Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning - Review

Abdullah Amawi
Computer Network Group
University of Göttingen
Göttingen, Germany
http://www.net.informatik.uni-goettingen.de/

***Abstract*—In this report we will be exploring the importance of deep learning schedulers and how they are crucial for the advancement of high-performance-computing(HPC) clusters schedulers in information technology, which is gaining a lot of traction recently since HPC clusters are important for many applications such as parallel applications, scientific computing, data analytics, artificial intelligence, machine learning, and deep learning it self that Pollux is based on. Because of the importance of HPC clusters, many HPC clusters schedulers exist to try to accommodate the need of managing the compute nodes on a compute cluster, the scheduler manages resources, which jobs run where and when; But the problem lies within the fact that HPC cluster schedulers are not developed in a way that are meant to accommodate deep learning jobs specifically. On the other hand, the very few existing deep learning schedulers are agnostic to the performance scalability of deep learning jobs in regards to the amount of allocated resources [1]. Pollux aims to improve the scheduling performance in deep learning(DL) clusters by co-optimizing inter-dependent factors on the two levels of per-job level and at the cluster-wide level. Not only Pollux tries to address those challenges, but also claims to further improve upon the few existing DL schedulers and reduce the average job completion times by up to 50% [1].**

This report will be based on the works of Qiao et al [1] and the aforementioned Pollux DL cluster scheduler; The Goal is to give a detailed review of the purposed study and to explore the weaknesses and strengths of it.

***Index Terms*—High-Performance-Computing, Deep-learning, Cluster-Schedulers.**

I. Introduction

In recent years, machine learning and specifically deep learning are increasingly becoming on of the top scientific topics both in information technology research and in industry, which is still incrementally advancing everyday; A lot of such advancements are accompanied with the need of doing a big amount of DL training in order to achieve those advancements, on the other hand, we still have many DL training challenges, the main challenge that is addressed in the work of Qiao et al [1] is related to the fact that a big amount of large DL training happens in shared resource environments such as datacenters, HPCs, and the cloud, or in general in compute clusters.

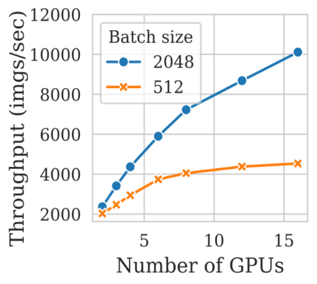
Recently, in order to meet the demand on working with DL jobs in compute clusters and being resource-intensive, running for long periods of times , and in most cases running on expensive hardware such as a graphical processing unit(GPU) or a tensor processing unit(TPU) since they are the specialized hardware for such DL jobs. In order to accommodate such demanding jobs, there are compute clusters that are dedicated for deep learning [2], [3], that have schedulers that are able to share resources between many concurrent DL jobs.

The current main challenge when it comes to existing cluster schedulers is that users are required to manually configure their submitted jobs, which not only requires a lot more knowledge to many factors such as the cluster architecture and capabilities, also when it comes to the DL job itself, if the configuration is done improperly it can lead to sever problems and degradation in training performance and the using the cluster resources efficiently, for example, the user can allocate too many GPUs assuming that those many GPUs will finish the submitted DL job faster, but in reality, in many cases that will result in long queuing times and inefficient resource management and usage; On the other hand, using too few GPUs may result in very long training times and possibly under-utilizing the the resources by leaving some unused; all of those decisions makes it very hard to decide how to submit jobs in a shared-cluster especially with the addition factors that relate not only to the DL jobs but also to the cluster itself as mentioned earlier [1].

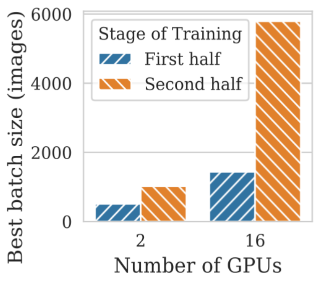
Even though with the recent advancements in schedulers and the introduction of elastic schedulers that can automatically appropriate job resources, they do so without considering the inter-dependant DL training-related configurations such as the batch size, and learning rate that highly affect the amount of resources a DL job needs [1]; In addition to the major challenge that was mentioned earlier, tha the user needs expert knowledge about the cluster hardware performance, on to of the DL model architecture. The work of Qiao et al [1] considers that a properly configured DL job tries to balance two main properties, which are:

- *System throughput*, the number of training examples processed per wall-clock time.
- *Statistical efficiency*, the amount of progress made per training example processed.

System throughput can is increased by increasing the batch size, so, a larger batch size may enable higher utilization of the compute resources, such as using more GPUs; But on the other hand, even with tuned learning rate, the increase of batch size is highly likely to decrease the statistical efficiency [4],



(a) Job scalability (and thus resource utilization) depends on the batch size.



(b) The most efficient batch size depends on the allocated resources and stage of training.

Fig. 1. Trade-offs between the batch size, resource scalability, and stage of training (ResNet18 on CIFAR-10) [1]

[5]. Fig. 1a. Illustrates the increase of system throughput with the increase of batch size, on the other hand, different number of allocated GPUs and the time slot of the training process can change the best batch size in question as Fig. 1b. demonstrates.

Considering all those factors and insights, the work of Qiao et al [1] introduces Pollux, which is a hybrid resource scheduler for DL jobs that is able to c-adaptively allocate resources and tune the batch size and learning rate for all the DL jobs that are in the shared cluster; Pollux does all that by managing both system-level and training-related parameters, including the number of GPUs allocated, per-GPU batch size, gradient accumulation, and learning rate scaling [1]. This paper [1] contribution is summarized by the authors as the following:

- They propose a formulation of *GOODPUT* for DL jobs, which is a measure of training performance that takes into account both system throughput and statistical efficiency.
- They demonstrate that a DL job GOODPUT can be learned by observation during training, then used to predict the performance for different resource allocations and batch sizes.
- They evaluate Pollux on a cluster testbed in comparison to recent DL schedulers, Tiresias [6] and Optimus [7]; They find that Pollux reduces average job completion time by up to 73% and up to 50% even when the competitors jobs are manually tuned beforehand, and Pollux improves finish-time fairness [8] by 1.5X - 5.4X [1].
- Lastly, they show that in cloud environments, Pollux can reduce the cost of training large DL models by up to 25% [1].

II. Background: Distributed DL Training

We know that in order to train a deep learning model, this is typically done by minimizing a loss function, this loss function can be minimized using stochastic gradient descent(SGD) or one of its variants, such as AdaGrad [9] and Adam [10] [1]. The authors used SGD as the example to explain the system throughput and statistical efficiency; SDG applies an update over and over until the loss converges to a stable value [1]; Where the learning rate is a scalar that controls the magnitude of each update while an estimate of the loss function is derived from the stochastic gradient, all evaluated using a random mini-batch, or the batch size [1]. Typically the learning rate and the batch size are the training parameters that are chosen by the user.

A. System Throughput

Qiao et al [1] defines the *system throughput* of DL training as ”the number of training samples processed per unit of wall-clock time”; Moreover, they also note that when a DL job starts across several nodes, several factors affect the system throughput, such as:

- Placement of the resources (e.g. GPUs assigned).
- How the job is distributed for execution and its synchronization.
- The batch size.

Data-parallel execution. *Synchronous data-parallelism* is used, which is a popular method that basically distribute the execution of the DL training by replicating the model parameters across the the set of the distributed available GPUs and the mini-batch is also divided equally across them, each of those GPUs computes a local gradient and then it is all averaged across all GPUs to obtain the desired gradient of all; Each such update will allow each node to obtain new model parameters [1]. Each training iteration, normally called epoch is affected by two main factors. First the time spent computing each gradient; Second, the time spent averaging all the gradients and synchronizing across all GPUs, which itself is affected by the size of the gradient, performance of the network, and usually faster when all GPUs are on the same physical node [1].

Limitations due to the batch size. It is important to note that even if we use many GPUs to compute the gradients, we are still limited by the time for synchronization, this limit is explained by Amdahl’s law, so, the time for sync is the lower bound, usually to overcome this, it is common to increase the batch size which makes the local gradient to be computed over more training examples to increase the ratio of the local gradients to the time of synchronization, which means using a larger batch size results in a higher system throughput when we are scaling using more GPUs in synchronous data-parallel setup [1].

B. Statistical Efficiency

Qiao et al [1] defines statistical efficiency as ”the amount of training progress made per unit of training data processed, influenced by parameters such as batch size or learning rate”, and the authors further explain that a larger batch size usually degrades the statistical efficiency, and being able to predict it makes us able to us that prediction to better adapt the parameters to improve the statistical efficiency itself [1]. **Gradient noise scale.** GNS which measures noise-to-signal ratio of stochastic gradient is related to the statistical efficiency of DL training as previous works show [4], [11]. We can increase the batch size and the learning rate to larger values if we have a larger GNS wit minimal reduction of the statistical efficiency; It is also important to note that GNS varies greatly between different DL models [12].

Learning rate scaling. We have to scale the batch size with the learning rate in a positive manner, meaning that if we increase the batch size, we should also increase the learning rate, otherwise we degrade the DL model quality [1], [4]. However, we have to note that the method of increasing the learning rate differs between different models and training algorithms [1].

C. Existing Dl Schedulers

Qiao et al [1] divides the existing DL schedulers into two categories in order to fit them with the context of Pollux. First, *non-scale-adaptive* schedulers which are neutral to the performance scalability of DL jobs in regards of the allocated resources(e. g. Tiresias [6]). Second, *scale-adaptive* schedulers which decides the amount of allocated resources automatically for each job, depending on the possible acquired speed up for the job, such as Optimus [7] which learns a predictive model for the system throughput for each job in regards of amount of resources. There are other schedulers that have different contributions that Qiao et al [1] mentioned and based some techniques on, such as Gavel [13] which adds a throughput metric across different accelerators. AntMan [3] improves cluster utilization, resource fairness, and job completion times by using dynamic scaling and fine-grained GPU sharing. Themis [8] contributes with the finish-time fairness idea. Pollux utilizes the inter-dependant values and co-adapts them to improve goodput for DL jobs [1].

III. The Goodput of DL Training and Pollux

In this section, we will review the GOODPUT idea and how the authors defined it. goodput of DL jobs is a new measure of training performance that takes into account both the system throughput and statistical efficiency [1], denoted by:

$$GOODPUT_t(*) = THROUGHPUT(*)EFFICIENCY_t(M(*)) \quad (1)$$

, where:

- t: denoting the iteration.
- where * representing any configuration parameters, the authors focus on three parameters(a,m,s) where:
- a: the allocation vector, where a n is the number of GPUs allocated from node n.
- m: the per-GPU batch size.
- s: number of gradient accumulation steps.
- The total batch size is then defined as:
  - $M(a, m, s) = SUM(a) * m * (s + 1)$.

A. Modeling Statistical Efficiency

In the works of Qiao et al [1], the authors consider modeling the statistical efficiency as the amount of progress made per training example while using the total batch size in comparison to the progress made by the initial batch size; In order to express this in SGD-based training, they used previous works that did so in the terms of gradient noise scale(GNS) [4]. On the other hand, the authors also extended support for other variants of SGD, such as Adam [10] and AdaGrad [14]. The authors also noted that their paper supplied the efficiency function that is able to demonstrate that its observation is accurate for different batch sizes and can predict the efficiency which serves a very good guideline.

Upper batch size. It is important to note that the authors also pointed out that their application may define a maximum batch size limit that will be followed by Pollux, because they observed that scaling the learning rate according to the batch size increase may break down, leading to degrading the final quality of the model in use [1].

B. Modeling System throughput

When it comes to modeling system throughput to give a complete picture of both statistical efficiency and system throughput to complete the idea of goodput, and in order to model the system throughput, they noted the main points that they considered [1]. Since the authors are using a data-parallel DL, they considered starting with predicting the time spent per training iteration, then acquiring the throughput by dividing the total on the iterations. This means that they needed to model the local gradient estimates and to model the time for synchronizations, since we are using data-parallelism.

Modelling the gradient. As in any deep learning algorithm, we have the back-propagation process, and the authors used it to estimate the local gradient and noted that the run-time naturally will scale linearly with the per-GPU batch size.

Modeling the synchronization. When we have a single GPU, we know that there is no synchronizations since it is a single GPU on a single node. Otherwise, the authors modeled the synchronization as normally done in data-parallelism, which is scaling the number of GPUs as a linear function [1].

Combining the gradient and synchronization. There is possibility that a DL framework overlaps both the gradient and the synchronization, at least partially, meaning that the network communication would overlap with the computations used for the gradient [15]. So, we have to note that each iteration time is the combined time for computing and the gradient and the synchronizations only assuming that there is no overlap; On the other hand, if there is perfect overlap, then the time taken would be almost equal to the max time of either the gradient or the max synchronization But, realistically, there would be an overlap that is not perfect, nor zero, so a realistic value of the time taken would be in between those two explained scenarios [1].

Gradient Accumulation. Many DL models have a problem with the GPU memory limit that would impair the per-GPU batch size, which would make batch size not large enough to overcome the synchronizations statistically, which will result in degrading the statistical efficiency, therefore, we have to use a technique to overcome this, many exit techniques exist to do so, as noted in multiple previous works [16]–[19]; the authors noted that gradient accumulation is used in Pollux, which can be easily implemented with popular DL frameworks [1].

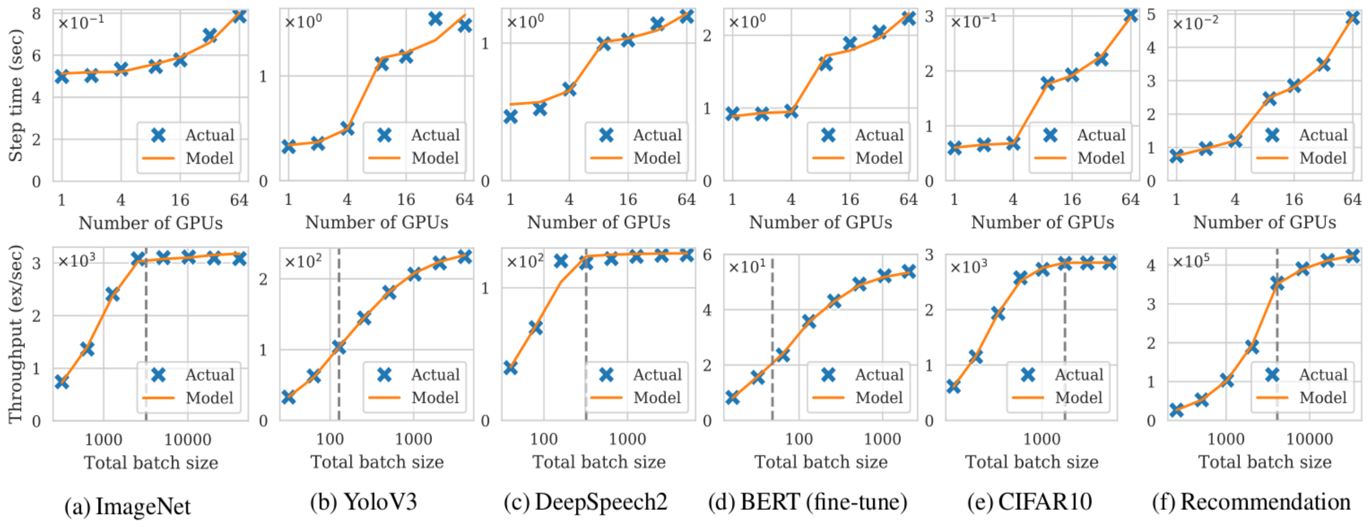


Fig. 2. Throughput modelling in Pollux, the dotted vertical line in the graphs illustrating the throughput per second denotes that on the left hand side of the dotted vertical line, the entire mini-batch fits within the GPU memory, on the other hand, the right side of the dotted vertical line denotes the use of gradient accumulation to achieve achieve the total batch size [1].

Throughput model validation. Fig. 2. Demonstrates how the authors throughput function corresponds to the measured throughput values, and showing it for different range of batch size, resources, and on different Dl models. Pytorch [20] was used in those DL task, which overlaps the computation used for the backward pass and the network communication [1]. The average error for the model was 10% at most, even considering different batch sizes, and different GPU placements in the cluster used [1].

Limits of the throughput model. Moreover, the work of Qiao et al [1] noted that their throughput model does have limits considering that they used some simplistic linear assumptions especially that this may diverge from other cases such as in specialized hardware in reality [**?**], or other synchronization algorithms that can be more sophisticated [22]–[24], other strategies used for parallelism [25]–[28], or even when the scale is larger [29], [30]. The solution to this in Pollux, is that it provides a modular way for others to be able to plug any technique or different additions to suit the case, instead of trying to cover all those scenarios [1].

IV. Pollux Design and Architecture

Pollux design revolves around two levels in the cluster. First the job-level which Pollux tunes the batch size and the learning rate dynamically to utilize the resources better. Second, is the cluster-wide optimization that Pollux does by allocating and re-allocating resources in the cluster to achieve better goodput [1]. Fig. 3. illustrates Pollux design, architecture and how those two components mentioned above interact. Fig. 3. also explains the two major components that Pollux revolve around. First, the PolluxAgent which is ran with each job to fit the previously mentioned efficiency and throughput functions for the job. Second, the PolluxSched, which is the scheduler that is concerned with the optimization with the resources in

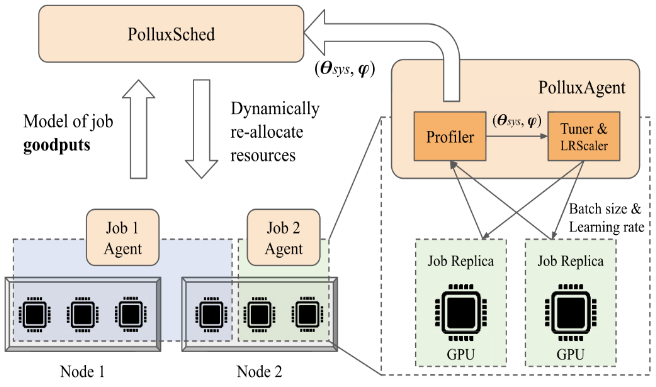


Fig. 3. Pollux co-adaptive scheduling architecture) [1]

the cluster-wide scope for all the jobs that are running to achieve better goodput and to take into account the cluster-wide resource contention [1]. The authors also note that both of the PolluxSched and PolluxAgent co-adapt to each other in a way that while the agent is making each job utilize the resources more efficiently, the scheduler is re-allocating the resources dynamically while taking the agent into accout.

A. *PolluxAgent: Job-level Optimization*

Qiao et al [1] work also mentions in detail how the job-level optimizations work; But I will summarize that as an abstract overview explanation. It basically explains that the PolluxAgent is started with each job, it measures the gradient noise scale and the system throughput during the training for each job then reports it to the PolluxSched [1]. Moreover, it uses this information to better understand the job so it can decide the best suited batch size and learning rate [1].

B. PolluxSched: Cluster-wide Optimization

Qiao et al [1] work also mentions in detail how the cluster-wide optimizations work; I will also summarize that as an abstract overview explanation. For every job in the cluster, the PolluxSched is always either allocating or re-allocating the resources to utilize them better in regards of the goodput. The paper [1] also explains how PolluxSched is also maintaining fairness between the jobs, and also applies a re-allocation penalty to avoid excessive re-allocations.

V. Evaluation

In the section, I will summarize how the authors presented their evaluation and focus only on the main parts of the evaluation and some of the main results presented by the work of Qiao et al [1]. The authors note that they utilized a testbed of 64 GPUs to compare Pollux with two state-of-the-art DL schedulers, they note that Pollux reduces the job completion time by up to 50% even though that they used a scenario that favors the previous SOTA schedulers, since they used Pollux which is dynamic in adaptation against two SOTA schedulers that have well-tuned job configurations [1]. Moreover, they also note that not only Pollux is faster in job completion times, but also that it had big improvements in finish-time fairness [8], and has the potential of reducing the training for large DL models by up to 25%.

A. Experimental Setup

Testbed. The experiments are conducted in AWS EC2 cluster that has 16 nodes, each of which has 4 NVIDIA T4 GPUs, which accumulates to 64 GPUs, 192GB memeory, 48 vCPUs, and all of it is conducted on SSDs.

Synthetic Workload Construction. Based on 160 jobs that were randomly sampled in the busiest 8-hours in the deep learning cluster, each job had multiple data that describes the information of the job, such as the submission time, duration, and number of the GPUs used. Moreover, since there is no information regarding the datasets, and the models being utilized, the models and datasets that were utilized as the synthetic workload are described in Table 1. The jobs were categorized based on their total GPU-time as Small(0–1 hours), Medium(1–10 hours), Large(10–100 hours), XLarge(100–1000 hours), that being GPU hours.

Manually-tuned jobs for baseline DL schedulers.

For the previous SOTA DL schedulers, the authors manually-tuned each job for the batch size and the number of GPUs, they did so by utilizing different ranges of batch sizes and number of GPUs and did a full training for each model using them, then, they considered the number of GPUs to be used valid if they see that increasing them with the corresponding batch size will result in a perfectly linear scalability versus using a single GPU with an optimal batch size [1]. The authors did all that additional work for the previous SOTA Dl schedulers to give them much more favor versus Pollux which does not need any of this extra manual-tuning since it dynamically allocates and re-allocates resources, but by doing so, they show that Pollux is very strong even when compared to unrealistically tuned SOTA DL schedulers.

Policy	Job Completion Time Average	Makespan
Pollux (p = -1)	**0.76h**	**16h**
Optimus+Oracle+TunedJobs	1.5h	20h
Tiresias+TunedJobs	1.2h	24h
Optimus+Oracle	2.7h	28h
Tiresias	2.8h	31h
Pollux (p = +1)	0.83h	**16h**
Pollux (p = -10)	0.84h	18h

TABLE I
SUMMARY OF THE EXPERIMENTS OF COMPARING POLLUX TO OPTIMUS AND TIRESIAS [1].

Comparison of DL schedulers.

In comparison to two recent DL schedulers, Tiresias [6] and Optimus [7], Pollux, as mentioned earlier is dynamic and co-adapts the number of GPUs and batch sizes of the DL training jobs. On the other hand, Tiresias does not adapt any of the two, and Optimus adapts on the number of GPUs used [1].

B. Testbed Macrobenchmark Experiments

Pollux was compared to the other two SOTA DL schedulers in seven configurations, at the start, Pollux with a "fairness nob" value of -1, which is the default value, the fairness knob denoted by p can be changed with a larger negative being more fair, but the authors find that p = -1 results in the best goodput improvements; On the other, we have two values of the two SOTA DL schedulers in the manually-tuned unrealistic configuration in their favor, then two additional realistic configurations of the same DL schedulers Optimus and Tiresias, then lastly, Pollux with different p values. Table 1. summarizes the average job completion time and the makespan time taken.

Comparisons using well-tuned job configurations. We note that from the results, even when Optimus and Tiresias are using a manually well-tuned job to test, Pollux is still performing much better and resulting in 50% ad 37% shorter average job completion time respectively. Moreover, it is resulting in 20% and 33% shorter makespan against Optimus and Tiresias [1]; And, as it was noted earlier, this even highly favors the baseline schedulers against Pollux, which shows the strength of Pollux even further.

Comparisons using realistic job configurations.

Realistically, without the use of Pollux, users have to configure their own jobs by just doing a number of attempts on which batch size, learning rate, and number of GPUs the job should utilize before deciding on which configuration is the most efficient in their tests. On the other hand, some users don't even invest the time to do so.

To make the comparison with Pollux more realistic, the authors decided that they use a better baseline than what was mentioned above, by running versions of Optimux+Oracle and Tiresias workloads in their synthetic benchmarks, and choosing the amount of GPUs exactly how it was specified in Microsoft cluster trace [31]. This workload resulted in Pollux having 72% and 73% shorter average job completion time,

Task	Dataset	Model	Optimizedr	LR Scaler	M0	Validation	Size	Frac.Jobs
Image Classification	ImageNet [34]	ResNet-50 [38]	SGD	AdaScale	200 imgs	75% top1 acc.	XL	2%
Object Detection	PASCAL-VOC [33]	YOLOv3 [39]	SGD	AdaScale	8 imgs	84% mAP	L	6%
Speech Recognition	CMU-ARCTIC [34]	DeepSpeech2 [40]	SGD	AdaScale	20 seqs	25% word err.	M	10%
Question Answering	SQuAD [35]	BERT (finetune) [41]	AdamW	Square-Root	12 seqs	88% F1 score	M	10%
Image Classification	Cifar10 [36]	ResNet18 [38]	SGD	AdaScale	128 imgs	94% top1 acc.	S	36%
Recommendation	MovieLens [37]	NeuMF [42]	Adam	Square-Root	256 pairs	69% hit rate	S	36%

TABLE II
MODELS AND DATASETS USED IN OUR EVALUATION WORKLOAD. EACH TRAINING TASK ACHIEVES THE PROVIDED VALIDATION METRICS. THE FRACTION OF JOBS FROM EACH CATEGORY ARE CHOSEN ACCORDING TO THE PUBLIC MICROSOFT CLUSTER TRACES.

and 43% and 48% shorter makespan than Optimus+Oracle and Tiresias, respectively. On the other hand, the authors note that even though Optimus+Oracle does change the number of the GPUs dynamically, this change doesn't have a big impact in comparison to Tiresias that doesn't change the number of GPUs, and the reason behind this is that Optimus does not increase the batch size alongside increasing the number of GPUs for better utilization [1].

A closer look at co-adapted job configurations.
Fig. 4. explains how Pollux co-adapts the resources over time, the left-hand side of the figure illustrates how it works on one of the ImageNet jobs, on the right-hand side the figure illustrates how it co-adapts the resources on YOLOv3 jobs [1]. Fig. 4. explains the author findings regarding co-adapting in three stages as follows:

- In the ImageNet example on the left, in the initial low cluster contention, Pollux allows the job to assign more GPUs because of the low contention, which results in lower statistical efficiency(A).
- When we have high cluster contention, fewer GPUs are re-allocated to the ImageNet job, which improves the statistical efficiency, but lowers the batch size(B).
- When cluster contention comes back down again, ImageNet task will again get more GPUs and higher batch size re-allocated, but the authors note that this time, since the training is in late stages, the batch size per GPU is much higher than in the first stage.

Fig. 4. also demonstrates similar trends regarding YOLOv3 jobs on the right-hand side.

Effect of the fairness knob. The authors note that the "fairness knob" value of p that they utilized were either 1, -1, or -10, with 1 being without any added fairness, -1 being the value of moderate fairness, and -10 would further increase the fairness. Depending on the job, the fairness knob value can have different effects, the authors found out that in the case of ImageNet, moderate degree of fairness, which is -1 yielded the best results, because without fairness, ImageNet would use too many GPUs, on the other hand, increasing the fairness value too much to -10 would lead to performance degradation in job time completion and makespan. The authors present a full analysis for the reason behind that in their work [1].

System overheads. When it comes to PolluxSched it would of course have some system overhead because of the resource re-allocations, the authors found that the jobs re-allocated resources each 7 minutes, and due to the checkpoints-restarts, the run-time overhead was 8% on average; Finding the optimal parameters for optimizing the goodput on the other hand, such as the batch size and gradient steps took 0.4 milliseconds [1].

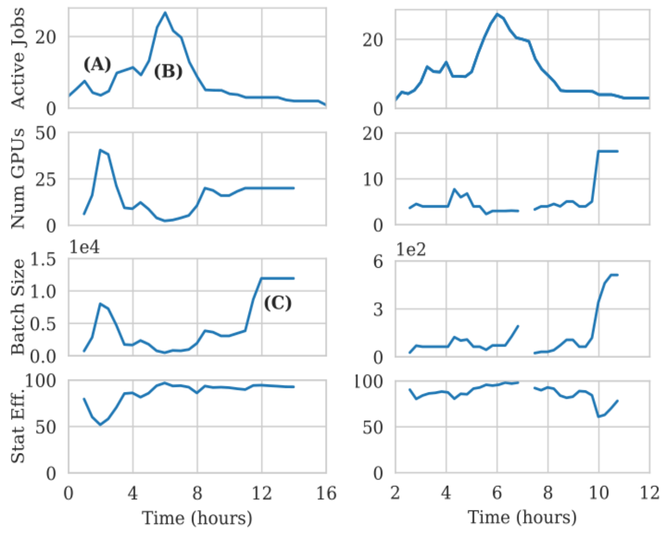


Fig. 4. Pollux resource co-adaptive ImageNet(left) and two YOLOv3 jobs(right) [1]

C. Simulator Experiments and other Applications

For the sake of providing a complete overview of the aforementioned work [1], we note that the work of Qiao et al. provides a lot more experiments and applications of Pollux, such as implementing a cluster simulator to further evaluate other areas such as scheduling fairness, other effects on scheduling, and more. On the other hand, Pollux is used in a wide range of other applications, such as cloud autoscaling, and the authors explain how they used their cluster simulator to improve how cloud provisioning works and how it can adapt resource provisioning further using the findings of Goodput-driven scheduling and Pollux; for example, how we noted earlier that later stages in a training job can use a lot more resources and larger batch sizes, this can be used in a way that the cloud environment can provision fewer resources for better statistical efficiency in the beginning of the job, and later on scales more with more GPUs and resources for faster training in the later stages, which would reflect how the resources provisioning should utilize Pollux. The authors also provide other details on how hyper-parameter optimization can be a future work and another application of use for Pollux.

VI. Related Work

While adaptive deep learning cluster scheduling have been researched and used consistently in the industry, and many of the industry giants have to utilize cluster scheduling for their data centers, Qiao et al. [1] main contribution was regarding that their Pollux solution co-adapted batch size, learning rate, and number of GPUs. But, when it comes to the base ideas that Pollux is based upon, the authors referred to multiple works when it comes to previous works in the area of deep learning schedulers [3], [6], [8], [13], [43], [44], but, none of them addressed how to dynamically co-adapt the DL parameters similarly to how Pollux did it, Optimux only adapted the number of GPUs [7]. Secondly, many other works in the scope of adaptive batch size training [45]–[48], but their works either assume unrealistic linear increase for the resources, and only KungFu [49] has adaptive training, but only for single-job training, not cluster scheduling. Lastly, when it comes to hyper-parameter tuning many works are mentioned [46], [50]–[56], but they only focus on the model quality, on the other hand, Pollux maintains the model quality and also is the most efficient [1].

VII. Opinion about Paper

While reviewing the work of Qiao et al [1], we identified the following strengths and weaknesses in the paper, that we will note in this section.

A. Strengths

While reviewing the aforementioned work, we can clearly identify a couple of major strong points when it comes to the paper, we think that the most important point in here is that the idea of the paper is a novel idea, that actually could be the future of deep learning scheduling in cluster-wide scenarios, no other previous work utilized co-adaptive, dynamic cluster-wide deep learning scheduling in such a manner. Moreover, when it comes to the paper quality itself, we find that the paper is well-made, demonstrating high-quality presentation and readability, it has a very clear explanations that are well-put together, not only when it comes to deep learning scheduling and the core ideas, but also it gives the reader a lot more info and background it various topics that are hot discussions in deep learning, and gives the reader much better understanding of the topic. Overall, this paper clearly explains the background of the topic with very good details, and contributes a lot to that and the novel idea that the authors present, providing the full implementation and details in github, and all of this clearly demonstrates many strengths of this valuable work, which leads to taking the title of being the best paper in a major conference, the 15th USENIX conference.

B. Weaknesses

Even when it comes to great works such as the work of Qiao et al. [1], almost always there is room for improvement, this can be a seen a good thing from an angle, but it is also worth to try to critically explore the weaknesses of any work, which is what we will try here. When it comes to the paper presentation itself, the reader can notice that there is no separation between clear future and current applications, but we believe that it should have been more clear which is a current application of Pollux, and which is future work. On the other hand, when it comes to deep learning jobs, there are some points in the paper that are neglected or not given the right amount of focus that we believe they should have, for example, when it comes to the GPUs used(NVIDIA T4), those are relatively old GPUs in comparison to the current or more recent offerings, this major downside can impact the findings, even though the NVIDIA T4 has a good amount of tensor cores which are very suitable for deep learning work, it struggles with an older type of GDDR 6 memory that provides much slower memory bandwidth than the current offerings, this would impact the batch size and the ability to scale better in a cluster-wide environment, which is the targeted setting in this scenario, on the other hand, being a GPU of an older architecture and having few NVIDIA CUDA Cores in comparison to recent offerings would also impact the performance, which would change the amount of needed GPUs for each job, since newer, faster GPUs can do the same job with less GPUs, which leads us to think about how would that impact the scheduling of Pollux.

VIII. Conclusion and Future Work

While reviewing the works of Qiao et al. [1], and previously pointing out the strengths and weaknesses of the aforementioned work, even with the possibility of having some weakness that needs to be further investigated if they hold or not, the paper does what it claims in presenting Pollux, being a DL-aware cluster scheduler that dynamically co-adapts the resources for each job in the cluster while consider all the cluster-wide environment, which is a great contribution to DL cluster schedulers. Moreover, Pollux indeed has a huge training performance improvement impact in comparison to previous works in share cluster settings, and improves the training time by up to 50% even when we compared it heavily-tuned DL schedulers, which is an unrealistic scenario that tilts the comparison against Pollux, but even then, Pollux still improves the training time while not needing all that time and effort from experts to further tune the DL jobs that they submit into the cluster, which makes Pollux not only great in performance, but also one of a kind in DL cluster schedulers.

Future Work. While we believe that works of Qiao et al. [1] is a great addition to DL schedulers and may be the right direction in this field, and could be the basis for the future of it, we believe there is good room for future works, some of which the authors mentioned, such as an intended cloud auto-scaling system based on GOODPUT, and a full evaluation of Pollux affects of different hyper-parameter algorithms. On the other hand, we would add that it could also provide a Tensorflow implementation being the industry standard for production environments in contrast to Pytorch in research, the usage of different, more capable GPUs as we mention as a weakness in the work, and the possibility for more comparisons between different cloud environments to evaluate how does the different environment affect the impact of Pollux.

References

[1] Aurick Qiao and Sang Keun Choe and Suhas Jayaram Subramanya and Willie Neiswanger and Qirong Ho and Hao Zhang and Gregory R. Ganger and Eric P. Xing, "Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning' 2021.

[2] 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 2019 USENIX Annual Technical Conference (USENIX ATC 19), pages 947–960, Renton, WA, July 2019. USENIX Association.

[3] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. Antman: Dynamic scaling on GPU clusters for deep learning. In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), pages 533–548. USENIX Association, November 2020.

[4] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. CoRR, abs/1812.06162, 2018.

[5] Christopher J. Shallue, Jaehoon Lee, Joseph M. Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. CoRR, abs/1811.03600, 2018.

[6] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), pages 485–500, Boston, MA, February 2019. USENIX Association.

[7] 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 Thirteenth EuroSys Conference, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[8] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient GPU cluster scheduling. In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), pages 289–304, 2020.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(61):2121–2159, 2011.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[11] Tyler B. Johnson, Pulkit Agrawal, Haijie Gu, and Carlos Guestrin. Adascale sgd: A scale-invariant algorithm for distributed training, 2020.

[12] Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W. Mahoney, and Joseph Gonzalez. On the computational inefficiency of large batch sizes for stochastic gradient descent. CoRR, abs/1811.12941, 2018.

[13] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In 14th USENIX Symposium on Op- erating Systems Design and Implementation (OSDI 20), pages 481–498. USENIX Association, November 2020.

[14] Rachel Ward, Xiaoxia Wu, and Leon Bottou. Ada- grad stepsizes: Sharp convergence over nonconvex landscapes. In International Conference on Machine Learning, pages 6677–6686. PMLR, 2019.

[15] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P. Xing. Poseidon: An efficient communi- cation architecture for distributed deep learning on GPU clusters. In 2017 USENIX Annual Technical Conference (USENIX ATC 17), pages 181–193, Santa Clara, CA, July 2017. USENIX Association.

[16] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. CoRR, abs/1604.06174, 2016.

[17] Henggang Cui, Hao Zhang, Gregory R. Ganger, Phillip B. Gibbons, and Eric P. Xing. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In Proceedings of the Eleventh European Con- ference on Computer Systems, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery.

[18] Chien-Chin Huang, Gu Jin, and Jinyang Li. Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '20, page 1341–1355, New York, NY, USA, 2020. Association for Computing Machinery.

[19] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gho- lami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. Checkmate: Breaking the memory wall with optimal tensor rematerialization. In I. Dhillon, D. Papail- iopoulos, and V. Sze, editors, Proceedings of Machine Learning and Systems, volume 2, pages 497–511, 2020.

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zem- ing Lin, Natalia Gimelshein, Luca Antiga, Alban Des- maison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chin- tala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8026–8037. Curran Associates, Inc., 2019.

[21] N. Jouppi, C. Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, R. Bajwa, Sarah Bates, Suresh Bhatia, N. Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, M. Dau, J. Dean, Ben Gelb, T. Ghaemmaghami, R. Gottipati, William Gulland, R. Hagmann, C. Ho, Doug Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, N. Kumar, Steve Lacy, J. Laudon, James Law, Diemthu Le, Chris Leary, Z. Liu, Kyle A. Lucke, Alan Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, Ravi Narayanaswami, Ray Ni, K. Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, A. Phelps, J. Ross, Matt Ross, Amir Salek, E. Samadiani, C. Severn,G. Sizikov, Matthew Snelham, J. Souter, D. Steinberg, Andy Swing, Mercedes Tan, G. Thorson, Bo Tian, H. Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, W. Wang, Eric Wilcox, and D. Yoon. In-datacenter performance analysis of a tensor processing unit. 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pages 1–12, 2017.

[22] J. Canny and Huasha Zhao. Butterfly mixing: Accelerat- ing incremental-update algorithms on clusters. In SDM, 2013.

[23] Jinliang Wei, Wei Dai, Aurick Qiao, Qirong Ho, Heng- gang Cui, Gregory R Ganger, Phillip B Gibbons, Garth A Gibson, and Eric P Xing. Managed communication and consistency for fast data-parallel iterative analytics. In Proceedings of the Sixth ACM Symposium on Cloud Computing, pages 381–394, 2015.

[24] Huasha Zhao and J. Canny. Kylix: A sparse allreduce for commodity clusters. 2014 43rd International Conference on Parallel Processing, pages 273–282, 2014.

[25] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019.

[26] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP '19, page 1–15, New York, NY, USA, 2019. Association for Computing Machinery.

[27] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake Hechtman. Mesh- tensorflow: Deep learning for supercomputers. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 31. Curran Associates, Inc., 2018.

[28] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. ArXiv, abs/1909.08053, 2019.

[29] 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. Language models are few-shot learners. In H. Larochelle, M.

Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[30] Masafumi Yamazaki, Akihiko Kasagi, Akihiro Tabuchi, Takumi Honda, Masahiro Miwa, Naoto Fukumoto, Tsuguchika Tabaru, Atsushi Ike, and Kohta Nakashima. Yet another accelerated sgd: Resnet-50 training on imagenet in 74.7 seconds, 2019.

[31] 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 2019 USENIX Annual Technical Conference (USENIX ATC 19), pages 947–960, Renton, WA, July 2019. USENIX Association.

[32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vi- sion and pattern recognition, pages 248–255. Ieee, 2009.

[33] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. International Journal of Computer Vision, 88(2):303–338, June 2010.

[34] John Kominek and Alan Black. The cmu arctic speech databases. SSW5-2004, 01 2004.

[35] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[36] Alex Krizhevsky. Learning multiple layers of features from tiny images. University of Toronto, 05 2012.

[37] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst., 5(4), December 2015.

[38] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learn- ing for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016.

[39] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. CoRR, abs/1804.02767, 2018.

[40] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Vaino Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16, page 173–182. JMLR.org, 2016.

[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirec- tional transformers for language understanding, 2019.

[42] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filter- ing. In Proceedings of the 26th International Conference on World Wide Web, WWW '17, page 173–182, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[43] 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: Intro- spective cluster scheduling for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pages 595–610, Carlsbad, CA, October 2018. USENIX Association.

[44] Haoyu Zhang, Logan Stafman, Andrew Or, and Michael J. Freedman. Slaq: Quality-driven scheduling for distributed machine learning. In Proceedings of the 2017 Symposium on Cloud Computing, SoCC '17, page 390–404, New York, NY, USA, 2017. Association for Computing Machinery.

[45] Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks. CoRR, abs/1712.02029, 2017.

[46] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. CoRR, abs/1711.00489, 2017.

[47] Lukas Balles, Javier Romero, and Philipp Hennig. Coupling adaptive batch sizes with learning rates. CoRR, abs/1612.05086, 2016.

[48] Nuwan Ferdinand, Haider Al-Lawati, Stark Draper, and Matthew Nokleby. ANYTIME MINIBATCH: EXPLOITING STRAGGLERS IN ONLINE DIS- TRIBUTED OPTIMIZATION. In International Conference on Learning Representations, 2019.

[49] Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis, Andrei-Octavian Brabete, and Peter Pietzuch. Kungfu: Making training in distributed machine learning adaptive. In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), pages 937–954. USENIX Association, November 2020.

[50] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In Advances in neural information processing systems, pages 2546–2554, 2011.

[51] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In Advances in neural information processing systems, pages 2962–2970, 2015.

[52] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In International conference on learning and intelligent optimization, pages 507–523. Springer, 2011.

[53] Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R Collins, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. arXiv preprint arXiv:1903.06694, 2019.

[54] Willie Neiswanger, Kirthevasan Kandasamy, Barnabas Poczos, Jeff Schneider, and Eric Xing. Probo: a frame- work for using probabilistic programming in bayesian optimization. arXiv preprint arXiv:1901.11515, 2019.

[55] Introduction to katib. org/docs/components/hyperparameter-tuning/overview/. Accessed: 2020-05-18.

[56] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. Google vizier: A service for black-box optimization. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, page 1487–1495, New York, NY, USA, 2017. Association for Computing Machinery.