_s} * f + d_{h_s} & otherwise, \end{cases} \quad (14)$$



(a) ResNet18 (b) VGG16 (c) Inception V3 (d) GPT2 (e) Deep Speech 2

Figure 4: GPU energy consumed per iteration by different DNN models with configurations. The scattered data points are the GPU energy values measured on real NVIDIA V100 GPUs and the lines show the predicted values of fitted models. The figures on the top show how GPU energy consumed per iteration changes with GPU frequency on one GPU. The figures in the middle show how GPU energy consumed per iteration changes with the number of GPUs with 600 MHz frequency (low frequency). The figures at the bottom show how GPU energy consumed per iteration changes with the number of GPUs with 1200 MHz frequency (high frequency). Note that the local batch sizes are kept the same and the global batch size scales linearly with the number of GPUs.

where a, b, c, d are learnable parameters, f_0 is the hardware-dependent breaking point between low frequencies and high frequencies.

Modeling P_{static}. According to Equation 11, P_{static} is proportional to V:

$$P_{static} = \begin{cases} P_{static_l} & if\ f < f_0 \\ c_h * f & otherwise, \end{cases} \tag{15}$$

where P_{static_l} and c_h are parameters that can be fitted.

Figure 4 shows how our energy model is fitted to measured energy consumption results on different DNN models and different configurations. We measured the throughputs on real NVIDIA V100 GPUs. As is shown in the figures, The fitted models represent the real energy consumption performance well. We will evaluate the fitted models in detail in Section 6.3.

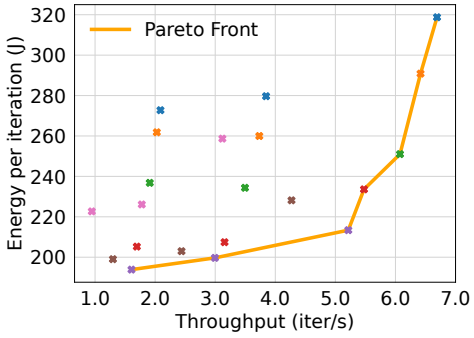


Figure 5: The energy-throughput tradeoff. The data points in the same color represents the values measured under the same GPU frequencies.

4.3 Energy-throughput Tradeoffs

It would be ideal to achieve a small JCT with little energy consumption. However, from our throughput model (Section 4.1), we can see that if we want to improve the throughput of a DL training job while keeping the global batch size unchanged, we need to use more GPU devices and set the GPUs to the highest supported frequency. However, according to our energy-consumption model (Section 4.2), training a DL job with a large number of GPUs and the highest GPU frequency consumes more energy than many other settings. There is a tradeoff between the JCT and the energy consumption of a DL training job.

Figure 5 is an example to show this tradeoff. The values were measured by training GPT2 with a global batch size of

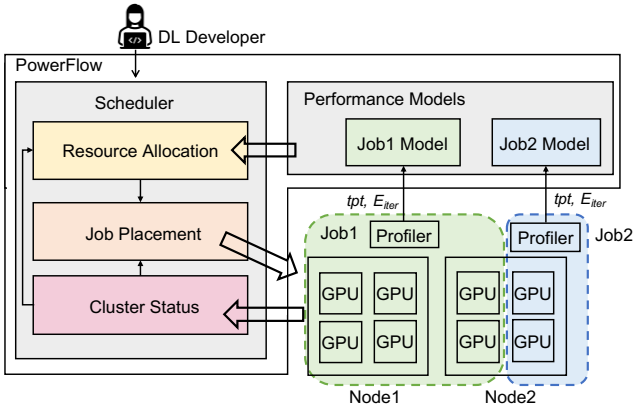


Figure 6: PowerFlow architecture.

64 on real NVIDIA V100 GPUs. Each data point represents the throughput and the energy consumed per iteration of one feasible configuration. The data points on the orange curve achieve Pareto optimality [15], for which we cannot reduce energy consumed per iteration without sacrificing throughput, and vice versa. Training with the configurations that are not on this curve would not be *energy-efficient*, as there are configurations that can achieve higher throughputs with similar energy consumption, or consume less energy with a similar throughput.

For a single DL training job, we can find a configuration that achieves Pareto equilibrium by constantly trying to use a different number of GPUs and GPU frequency. However, it is more complicated to achieve the tradeoff in a GPU cluster where the total number of GPUs is limited. If one training job runs under a configuration that achieves Pareto equilibrium, there might not be enough GPUs left for other jobs to achieve Pareto equilibrium. Also, with an energy budget, there is no straightforward solution to decide which job should use more GPUs or which GPU's frequency should be higher.

5 System Design

5.1 System Overview

Figure 6 shows the architecture of PowerFlow. DL developers submit DL training jobs to PowerFlow. For each job, a throughput model and an energy consumption model are built to predict the DL training job's performance under different configurations (i.e., the number of GPUs used and the GPU frequency). To make more accurate performance predictions, PowerFlow first pre-runs the submitted jobs offline on one GPU with different frequencies and profiles the performance to fit the performance models. This takes about four minutes. As DL training jobs usually run for hours or even days, this pre-running overhead is rather small and has little influence on the final average JCT. Then, PowerFlow schedules the jobs with a resource allocation module and a job placement module.

The resource allocation module decides the number of GPUs allocated to each job based on its performance models. Upon each scheduling event such as job arrival or completion, the resource allocation module may update the resource allocation of some jobs through elastic scaling, i.e., adjusting the number of GPUs allocated to a job based on the job's real performance and the number of available GPUs. The module also computes the local batch size (i.e., dividing global batch size by the number of GPUs) for a given job. The job placement module selects GPUs from the cluster for each job based on the topology. While each job is running, PowerFlow profiles the real throughput and energy consumption of each iteration. Based on the online and offline profiled performance, PowerFlow continuously fits the performance models of each job to make more accurate performance predictions and better scheduling decisions.

5.2 Resource Allocation

In this section, we describe the detailed design of the resource allocation module which finds the tradeoff between JCT and energy consumption. We propose *energy efficiency* to capture whether the consumed energy brings enough throughput improvement, based on which PowerFlow dynamically allocates resources to submitted jobs. PowerFlow applies network packing and buddy allocation to avoid resource fragmentation, thus reducing energy consumption on idle GPUs.

Energy efficiency. The key to achieving a short JCT with an energy budget is to be more *energy-efficient*. For a DL training job with a fixed global batch size, the larger the throughput, the smaller the JCT. Therefore, we define *energy efficiency* as the throughput brought by each unit of energy consumption.For job j, we have:

$$ee_j = \frac{tpt_j}{E_j}, \tag{16}$$

where ee represents the energy efficiency of a DL training job, E is the total energy consumed to finish the job, and tpt is the average throughput of the job. DL developers usually specify a maximum number of iterations to execute as a termination condition of a DL training job. Let us denote this maximum iteration number of job j as $iter_j$. Combining Equation 1 and 16, we have:

$$ee_j = \frac{iter_j}{JCT_j * E_j}. \tag{17}$$

According to our performance models in Section 4.1 and Section 4.2, if the GPU frequency is too large, the total energy consumption would be large, thus is energy-inefficient; if the GPU frequency is too small, the JCT would be very long, leading to poor energy efficiency as well.

Algorithm 1 PowerFlow Resource Allocation

```
1:  function PowerFlow(jobs)
2:      // GPU allocation
3:      for i from 0 to J do
4:          i.n ← 0
5:          // The most energy efficient GPU frequency
6:          i.f ← f_0
7:          i.priority_g ← GetPriorityG(i)
8:          QueueG.insert(i)
9:      g ← G
10:     while g > 0 do
11:         // Pick the job with largest marginal return
12:         i ← QueueG.dequeue()
13:         i.n ← i.n + 1
14:         g ← g − 1
15:         // Update the marginal return of job i
16:         i.priority_g ← GetPriorityG(i)
17:         QueueG.insert(i)
18:         if g || gpu_per_node = 0 then
19:             if ∑ j = 0^J i.P ≥ η ∗ G ∗ P_max then
20:                 Break
21:     // GPU frequency configuration
22:     for i from 0 to J do
23:         i.priority_f ← GetPriorityF(i)
24:         QueueF.insert(i)
25:     while ∑ j = 0^J i.P ≤ η ∗ G ∗ P_max do
26:         // Pick the job with largest marginal return
27:         i ← QueueF.dequeue()
28:         i.f ← i.f + Δ_f
29:         i.priority_g ← GetPriorityF(i)
30:         QueueD.insert(i)
```

The energy-aware scheduling problem. To tradeoff between average JCT and total energy consumption with an energy budget, we propose a simple parameter $\eta \in [0, 1)$: η represents how much *power* the service provider wishes to consume compared to the case where the cluster is fully allocated with the default frequency. η can converse to an "energy budget" by $budget = \eta * G * P_{max} * t$, where G is the total number of GPUs in the cluster, P_{max} is the average GPU power when executing a DL training job with the highest supported GPU frequency, and t is the period of time during which the energy consumption in the cluster does not exceed $budget$. PowerFlow allocates as many GPUs as possible to the jobs and uses the highest GPU frequencies when $\eta = 1$ and allocates as few GPUs as possible with the most energy-efficient frequencies when η gets closer to 0. The smaller the η, the smaller the power of the cluster. The whole cluster may consume even more energy with a small η: although the energy consumed by the cluster in every t does not exceed $budget$, the time needed to finish all submitted jobs might be multiplied several times, which is not energy-efficient. Therefore, the cloud provider should choose a proper η.

With η, PowerFlow allocates GPUs to submitted jobs while making sure that the total power of the cluster does not exceed $power\_limit = \eta * G * P_{max}$. The power of each job can be estimated by our performance model as $P = E_{iter}/T_{iter}$. Suppose that there are J jobs submitted to the cluster, we try to solve the following optimization problem:

$$\min \ \frac{1}{n} \sum_{j=0}^{J-1} JCT_j \tag{18}$$

$$s.t. \ \sum_{k=0}^{G-1} P_i \leq \eta * G * P_{max}. \tag{19}$$

A straw-man solution is to search for the most energy-efficient configuration of each job and execute the jobs with these energy-efficient configurations. This makes the jobs energy-efficient if the cluster has enough GPUs to run all of the submitted jobs. However, this solution does not consider the limited resources in a cluster: if a job uses more GPUs to achieve shorter JCT or uses low GPU frequency to reduce energy consumption, the other jobs in the cluster might have to wait for the GPU resources for a longer time. This might lead to an even longer average JCT compared to the schedulers that are not energy-aware.

From the straw-man solution, we can see that the key to achieving our goal is to consider energy efficiency from the perspective of the GPU cluster. We should not only consider how the performance would change if we adjust the configuration of a DL job, but also the influence on the whole cluster.

Guided by this insight, PowerFlow starts by executing all of the jobs with the most energy-efficient GPU frequency. Then, PowerFlow's resource allocation module schedules the submitted jobs in two steps: (1) allocate the GPUs in the clusters one by one under the most energy-efficient frequencies; (2) if the total power of the cluster does not exceed $power\_limit$, increase the GPU frequencies to further accelerate the jobs. Algorithm 1 shows the pseudocode of these two steps, which we will describe separately in detail.

GPU allocation. We develop a greedy algorithm to solve the GPU allocation problem (lines 10-20). The intuition is to allocate the GPUs to the job with the highest *marginal return*. PowerFlow takes this marginal return as the *priority* of each submitted job. The *marginal return* of job j for GPU allocation is defined as:

$$priority_G = \frac{(JCT_{j,n,f} - JCT_{j,n+1,f})/JCT}{(E_{j,n+1,f} - E_{j,n,f})/E}, \tag{20}$$

where the numerator is the relative JCT reduction of allocating one more GPU to j compared to the total JCT of the whole cluster, and the denominator is the relative energy increase compared to the total energy consumption of the cluster.

PowerFlow maintains a priority queue to order the jobs in $priority_G$ (lines 12-17). For each iteration, the algorithm

dequeues the head from the queue, which is the job with the largest marginal return (lines 11-12). The algorithm allocates one GPU to the job (lines 13-14). Then, the algorithm computes the new marginal return of the job and inserts the job back into the queue (lines 15-16). The iterations finish until all GPUs are allocated, or the total power of the cluster exceeds *power_limit* (line 10 and lines 18-20).

GPU frequency configuration. After allocating the GPUs, if the total power of the cluster does not exceed *power_limit*, PowerFlow will increase the GPU frequencies to further accelerate some jobs. Similar to GPU allocation, PowerFlow defines another *marginal return* for frequency configuration:

$$priority_F = \frac{(JCT_{j,n,f} - JCT_{j,n,f+\Delta_f})/JCT}{(E_{j,n,f+\Delta_f} - E_{j,n,f})/E}. \quad (21)$$

Because the GPU frequency can only be configured as a value from a set of supported values, we increase the GPU frequency by Δ_f each time. The Δ_f is determined by the hardware. PowerFlow keeps increasing the f of the job with the highest marginal return until all jobs use the highest supported frequency or the total power of the cluster exceeds *power_limit* (lines 25-30).

PowerFlow re-allocates each job and adjusts the configurations at each *scheduling event*, including job submission, job scaling, and job completion. For each scheduling decision (changing the configuration of a job), PowerFlow checks if each job has been run on this number of GPUs before. If not, PowerFlow profiles all possible GPU frequencies during the following four minutes by partitioning the four-minute time slot into slices at iteration boundaries and dynamically changing the GPU frequency for each slice. Meanwhile, PowerFlow further fits the performance models so that the models can help make more accurate performance predictions in the future. After the profiling, PowerFlow would receive a job scaling event and adjust the scheduling decisions with the updated performance models.

5.3 Job Placement

The job placement mechanism of some existing schedulers [61, 65] might lead to cluster fragmentation, i.e., there are unutilized GPUs in more than one node. We follow previous work [31] to adopt "network packing" and restrict the number of workers of each job to a power of two. It is proved that in this way, at most one job on any node uses two or more nodes, thus avoiding the training performance to be affected by placement [31].

Then, we apply buddy allocation [67] combined with job migration to eliminate resource fragmentation. The unused nodes are shut down by PowerFlow to avoid extra energy consumption. With this job placement mechanism, we can reduce the energy consumed by the idle GPUs on the nodes that are turned on.

Task	Dataset	Model	Batch Size
CV	ImageNet [19]	ResNet18 [26]	32 - 512
		VGG16 [56]	32 - 512
		Inception V3 [58]	16 - 512
NLP	aclImdb V1 [42]	GPT-2 [54]	8 - 128
Speech Recognition	LibriSpeech [48]	Deep Speech 2 [9]	8 - 256

Table 1: DNN models used in the evaluation.

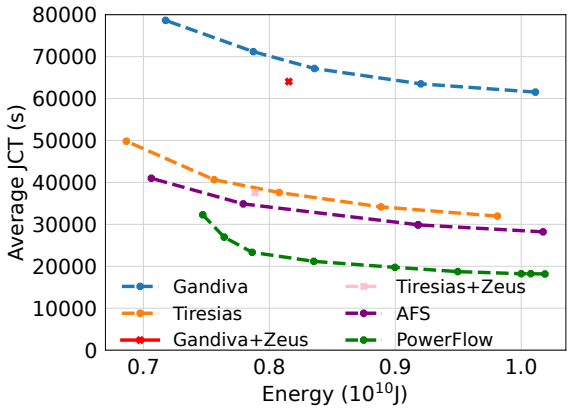


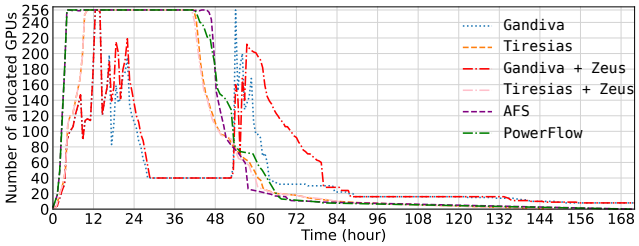
Figure 7: End-to-end scheduling result.

6 Evaluation

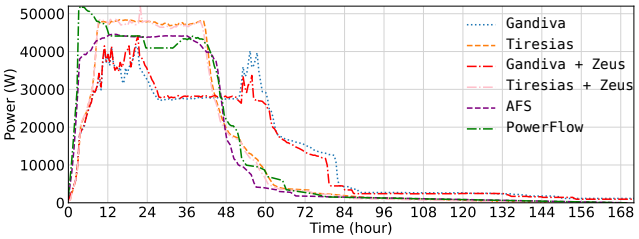
6.1 Methodology

Simulator. To evaluate our scheduler at large scales, we develop a simulator using the profiled information in real NVIDIA V100 GPUs. The simulator simulates all job-level events, including job arrival, scaling, and completion. We profile the throughputs and energy consumption of each job on real GPU servers as the input of the simulator. To make the simulator more realistic, we have also included the pre-run overhead (including JCT overhead and energy consumption overhead) and incorporated it into the simulator. The simulator assigns this overhead to each job on job submission.

Workloads. We use a subset of the public Alibaba trace [64] for end-to-end evaluation. The trace contains 1901 jobs submitted in a 24-hour period, which is substantial compared to existing work [24, 53]. Each job in the original trace has information on its submission time, number of GPUs, and duration. However, no information is provided on the DNN architectures being trained, the dataset, and the batch size. For each job, we randomly choose a DNN model with a batch size from a pool of representative settings listed in Table 1. Similar to previous work [31, 47], we use the duration in the trace and the pre-measured throughputs to calculate the number of iterations needed to finish each job.



(a) GPU allocation of different schedulers over time.



(b) Power of different schedulers over time.

Figure 8: Comparison between different schedulers in end-to-end experiments.

Baselines. We compare PowerFlow to five baselines.

- **Gandiva:** Gandiva [65] is a DL scheduler that uses introspective scheduling to refine scheduling decisions continuously. It is not elastic (i.e., uses the number of GPUs specified in job traces) and is not energy-aware.
- **Tiresias:** Tiresias [24] uses two-dimensional scheduling algorithms customized for DL jobs. It is also not elastic and is not energy-aware.
- **Gandiva + Zeus:** Zeus [66] is an optimization framework that finds the tradeoff between energy consumption and training time for a *non-elastic* DL training job. We combine Zeus to Gandiva [65] to evaluate this tradeoff in the scheduling of a GPU cluster. It is energy-aware but not elastic.
- **Tiresias + Zeus:** Similar to Gandiva + Zeus, we combine Zeus to Tiresias [24]. It is energy-aware but not elastic.
- **AFS:** AFS [30] is a scheduling algorithm that accelerates DL training jobs by balancing resource efficiency and short job prioritization. It is elastic but not energy-aware.

Evaluation metric. The design goal of PowerFlow is to reduce the average JCT under a certain energy budget. Therefore, we compare the average JCT with different energy consumption for all of the mentioned scheduling algorithms. We also compare the GPU utilization and the power of the cluster over time.

DNN Model	Throughput	Energy
ResNet18	0.061	0.069
VGG16	0.038	0.060
Inception V3	0.026	0.045
GPT2	0.021	0.068
Deep Speech 2	0.045	0.035

Table 2: MAPE of the fitted throughput model and energy consumption model.

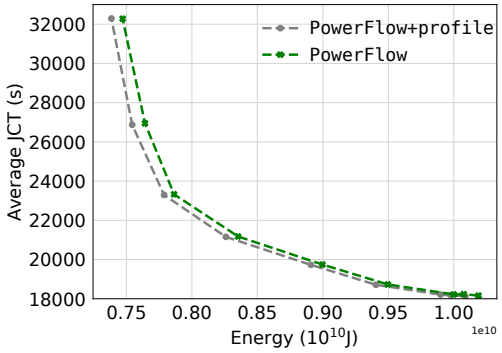
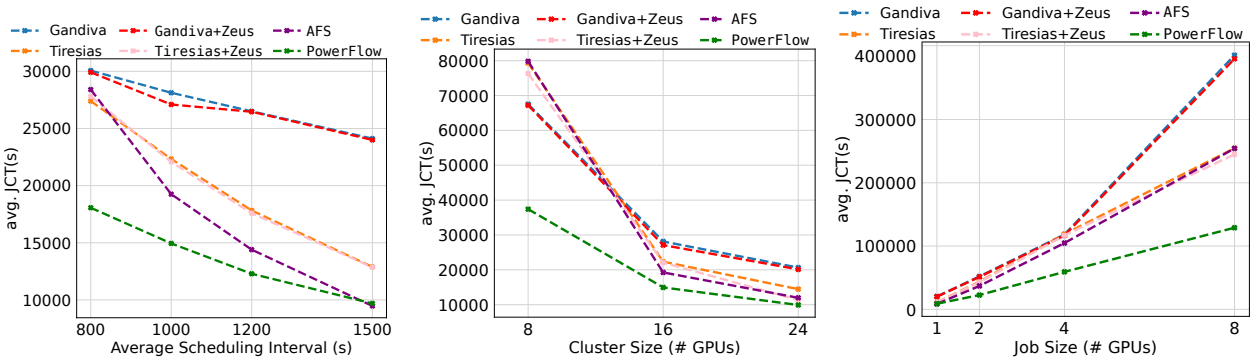


Figure 9: PowerFlow's scheduling results with pre-profiled performance data and with the performance models.

6.2 End-to-End Results

Figure 7 compares the average JCT under different energy consumption. For the baselines that are not energy-aware, we change the total energy consumption of executing all of the jobs by changing the frequencies of all GPU devices. Zeus searches for the energy-efficient configuration for each DL job, so for Gandiva + Zeus or Tiresias + Zeus, we can only get a fixed result with the configurations that Zeus found energy-efficient. As is shown in the figure, PowerFlow achieves the shortest average JCT under different energy consumption. If the GPU frequency is set smaller than 1020MHz for Gandiva, Tiresias, and AFS, they achieve longer JCT while consuming even more energy. Therefore, we do not show these results in the figure.

Compared to Gandiva, Tiresias, Gandiva + Zeus, Tiresias + Zeus, and AFS, PowerFlow improves the average JCT by 3.39×, 1.73×, 2.90×, 1.62×, and 1.57× at most, respectively. Gandiva and Tiresias are non-elastic schedulers for DL jobs. Compared to them, PowerFlow utilizes more GPUs to accelerate the jobs when there are spare GPUs in the cluster and uses fewer GPUs when the cluster is not large enough to serve all of the submitted jobs. Also, with the placement strategies of Gandiva and Tiresias, there might be fragmentations in the cluster. The energy consumed by these fragmentations also makes Gandiva and Tiresias non-energy-efficient. Zeus only applies to non-elastic jobs and is not flexible to achieve a shorter JCT if the cloud provider sets a higher energy budget



(a) Sensitivity to scheduling interval. (b) Sensitivity to cluster size. (c) Sensitivity to job size.

Figure 10: PowerFlow's sensitivity to different factors.

goal. AFS achieves a rather short JCT because of its elasticity, but it is not energy-aware and can not consume energy more efficiently.

Next, to better analyze the features of different scheduling algorithms, we picked one data point from each of the curves in Figure 7. Then, we compare the number of GPUs allocated and the power of the cluster. For the schedulers that are not combined with Zeus, we choose the data point whose total energy consumption is closest to Gandiva + Zeus. Figure 8 shows the number of allocated GPUs and the power for the chosen data points over time. From the figure, we observe that PowerFlow takes full advantage of the idle GPU resources while controlling the total power of the cluster. In this way, the submitted jobs can complete earlier. As is described in Section 5.2, PowerFlow profiles the performance of DL jobs and dynamically adjusts the jobs' resource allocation while fitting the performance models. Therefore, compared to the other baselines, PowerFlow consumes more energy in the first few hours. Gandiva, Tiresias, Gandiva + Zeus, and Tiresias + Zeus are not elastic, so they do not utilize idle GPUs if any.

6.3 Effects of Performance Models

Here we evaluate if our throughput model and energy consumption model are accurate enough for PowerFlow to make scheduling decisions. First, we evaluate the Mean Absolute Percentage Error (MAPE) of the two models by using 90% of the profiled performance values to fit the models and the 10% left to evaluate the models. Table 2 shows the MAPE results. Across the results of all DNN models, the average MAPE of the fitted models does not exceed 10%, indicating that our models represent the observed performance measurements very well.

Figure 9 compares the results of using our fitted performance models for scheduling and using pre-profiled training performance for scheduling. Note that when scheduling with profiled performance, we assume that the scheduler already knows the performance of each job, so the energy for profiling the performance is not included, which is the ideal case. We use the same workload in Section 6.2 and compare the average JCT as well as total energy consumption. Under the same energy consumption, the difference in average JCT between using profiled performance data and using the performance models is less than 2%. This indicates that our performance models can be well-fitted for the scheduler to make scheduling decisions. Although using the pre-profiled performance results achieves a slightly shorter average JCT, if we pre-run the performance of all of the configurations offline, it requires too many GPU resources and has an overhead of more than one hour. In reality, this JCT overhead is too large for many DL training jobs, not to mention that the pre-running consumes a lot of extra energy.

6.4 Sensitivity Analysis

In this section, we evaluate PowerFlow's sensitivity to different factors. Following previous work [47], we generated 100-job traces randomly for sensitivity analysis. As we explained in Section 6.2, unlike other schedulers that have different scheduling results when the GPU frequency is set to different values, Zeus only chooses one energy-efficient configuration for each job. We observe that when scheduling jobs from the same trace in the same cluster, Gandiva + Zeus and Tiresias + Zeus achieves similar total energy consumption (less than 5% difference). To make a fair comparison, for the schedulers that are not combined with Zeus, we choose the scheduling results that have comparable total energy consumption with Gandiva + Zeus and Tiresias + Zeus (less than 5% difference). Then, we compare the average JCT achieved by these schedulers.

Sensitivity to scheduling interval. First, we compare the average JCT of PowerFlow and the baselines under different loads in terms of average job arrival interval. Figure 10(a) shows the results. As expected, for all schedulers, the average JCT gets shorter as the interval increases. Compared with the baselines, PowerFlow always achieves a short average JCT

and has a larger JCT improvement with a heavier cluster load. This is because PowerFlow allocates the limited GPUs to the jobs that can bring higher energy efficiency to the cluster.

Sensitivity to cluster size. Given the same set of jobs, the average JCT is shorter in a larger cluster with more GPUs. We run PowerFlow and the baselines on the same job trace with different cluster sizes. Figure 10(b) shows that PowerFlow outperforms the baselines more in small clusters. This is because in a large cluster, both PowerFlow and AFS can scale out the jobs to utilize more GPUs for average JCT improvement; but in a small cluster, PowerFlow utilizes the GPUs more efficiently.

Sensitivity to job size. In today's DL training platforms, users usually specifically allocate a fixed number of GPUs to the jobs. Both Gandiva and Tiresias require DL developers to specify the number of GPUs for DL jobs. We evaluate the average JCT on the same cluster size when job size varies. As expected, from Figure 10(c), we can see that for all scheduling algorithms, the average JCT is smaller with small jobs. For larger jobs, we observed a larger improvement by PowerFlow compared to the baselines. This indicates that training some DNN models with a small number of GPUs is more energy efficient than training with a large number of GPUs and default GPU efficiency.

7 Related Work

Scheduler for DL jobs. Early efforts used cluster managers like Kubernetes or YARN to schedule DL jobs in the cloud without considering the characteristics of DL jobs, which results in low performance [13, 32, 33]. Recent efforts proposed specialized cluster schedulers for DL training jobs [16, 24, 43, 51, 53, 65]. These efforts focus on optimizing for JCT, fairness, or GPU utilization while ignoring the energy consumption of DL jobs and the GPU cluster.

Energy measurement for DL jobs. Existing studies have investigated the measurement or estimation of the carbon emission and energy consumption of machine learning or DL jobs [27, 50]. Similar to these studies, PowerFlow measures the energy consumption on GPU via NVML [2] and then estimates the energy consumption with different configurations with PowerFlow's performance models.

Energy optimization for DL jobs. Several efforts have studied the impact of GPU DVFS and power configuration on the performance of GPU devices and DL jobs [45, 60]. These methods require offline profiling or modeling, which is not realistic or brings huge overheads to online cluster schedulers. Zeus [66] is an online optimization framework for recurring DL training jobs that finds the trade-off between throughput optimization and energy consumption by automatically tuning the batch size and GPU power limit of training jobs. However, it does not apply to elastic jobs and lacks the view of the whole cluster if applied to a GPU cluster scheduler directly.

Energy efficient VM scheduling. Energy consumption optimization in traditional datacenters has been studied by a recent line of research work. Some studies leverage DVFS based on system performance requirements at the given time [62]. This requires fine-grained resource sharing, which is undeveloped and unstable in today's GPU usage. There are also energy consumption optimization schedulers based on workload forecasting [55, 59, 63]. These methods either require user-defined SLA, which does not apply to today's common practice of DL training, or balance workloads and shut down unused servers, which is similar to PowerFlow's job placement mechanism.

8 Conclusion

In this paper, we proposed PowerFlow, an energy-aware scheduler for GPU clusters that reduces the average JCT under an energy budget with the tradeoff between the average JCT and total GPU energy consumption. PowerFlow predicted the throughput and energy consumption performance of DL training jobs with performance models. We developed a scheduling algorithm that dynamically allocates GPUs to submitted jobs and adjusts the GPU frequencies based on the performance predictions made by the performance models. PowerFlow applied network packing and buddy allocation to job placement, avoiding extra energy consumption of cluster fragmentations. The evaluation results showed that under the same energy consumption, PowerFlow improved the average JCT by 1.57 - 3.39× at most, compared to competitive baselines.

References

[1] AI and Compute. https://openai.com/blog/ai-and-compute/, 2019. Retrieved on March 16, 2022.

[2] NVIDIA Management Library (NVML). https://developer.nvidia.com/nvidia-management-library-nvml, 2019. Retrieved on November 26, 2022.

[3] Training a single AI model can emit as much carbon as five cars in their lifetimes. https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/, 2019. Retrieved on November 5, 2022.

[4] Google Aims to Attain Zero Carbon Footprint Goal by 2030. https://www.nasdaq.com/articles/google-aims-to-attain-zero-carbon-footprint-goal-by-2030-2020-09-15, 2020. Retrieved on December 12, 2022.

[5] Microsoft will be carbon negative by 2030. https://blogs.microsoft.com/blog/2020/01/16/microsoft-will-be-carbon-negative-by-2030/, 2020. Retrieved on December 12, 2022.

[6] Facebook reaches 100% renewable-energy milestone. https://www.cbsnews.com/news/facebook-renewable-energy-commitment-100-percent-milestone/, 2021. Retrieved on December 12, 2022.

[7] GPT-3 Powers the Next Generation of Apps. https://openai.com/blog/gpt-3-apps/, 2021. Retrieved on March 16, 2022.

[8] What Is Intel Turbo Boost Technology? https://www.intel.com/content/www/us/en/gaming/resources/turbo-boost.html, 2021. Retrieved on November 8, 2022.

[9] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse H. Engel, Linxi Fan, Christopher Fougner, Awni Y. Hannun, Billy Jun, Tony Han, Patrick LeGresley, Xiangang Li, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Sheng Qian, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Chong Wang, Yi Wang, Zhiqian Wang, Bo Xiao, Yan Xie, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*, volume 48, pages 173–182, 2016.

[10] Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *CoRR*, abs/2007.03051, 2020.

[11] Alexandre Berard, Olivier Pietquin, Christophe Servan, and Laurent Besacier. Listen and translate: A proof of concept for end-to-end speech-to-text translation. *CoRR*, abs/1612.01744, 2016.

[12] Srikant Bharadwaj, Shomit Das, Yasuko Eckert, Mark Oskin, and Tushar Krishna. DUB: dynamic underclocking and bypassing in nocs for heterogeneous GPU workloads. In Tushar Krishna, John Kim, Sergi Abadal, and Joshua San Miguel, editors, *NOCS '21: International Symposium on Networks-on-Chip, Virtual Event, October 14-15, 2021*, pages 49–54. ACM, 2021.

[13] Scott Boag, Parijat Dube, Benjamin Herta, Waldemar Hummer, Vatche Ishakian, K Jayaram, Michael Kalantar, Vinod Muthusamy, Priya Nagpurkar, and Florian Rosenberg. Scalable multi-framework multi-tenant lifecycle management of deep learning training jobs. In *Workshop on ML Systems, NeurIPS 2017*, 2017.

[14] J Adam Butts and Gurindar S Sohi. A static power model for architects. In *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, pages 191–201. IEEE, 2000.

[15] Yair Censor. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, 4(1):41–59, 1977.

[16] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In *Proceedings of the 15th European Conference on Computer Systems, EuroSys 2020*, pages 1:1–1:16, 2020.

[17] Chenyi Chen, Ari Seff, Alain L. Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of 2015 IEEE International Conference on Computer Vision, ICCV 2015*, pages 2722–2730, 2015.

[18] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Proceedings of 26th Annual Conference on Neural Information Processing Systems, NeurIPS 2012.*, pages 1232–1240, 2012.

[19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *Proceedings of 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2009*, pages 248–255, 2009.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*, pages 4171–4186, 2019.

[21] Youwei Ding, Xiaolin Qin, Liang Liu, and Taochun Wang. Energy efficient scheduling of virtual machines in cloud with deadline constraint. *Future Gener. Comput. Syst.*, 50:62–74, 2015.

[22] Jesse Dodge, Taylor Prewitt, Remi Tachet des Combes, Erika Odmark, Roy Schwartz, Emma Strubell, Alexandra Sasha Luccioni, Noah A. Smith, Nicole DeCario,

and Will Buchanan. Measuring the carbon intensity of AI in cloud instances. In *FAccT '22: 2022 ACM Conference on Fairness, Accountability, and Transparency, Seoul, Republic of Korea, June 21 - 24, 2022*, pages 1877–1894. ACM, 2022.

[23] Ricardo Gonzalez, Benjamin M. Gordon, and Mark A. Horowitz. Supply and threshold voltage scaling for low power CMOS. *IEEE J. Solid State Circuits*, 32(8):1210–1216, 1997.

[24] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Harry Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *Proceedings of 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019*, pages 485–500, 2019.

[25] João Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomás. GPGPU power modeling for multi-domain voltage-frequency scaling. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*, pages 789–800. IEEE Computer Society, 2018.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition, CVPR 2016*, pages 770–778, 2016.

[27] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *CoRR*, abs/2002.05651, 2020.

[28] Miro Hodak, Masha Gorkovenko, and Ajay Dholakia. Towards power efficiency in deep learning on data center hardware. In Chaitanya K. Baru, Jun Huan, Latifur Khan, Xiaohua Hu, Ronay Ak, Yuanyuan Tian, Roger S. Barga, Carlo Zaniolo, Kisung Lee, and Yanfang (Fanny) Ye, editors, *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*, pages 1814–1820. IEEE, 2019.

[29] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Proceedings of 33th Annual Conference on Neural Information Processing Systems, NeurIPS 2019.*, pages 103–112, 2019.

[30] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. Elastic resource sharing for distributed deep learning. In James Mickens and Renata Teixeira, editors, *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, pages 721–739. USENIX Association, 2021.

[31] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. Elastic resource sharing for distributed deep learning. In *Proceedings of 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021*, pages 721–739, 2021.

[32] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *Proceedings of 2019 USENIX Annual Technical Conference, ATC 2019*, pages 947–960, 2019.

[33] Myeongjae Jeon, Shivaram Venkataraman, Junjie Qian, Amar Phanishayee, Wencong Xiao, and Fan Yang. Multi-tenant GPU clusters for deep learning workloads: Analysis and implications. *Technical report, Microsoft Research*, 2018.

[34] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. In *Proceedings of Machine Learning and Systems 2019, MLSys 2019*, 2019.

[35] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 463–479. USENIX Association, 2020.

[36] Toshiya Komoda, Shingo Hayashi, Takashi Nakada, Shinobu Miwa, and Hiroshi Nakamura. Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping. In *2013 IEEE 31st International Conference on Computer Design, ICCD 2013, Asheville, NC, USA, October 6-9, 2013*, pages 349–356. IEEE Computer Society, 2013.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of 26th Annual Conference on Neural Information Processing Systems, NeurIPS 2012.*, pages 1106–1114, 2012.

[38] Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In Zhipeng Cai,

Rafal A. Angryk, Wen-Zhan Song, Yingshu Li, Xiaojun Cao, Anu G. Bourgeois, Guangchun Luo, Liang Cheng, and Bhaskar Krishnamachari, editors, *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom), BDCloud-SocialCom-SustainCom 2016, Atlanta, GA, USA, October 8-10, 2016*, pages 477–484. IEEE Computer Society, 2016.

[39] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Proceedings of 11th USENIX Symposium on Operating Systems Design and Implementation, (OSDI 2014)*, pages 583–598, 2014.

[40] Mu Li, David G. Andersen, Alexander J. Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Proceedings of 28th Annual Conference on Neural Information Processing Systems, NeurIPS 2014.*, pages 19–27, 2014.

[41] Gangmuk Lim, Jeongseob Ahn, Wencong Xiao, Youngjin Kwon, and Myeongjae Jeon. Zico: Efficient GPU memory sharing for concurrent DNN training. In Irina Calciu and Geoff Kuenning, editors, *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, pages 161–175. USENIX Association, 2021.

[42] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, ACL 2022*, pages 142–150, June 2011.

[43] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient GPU cluster scheduling. In *Proceedings of 17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020*, pages 289–304, 2020.

[44] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020.

[45] Xinxin Mei, Qiang Wang, and Xiaowen Chu. A survey and measurement study of GPU DVFS on energy conservation. *Digit. Commun. Networks*, 3(2):89–100, 2017.

[46] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. Looking beyond gpus for DNN scheduling on multi-tenant clusters. In Marcos K. Aguilera and Hakim Weatherspoon, editors, *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, pages 579–596. USENIX Association, 2022.

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

[48] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An ASR corpus based on public domain audio books. In *Proceedings of 2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015*, pages 5206–5210, 2015.

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[50] David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *CoRR*, abs/2104.10350, 2021.

[51] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the 13th European Conference on Computer Systems, EuroSys 2018*, pages 1–14, 2018.

[52] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed DNN training acceleration. In Tim Brecht and Carey Williamson, editors, *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, pages 16–29. ACM, 2019.

[53] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In Angela Demke Brown and Jay R. Lorch, editors, *15th USENIX Symposium on Operating Systems Design and*

Implementation, OSDI 2021, July 14-16, 2021. USENIX Association, 2021.

[54] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[55] Milad Ranjbari and Javad Akbari Torkestani. A learning automata-based algorithm for energy and SLA efficient consolidation of virtual machines in cloud data centers. *J. Parallel Distributed Comput.*, 113:55–62, 2018.

[56] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of 3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[57] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 13693–13696. AAAI Press, 2020.

[58] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pages 2818–2826, 2016.

[59] Xiaoyong Tang, Xiaoyi Liao, Jie Zheng, and Xiaopan Yang. Energy efficient job scheduling with workload prediction on cloud data center. *Cluster Computing*, 21(3):1581–1593, 2018.

[60] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. 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 2019, Phoenix, AZ, USA, June 25-28, 2019*, pages 315–325. ACM, 2019.

[61] 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. Apache hadoop YARN: yet another resource negotiator. In *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*, pages 5:1–5:16. ACM, 2013.

[62] Bin Wang, Fagui Liu, and Weiwei Lin. Energy-efficient VM scheduling based on deep reinforcement learning. *Future Gener. Comput. Syst.*, 125:616–628, 2021.

[63] Jidong Wang, Peng Li, Kaijie Fang, and Yue Zhou. Robust optimization for household load scheduling with uncertain parameters. *Applied Sciences*, 8(4):575, 2018.

[64] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, pages 945–960. USENIX Association, 2022.

[65] 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. Gandiva: Introspective cluster scheduling for deep learning. In Andrea C. Arpaci-Dusseau and Geoff Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 595–610. USENIX Association, 2018.

[66] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and optimizing GPU energy consumption of DNN training. *CoRR*, abs/2208.06102, 2022.

[67] 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. Hived: Sharing a GPU cluster for deep learning with guarantees. In *Proceedings of 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020*, pages 515–532, 2020.

[68] Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. Multi-resource interleaving for deep learning training. In Fernando Kuipers and Ariel Orda, editors, *SIGCOMM '22: ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, August 22 - 26, 2022*, pages 428–440. ACM, 2022.