Contents

First of all, we would like to extend our appreciation to the reviewers for their valuable comments on our paper. We have carefully reviewed all the comments. If we have a chance of the second round modification, we will revisions according to these comments. In this document, I will explain the concerns of the reviews which are summarized as follows.

1. Reviewer 1

> The paper is exceptionally hard to follow. Even though experimental results seem to indicate that the proposed framework outperforms existing state-of-the-art schedulers, I really have a very hard time following the design and making a good sense of the described contribution.

Thank you for the comments of our paper manuscript. We will revision the architecture of the papers and describe the design of the Qore-DL more clearly. For example, we will remove the unnecessary equations and use the clear text description to present the Qore-DL. I will describe the training process of DLT jobs and summarize the inherent characteristics of DLT jobs before the specific design of Qore-DL. Now let me describe the DLT job, system design and the contribution of Qore-DL generally. More code, data and information can be found at: https://github.com/qore-dl/qore-dl-code

(1) System Design Description

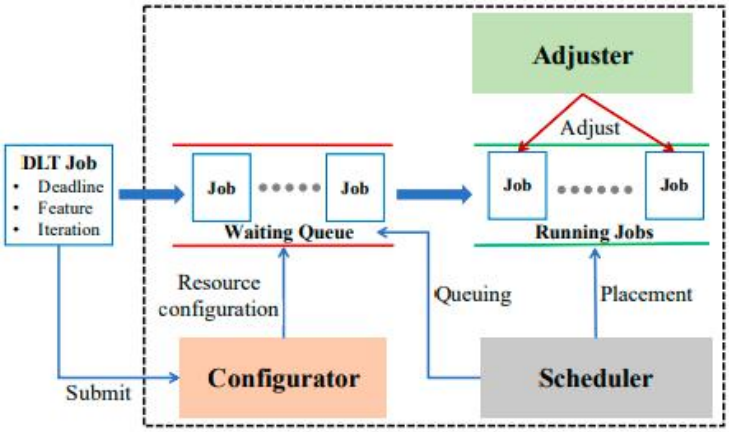


Fig. 1: The framework of Qore-DL.

Figure 1:The framework of Qore-DL

Let we review the Figure 1 in the paper. Qore-DL consists of three modules: **configurator, scheduler** and **adjuster.** The design of the three modules aims to jointly optimize the resource configuration of the DLT jobs in the three stages (**submission, queuing** and **running**) in DLT-jobs' lifecycle. All the optimizations made by Qore-DL is bi-objective: QoS satisfaction and cluster resource efficiency. First of all, let me explain the DLT jobs.

1) DLT job:

DLT job means the deep learning training jobs. Each DLT job consists of multiple instances. For example, a DLT job in the PS-framework have two types of instances: Parameter Server (PS) and the worker. The DLT jobs runs iteratively as shown in Figure 2 in the paper manuscripts.

In our paper, for each iteration, a batch size (e.g., 32) of samples in the datasets are inputted

to each worker in the DLT job. Each worker have the same model with millions of trainable parameters. Then Each worker computes the gradient of the trainable parameters by the forward propagation and back propagation, which consist of a large number of floating-point calculation operations. In PS framework, the trainable parameters are divide into multiple groups. In each iteration, each PS aggregates the gradients of different part of the trainable parameters in the model from all the workers. At the end of the iteration, PSs send the aggregated gradient to the workers for synchronization and workers update the trainable parameters based on the aggregated gradient.

Therefore, we summarize that the training model of each DLT job have three inherent **features**: **batch size, the number of the floating-point calculation operations and the number of the trainable parameters**. All these three features is extracted by the Qore-DL automatically.

2) Three stages:

In our paper, we divide the lifecycle of a DLT job into three general stages: **submission, queuing** and **running**. For a DLT job, 1) after users submit the the job with some initial configuration, the DLT jobs proceeds to the **submission** stage; 2) after Qore-DL completes the initial resource configuration of the DLT job, Qore-DL push this job into the waiting queue, and this DLT job proceeds to the **queuing** stage; 3) after the Qore-DL schedule the DLT job into the cluster, the DLT job executes the model training and proceeds to the **running** stage until the completion of the DLT job. We designed three modules corresponding to the three stages respectively. Qore-DL address the QoS-aware optimization of DLT jobs. This means that Qore-DL attempts to complete the DLT jobs before the user-specified deadline to guarantee the deadline satisfaction and considers the resource efficiency of the cluster. Now let me describe the three stages of DLT jobs and the three modules of Qore-DL generally.

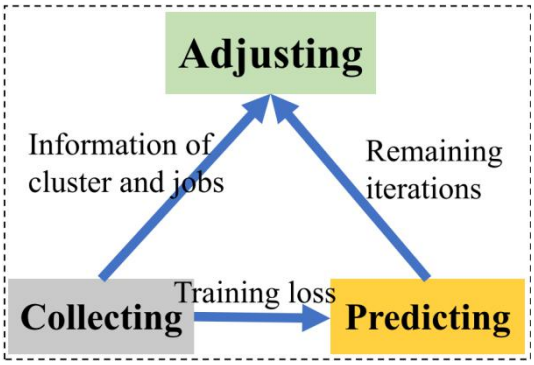
3) The design and execution of the three modules

We can use specific cases to present the procedure of the three modules. As shown in Figure 1, after the user submits a DLT job to the cluster, this DLT job proceeds to the **submission** stage. For users, Qore-DL aims to simplify the submitting operations. This means that users only need to specify two user-level configuration item: 1) the **deadline** of the DLT job and 2) the upper limit of the number of **iterations** of the DLT job, which can be regarded as the expected number of **iterations** when the DLT job only have 1 worker. In the **submission** stage, the Qore-DL first extracts the three inherent **features** from the DLT job: **batch size, the number of the floating calculation operations and the number of the trainable parameters**. Then the **configurator** module will make two configurations: 1) use the GBDT model to predict the resource quota (or limit) of the worker and the PS of the DLT job based on the inherent features; 2) use the random forest model to predict the average iteration time (duration of executing the training iteration) and compute the number of PSs and workers according to the predicted iteration time, **deadline** and **the upper limit of the number of iterations** of the DLT job. Then the configured DLT jobs will enter the waiting queue and proceeds to the **queuing** stage.

The **scheduler** module will schedule the jobs in the **queuing** stage. When selecting the target job for scheduling, the priority of the **queuing** jobs is computed according to the deadline and the dynamic cluster resource availability as shown in Eq.(8) in the paper manuscripts. This priority computation reflect the dual-optimization of the deadline satisfaction and resource efficiency. We can summarize to principles when quantifying the priority: 1) deadline satisfaction: the queuing job with a urgent deadline will have a high priority; 2) resource efficiency: when the cluster has a

light workload, the jobs with a large resource requirement may have a higher priority, and when the cluster has a heavy workload, the jobs with a small resource requirement may have a higher priority. Eq.(8) is the quantification of these principles for the priority computation. After the scheduling target selection, the greedy scheduling algorithm of the Qore-DL is shown in the Algorithm 1 in the paper manuscripts. After placing the jobs in the cluster, the job proceeds to the **running** stage.

For **adjuster**, the cluster is highly dynamic and the resource configuration and scheduling decision is not the globally optimal solution of the NP-hard problem (i.e., the dual optimization problem of deadline satisfaction and resource efficiency). Therefore, the resource configuration of the **running** DLT jobs need to be adjusted. The **adjuster** has to realize three functionalities: 1) **collecting** the resource and loss trace of the DLT jobs; 2) **predicting** the remaining iterations using the LSTM model; and 3) **adjusting** the resource configuration of DLT jobs. The **collecting** and **predicting** provides the information to the adjusting functionality.



In the **adjusting** process, the adjuster select the target jobs based on the Section 4.D.1) in the manuscript as:

Then, we compute the $S_i = \max\left(S_i^1, S_i^0\right)$ of τ_i as the maximum of the two values corresponding to the two adjustment methods: S_i^1 obtained by increasing or decreasing one worker and S_i^0 obtained by increasing or decreasing $\frac{r_{ij}^w}{N_i^w}$ resources to each worker for $j \in J$.

Finally, Qore-DL selects n jobs in the four cases according to the guidelines and the normalized cluster resource utilization U: 1) existing timeout jobs and $U \leq \theta_2$: selecting the timeout jobs with the highest S_i; 2) existing timeout jobs and $U > \theta_2$: selecting the timeout jobs with the highest S_i to increase resources and selecting the deadline-redundancy jobs ($O_i < 0$) with the lowest S_i to preempt resources; 3) no timeout jobs and $U \leq \theta_1$: selecting the jobs with the highest S_i to increase resources; and 4) no timeout jobs and $U > \theta_3$: selecting the jobs with the lowest S_i to reduce resources for future jobs.

At last, we use the heuristic genetic algorithm to generate the resource adjustment solutions with highest fitness defined in Eq.(15) in the paper manuscripts.

(2) System Contribution Description:

1) Bi-objective optimization: first of all the Qore-DL is QoS-aware, the optimization in the three stages are all QoS-oriented to guarantee the deadline satisfaction. On this basis, Qore-DL considers the resource efficiency optimization, which is reflected at: 1) the scheduling priority and algorithm in Section IV.C.1 and IV.C.2; 2) the adjusting scheme in Section IV.D: (i) design 4 cases when job selection; (ii) increasing resource quota of the jobs with high efficiency and preempt the resource of jobs with poor efficiency and redundant deadline; and (iii) generate adjusting solution

with high marginal benefit of resource efficiency and deadline satisfaction.

2) Three-stage joint optimization: according to the system design description above, the three modules in the Qore-DL realize the bi-objective optimization in the three stages of the DLT jobs: (i) the scheduler and adjuster leverage the configuration or prediction results of the configurator; and (ii) adjuster relieve the limitation of the scheduler and configurator caused by the dynamic property of the clusters.

3) Implementation and the Evaluation: We implement the prototype in the real clusters and execute realistic training studies in the real cluster. We are developing the Qore-DL to the production cluster of the cooperation company now.

> The referenced github code repository is not helpful either and the code is poorly commented, with many comments written in con-English language, and lengthy sections of the code commented out.

(3) Refactor the code and add the comments

We are developing the Qore-DL to the production cluster of the cooperation company and now we refactor some codes including the frontend, monitor, flask server and backend controllers. As for the experiments codes in the github, we are adding the comment to describe the role of the functions. With the project advancement, we will update the detailed document as well. More code, data and information can be found at: https://github.com/qore-dl/qore-dl-code .

2. Reviewer 2

Thank you for the comments of our paper manuscript. Now we are developing the Qore-DL in the production cluster and attempt to support the non-PS framework and Pytorch. We will tune the parameter settings during the test and deploy in the production cluster. More code, data and information can be found at: https://github.com/qore-dl/qore-dl-code .

3. Reviewer 3

Thank you for the comments of our paper manuscript. I will explain the main concern in this document.

More code, data and information can be found at: https://github.com/qore-dl/qore-dl-code .

> The primary weakness of the current manuscript is clarity in both the approach and the experiments. The paper describes about six pages worth of equations making a number of assumptions about both the resources involved and the applications. The description is hard to follow and unnecessarily complicated. The assumptions are not clearly stated and in numerous places.

We will remove the unnecessary equations in revisions of the paper manuscripts. For example, the Definition 1 and Definition 3 in the Section III of paper manuscript use the equation to define the DLT job and the QoS satisfaction. We will use the text description to define them. For the DLT job, we have summarize the features of a DLT job including: 1) **inherent features**: **batch size, the number of the floating calculation operations and the number of the trainable parameters;** 2) the **deadline** of the DLT job and 3) the **the upper limit of the number of**

iterations of the DLT job. For the QoS satisfaction of a DLT job, we define it as that the DLT job complete the **iterations** of training or achieve the convergence before the user-specified **deadline**.

We made some assumption when define and solve the problem. These assumption about both the resources involved and the applications.

In summary, we make two assumptions on resource to quantify the job resource requirements and the cluster utilization, and these two assumptions are the formalized description of running realistic DLT jobs and the real system.

As for the assumptions of the application, we utilize the real characteristics of DLT jobs, including iterativeness, convergence and decline rate. These are not assumptions, as verified by many works (such as Gandiva[2], Optimus[5] and SLAQ [6]).

Now let we explain the assumptions on resource and applications in the paper manuscript as follow:

(1) Assumption of resources involved:

1) weighted cluster utilization score U and dominant resource utilization of host H_l, U_l:

These variables are defined in the Eq.(1) in the manuscript. Actually, U is not the average resource utilization but is a metric or score to quantify the pressure of the cluster workload, so we compute it in a weighted way and give a larger weight to hosts(machines) having lighter workloads. This is because that when scheduling and adjusting the DLT jobs, the host (machine) with a lighter workload will have a larger resource capacity to place the containers. Actually, our scheduler and adjuster tends to leverage the resource of the machines which have lighter workload. Meanwhile, the weights of H_l is in the range of (0,1) and the sum of the weights is 1, which guarantee that the weighted cluster utilization score U is positive related to the actual arithmetic mean value of the machine resource utilization in the cluster.

As for the U_l, i.e., the dominant resource utilization of the host H_l in the cluster. Firstly, we explain the dominant resource. In the manuscript, we reference the idea of dominant resource type from the Dominant Resource Fairness (DRF) [1] scheduling strategy. In DRF, the dominant resource type of a job is defined as the resource type whose quota of the job has the largest percentage of total cluster resource of this type. For example, for a job A, CPU quota has the 10% of the total CPU capacity of the cluster and Memory quota has the 5% of the total Memory capacity of the cluster, then CPU is the dominant resource type of the job A. In our paper manuscript, the dominant resource in Eq.(1) is a formalized description since this expression in the Section III, the problem definition section. It means the resource type whose quota or consumption has the largest percentage of the cluster resource when running DLT jobs, compared to other resource types. For example,when running DLT jobs in our testbeds, we can regard CPU and GPU are the dominant resource types in the CPU and GPU clusters respectively.

(2) Assumption of applications

Actually, we leverage the real characteristics of the DLT jobs in our paper manuscript but not the arbitrary assumptions of the applications. Moreover, these characteristics are verified by many prior researches. We can summarize these assumptions or characteristics below:

1) Iterativeness

We ran 2,000+ DLT jobs, including VGG, ResNet, DenseNet, Xception and etc in the CPU/GPU Kuberentes clusters and collect the data of resource consumption and iteration

time(duration) of these jobs. Then we find the iterativeness of the DLT jobs: 1) the resource consumption of the instance in the DLT job has a iterative property, and a almost constant peak of resource consumption appears in each iteration; 2) without the resource adjusting, the iteration time of a DLT job is similar (or the average iteration time is almost constant). These insights are shown in th Figure 2 in the paper manuscript. We can find the similar insights or properties in some references, such as Gandiva (OSDI'2018)[2]. In addition, we have present the iterative process of the running DLT job in the answer of Review 1 (Page 1). The iterative execution of DLT jobs is the inherent reason of the iterativeness of the DLT job. And the resource consumption and the average iteration time should be affected by the inherent features: **batch size, the number of the floating-point calculation operations** and **the number of the trainable parameters.** Meanwhile, the cluster workload also affects the iteration time according to the [2].

Therefore, we propose a GBDT model to predict the resource consumption peak of the DLT jobs. Since the PS and workers have the different execution process as mentioned in the part of this document: Review1.(1).1), we train two GBDT models for PSs and Workers respectively. As for the average iteration time, we use the random forest model to predict it based on the inherent features, weighted cluster utilization score and the resource utilization of the host placing the job.

Furthermore, we need to build the models for the CPU cluster and GPU cluster respectively. As for the performance of the GBDT models, the R^2 value of the CPU, RAM and GPU estimations reaches 86.9%, 96.9% and 97.57%, respectively. As for the RF model, the R2 value of the iteration time estimation is 97.9%. The mean squared error of the iteration time estimation is 21.79 and 0.0556 for the CPU and GPU cluster, respectively. The performance of models for CPU resource seems to be kind of unsatisfied. But this is because that the Kubernetes can support a more fine-grained CPU resource unit (1/1000 CPU cores) but only support the GPU resource unit as 1 GPU card. In addition, iteration time may be up to tens of seconds in the CPU cluster.

2) Assumption of the convergence

Since the inherent of the training process in DLT jobs is to update the parameters with gradient, a large number of the deep learning models will be convergent to a locally optimal solution of parameters [3]. As for the data collected from our submitted DLT jobs we can observe the convergence as well. Then we can leverage this convergence property when predicting the remaining iterations.

3) Assumption of the ratio between the number of PS and Worker in a DLT job.

Since I can not find the theoretical reference of the set of this ratio, when I design the Qore-DL, I have to analysis the collected data of our submitted DLT jobs to find the effective ratio.

Specifically, we give the range of the ratio mainly based on the affect of this ratio on the average execution time and iteration time shown in the Figure 3 of the manuscript. We will collect data from the DLT jobs training more models, such as conformer and transformer models in the production cluster and analysis the open source data such as [4] to quantify this ratio more reasonably.

4) Assumption of the decline rate

We define the decline rate in the Eq.10 in the Section IV.D.1) of the paper manuscript and leverage this metric to predict the convergence of the DLT jobs. In our collected data, the decline rate can reflect the convergence property of the DLT job and we present a threshold range of the decline rate in the Section V.A.3) (the workload generation section in the Setup of the Evaluation).

The prediction of the convergence is also utilized in some prior work, such as Optimus [5], SLAQ [6] and Klein, Aaron, et al's prediction[7]. However, these researches tends to assume that the value of training or validation loss has a inversely proportional or inversely-square proportional relationship with the number of the executed iterations. In Qore-DL, we use the LSTM model to learn the relationship to avoid the arbitrary assumption. As for the decline rate, a similar metric is defined in SLAQ[6], i.e., the reduction of the normalized loss value between iterations.

the connection of a variable or statement are not justified by references or relation to what is happening on the system.

(3) Definition of variables

We list all the variables in the Table I in the Section III.B of the paper manuscript as follow:

TABLE I: Symbols used in Qore-DL

Symbol		Description
Job request	d_i	Deadline
	I_i	Maximum number of iterations
	γ_i	Threshold of loss decline rate
(L hosts) Cluster	U_l	Weighted resource utilization of host H_l defined in Eq. (17)
	U	Weighted resource utilization of the cluster defined in Eq. (1)
	α_j	Weight of resource type $j \in J$, $\sum_{j \in J} \alpha_j = 1$
	β_l	The probability of placing a task on host H_l defined in Eq. (16)
	$\theta_1 < \theta_2 < \theta_3$	Thresholds of U for cluster status
Configurator	N_i^w, N_i^p	Number of workers and PSs
	r_{ij}^p, r_{ij}^w	Peak requirement of resource $j \in J$ of PS and worker
	ρ	The ratio between N_i^p and N_i^w in Eq. (5)
Scheduler	T_i^A	Available time before deadline defined in Eq. (9)
	R_i^p	Normalized resource requirements of PSs defined in Eq. (7)
	R_i^w	Normalized resource requirement of workers defined in Eq. (7)
	R_i	Normalized overall resource requirement defined in Eq. (6)
	p_i	Priority of τ_i when scheduling defined in Eq. (8)
	N^C	The number of the candidate hosts in Algorithm 1
Adjuster	I_i^r	Predicted necessary remaining iterations
	T_i^I	Average iteration time
	T_i^E	Remaining execution time defined in Eq. (11)
	dr	Decline rate defined in Eq. (10)
	O_i	Time-out degree defined in Eq. (12)
	S_i	Resource sensitivity defined in Eq. (13)
	f_{iu}	Fitness of adjustment solution u in GA defined in Eq. (15)

In our revision, we will add the reference of the variables when using it if we have the equation to define it or if this variable definition takes the idea of the prior work. We divide the variables in Qore-DL into five groups.

1) "Job request" variable set: symbols or variables in this set mean the users need to specified for the DLT jobs, including: 1) **deadline**, 2) **upper limit of the number of iterations:** both are as presented in part Review1.(1).2) in this document. 3) **expected decline rate threshold**: we explain the characteristic (assumption) and threshold of loss decline rate in the part: Review3.(1).4) of this document. This threshold of loss decline rate is used to predict the convergence and the necessary remaining iterations of running DLT jobs. If the decline rate of the loss value remains below the **expected decline rate threshold** in enough number of continuous iterations, we can judge that the DLT job is convergent. If the jobs complete the **upper limit of the number of iterations** or become convergent, we think the job is completed. The similar completion judgement of deep-learning training is applied in the Tensorflow, Pytorch and scikit-learn library, which are widely used in the real systems.

As for the user-specified **deadline**, Qore-DL is QoS-aware to guarantee the deadline

satisfaction and allow the users to specify the deadline of the DLT jobs. In production, a DLT job is often time-consuming and execute for hours or even days as shown in the paper manuscript. Client concerns the time cost of the jobs in production as presented in [4]. A series of prior research, such as Optimus[5], SLAQ[6] attempts to reduce the completion time of the DLT jobs, however, these frameworks are not QoS-aware, they tend to give more computing resource to the jobs with high resource efficiency or high convergence speed, but users can not control the jobs to complete before the user-expected deadlines.

2) "Cluster" variable set: Since the DLT jobs need to run in the real clusters. The variables in this set attempt to describe the cluster behavior in the real system. We explain the definition of U and U_l in the part: Review 3.(1).1) of this document. We use the weighted cluster utilization score U to quantify the pressure of the cluster workload or the weighted cluster utilization, and this metric can be computed in the real systems' clusters. We will validate this metric in the production cluster. If in Eq.(1) of paper, we set the weight $\beta_l = \frac{1}{L}$ for each host in the cluster, U is equivalent to the arithmetic mean value of the cluster resource utilization. As for the dominant resource utilization of host H_l, U_l, we think this metric reflect the resource consumption of the concerned resource type in the real cluster. For example, in our testbeds or the production cluster of the top-5 cloud-computing company shown in [4], the resource type: CPU, GPU and Memory are considered.

3) "Configurator" variable set: As we have mentioned in part Reviewer1.(1) of this document, a DLT job running the real clusters have multiple instances. The **configurator** needs to quantify the number of instances, e.g.., the number of PSs and workers for PS framework. As the assumption of iterativeness presented in the part Reviewer3.(2).1) of this document, the instance of the DLT job has a peak resource consumption in each iteration for each resource type, we can use this peak as the resource quota of the instance.

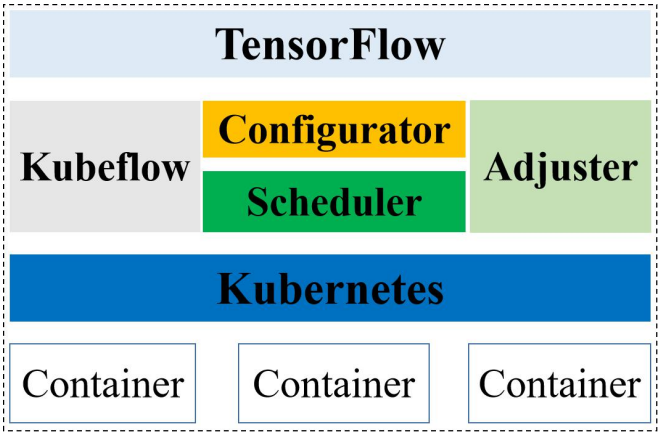
4) "Scheduler" variable set: As we have mentioned in part Reviewer1.(1) of this document, the **scheduler** of the Qore-DL considers the **deadline** and the resource workload of the cluster. This means that we need to compute the priority of the **queuing** jobs. Meanwhile, the normalized resource requirement of PS and worker (i.e., N_i^P and N_i^W) defined in Eq.(7) of the paper manuscript and the available time before deadline (i.e., T_i^A) defined in Eq.(9) of the paper manuscript are computed for the scheduling priority quantification. As for the candidate host N^C in Algorithm 1, when placing a DLT job into a real cluster, traversing each machine in the large cluster is not feasible, Qore-DL first selects some machines as the candidate hosts (machines) to place the scheduling job. This scheme is designed to simplify the placing process.

5) "Adjuster" variable set: According to the running process, the iterativeness and convergence of DLT jobs, which are explained in the part Review3.(2) of this document, we can estimated the average iteration time (expressed as T^l) and the remaining necessary iterations before convergence (expressed as I^r) to estimate the remaining running time T^E of a DLT job. If we know the remaining running time, T^E, we can judge if the DLT jobs face the QoS violation risk, and can make the adjusting decision in the real system. The time-out degree O_i and the resource sensitivity S_i are the metrics, which represent the deadline satisfaction and the resource efficiency of DLT jobs in the adjusting methodology respectively.

The Experiments are simply indecipherable. It appears the authors are using containers running on "testbeds" that are running on "7-host" systems. I have no idea what this means. Is this 7 containers? 7-nodes? 7 CPUs or GPUs? vCPUs?

(4) Resource Configuration of the Testbed Clusters

The testbed are the Kubernetes clusters consists of machines. We mentioned run “containers” on the testbeds, which means that the prototype of the modules of Qore-DL and the realistic training DLT jobs are run as containers in the cluster as the figure follow:



In our evaluation, we write the code of the deep learning model using the Tensorflow library and then we add the codes to the docker images which package the Tnesorflow and CUDA environment. When We submit a job, Qore-DL leverage the tf-operator of the Kubeflow to organize the DLT job as the containers running in the Kubernetes cluster. Each instance (PS or worker) in the DLT job is running as a container, which execute the program packaged in the docker image. The containers of a DLT job are communicated in the PS framework.

The prototype of the Qore-DL are based on the Kubernetes and some operators of Kubeflow and run as the containers in the Kuberentes cluster.

For details of the testbeds, we will modify the invalid description in the revision. In Section V (EXPERIMENTS) of the paper manuscript, “7-host CPU”, “24-host CPU” means the CPU clusters consist of 7 and 24 cloud hosts (machines) respectively. The “5-host GPU” means a GPU clusters consists of 5 cloud hosts (machines). The detail of the resource capacity of the testbed clusters are presented as the follow table:

TABLE II: Resource configuration of the three clusters

CPU cluster consists of 7 hosts						CPU cluster consists of 24 host						GPU cluster consists of 5 hosts		
Memory(GB) / vCPUs	48	96	128	192	256	Memory(GB) / vCPUs	64	96	128	192	256	GPUs / vCPUs	4	1
8						8						8		×2
16			×1			16	×16		×1			16		
24	×1	×3				24		×4		×1		24		
32			×1		×1	32			×1			32	×3	
64						64					×1	64		

The CPU cluster consisting of 7 cloud hosts has a total of 160 vCPUs and 908GB Memory. The CPU cluster consisting of 24 cloud hosts, has 488 vCPUs and 2TB Memory. The GPU cluster

consisting of 5 cloud hosts, has 14 V100 GPU cards, each with 16GB GPU memory. For each cluster, we use NFS to share a 3TB SSD storage space, and bandwidth of the LAN is 10Gbps.

> Later, the CPUs and GPUs are described in terms of weighted coefficients alpha1 = .7 and alpha2 = .3 seemingly because "DLT jobs are always compute intensive". Several things wrong here: what do these numbers mean? how are they being used? In the paper this leads to some concept called "dominant resource utilization" of a host?

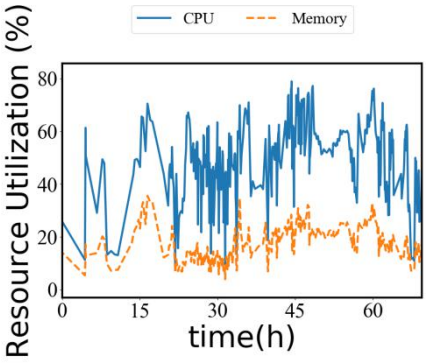
(5) Parameter settings

The parameter settings is based on four aspects: 1) the inherent characteristics and training process of the DLT jobs; 2) the collected resource and training trace of the realistic training DLT jobs in our testbeds; 3) the designed approach of Qore-DL; 4) some open-source dataset of the large-scale production clusters. The parameter settings below are for our evaluation. Since these parameters are general for the real clusters and real DLT jobs, they can be tuned when running on the production cluster to satisfy the user requirement in real production. However, some profile of the production clusters may be needed.

1) alpha1 (α_1) and alpha2 (α_2)

We are sorry that we do not explain the parameter alpha1 and alpha2 clearly in Section V.A.2) (Parameter Setting) of the paper manuscript. Actually, we consider the resource type: CPU, GPU and Memory in our experiments. In order to consider the multiple resource types in a metric, in our evaluation, we compute the dominant resource utilization of host H_l, U_l in the Eq.(17) of the paper manuscript. alpha1 is the wight of normalized utilization of CPU / GPU in the CPU / GPU cluster respectively and alpha2 is the weight of the normalized utilization of Memory in the cluster. alpha1 and alpha2 are used in the Eq.(17) and Eq.(7) in the paper manuscript to compute the normalized utilization of the dominant resource of hosts and the normalized resource requirement of DLT jobs.

As for the value of alpha1 and alpha2, we set it as 0.7 and 0.3 in our evaluations. It can be attributed to three reason: 1) we ran the 2,000+ DLT jobs in the cluster consists of 24 hosts using Kubeflow scheduler and we discover that the average cluster utilization of CPU/GPU and Memory is about 46.62%/54.25% and 18.43% respectively. These values show that CPU/GPU is more dominant and the ratio between CPU/GPU and Memory is about 7:3 in our evaluations according to the DRF algorithm[1]; 2) since we have present the training process in the part Review1.(1).1) in this document, we can find that the DLT jobs are often compute intensive for the gradient computation; 3) CPU/GPU is often more costly in the production. For instance, the figure bellow presents an example of running DLT jobs in our cluster for 70 hours using the Kubeflow scheduler:



2) Parameters in the machine learning models GBDT and random forest.

The number of the trees in the GBDT models and random forest models are tuned when we training the models. In our experiments, more trees for these models do not bring the valuable accuracy improvement and we do not need to complicate the models with more regression trees to waste the computing resources.

3) β_l in Eq.(1) and Eq.(16)

We define the weight of the U_l as the β_l when computing the weighted cluster utilization score U. The value of β_l is in the range of (0,1) and the sum of the β_l is 1. In our scheduling algorithm, Qore-DL tends to leverage the resource in the hosts (machines) having lighter workload to optimize the cluster efficiency in a balance way. Therefore, we use β_l as the probability of placing the DLT-job container on the host H_l. and the computation of β_l in Eq.(16) of the paper manuscript means that, in our evaluation, the confidence level of placing the jobs on the first third of hosts with lightest workload is about 95%. And the weight β_l gives larger weight to these first third of hosts can reflect the available resource for scheduling and adjusting more fitting to our approaches. However, we will conduct more evaluations to validate the influence of the β_l.

4) Three threshold of the U: θ_1,θ_2,θ_3

We set three threshold of the weighted utilization score U. These three threshold $\theta_1 < \theta_2 < \theta_3$ are used to describe the pressure of the cluster workload. In different pressure level (status) of the workload, we use different scheduling and adjusting approach to retain the cluster resource efficiency. In order to get closer to the large-scale clusters in production, we analysis the open-source public Alibaba datasets (4000+ hosts) and compute the weighted utilization score U as the definition in paper. Then we get the result shows in the Figure 5 and Figure 6 in the paper manuscript and set the $\theta_1 = 0.35$, $\theta_2 = 0.55$, $\theta_3 = 0.7$ in our evaluations. The threshold can be tuned when using different scenarios or production clusters according to the behaviors of the real systems.

5) Submission rate of DLT jobs.

We analysis the submission rate of machine learning jobs for the production cluster according to the public open-source Alibaba datasets. And use a more frequent submission in our testbed to guarantee the workload behaviors are more closer to the real production.

6) Decline rate range.

In our collected data from 2,200+ DLT jobs training different models, the decline rate can reflect the convergence property of the DLT job and we present a threshold range of the decline rate. In the future, we will validate our metric in two ways: using the larger open-source datasets of the training loss of benchmark models to calculate the more robust decline rate range and

collect data from more deep learning models, such as conformer and transformer with deploying Qore-DL into larger production clusters to validate and update the decline rate range.

> Again, I have no way to decipher what this means in the context of a real system and there are numerous examples of this lack of clarity throughout the paper. The results are presented as if they are measurements of a job scheduler running workloads on a real system though I think this is basically an unvalidated simulation using AliBaba traces running in containers.

(6) Evaluations to the real clusters

In our evaluations, we built three clusters as the testbeds as we shown above. All the DLT-job workloads are the realistic training DLT jobs. For example, we trained widely used deep learning models, including, VGG, ResNet, DenseNet and Xception. By varying the architecture of the models (e.g., number of layers, number of neural units, size of convolution kernel, activation functions, batch size) of a model, we can also generate thousands of different models to train. In our evaluations, DLT jobs train these models on the cifar10 and imagenet datasets for CPU cluster and GPU cluster respectively. In order to run the DLT jobs as containers, we build the docker image: qoredl/cifar and qoredl/imagenet, which are available at dockerhub. These workload submit to the real cluster and the instance of the real DLT jobs runs as a container. Some data are available at:

https://github.com/qore-dl/qore-dl-code

(7) Deploying the Qore-DL into prodcution cluster

Now we are developing the Qore-DL to the production cluster of an industrial company. We have build a origin version in our private GPU cluster and have schedule tens of DLT jobs for the real production. Some code are available at:

https://github.com/qore-dl/qore-dl-code/tree/main/qoredl_project

In addition, the Qore-DL in the production cluster can support the DLT jobs in non-PS framework and we also build the docker image: qoredl/wenet-k8s-torch to support to run the DLT jobs which are in non-PS framework or coding with Pytorch.

4. Reviewer 4

Thank you for the comments of our paper manuscript. I will explain the main concern in this document.

> The description of the state of the art could be improved. Are PSs still the best architecture for DL training? The focus on that architecture should be better explained and compared to other architectures not using PSs. Also, more recent GPU schedulers like Gavel or Themis are not mentioned.

(1) PS framework is not the state-of-the-art

Tensorflow and Pytroch are phasing in non-PS frameworks, e.g., ring-all-reduce framework.

However, in the non-PS framework, we still need to configure, schedule and adjust worker resources. Qore-DL is a general framework for the DLT jobs and can be applied to the non-PS framework. This is because that the key contributions of the Qore-DL are: 1) the QoS-aware resource optimization scheme for the DLT jobs; 2) achieve the bi-objetctive optimization of DLT-job QoS satisfaction and cluster resource efficiency, which are concerned by the users and cluster providers respectively; 3) realize the joint optimization for the three stages of the DLT jobs; 4) automatically configuring the reasonable resource to the DLT jobs. These optimizations are not depended on the framework of the DLT job but are general for the DLT jobs in the clusters.

We will apply the Qore-DL to the non-PS framework and add the completion with the popular non-PS framework and the recent GPU schedulers in our future work. In our revision, we think we will investigate the state-of-the-art framework and explain the adaption of Qore-DL for the PS framework and non-PS framework more clearly.

Actually, we have submitted non-PS DLT jobs in an industrial company's cluster where we are deploying the Qore-DL. We made the public docker images: qoredl/wenet-k8s-torch to support the DLT jobs in the non-PS framework and We will support the non-PS framework better with the project advancement.

Some choices are not motivated. In particular, heuristic techniques and models are not discussed.

(2) Choice Motivation and the heuristic techniques

I will add more description about the choices in our revisions. I think I need to explain the motivation of this work more and I need to explain why I choose the methodology.

As for the motivation of this work, we analyzed the traces of 1.2 million DLT jobs on a production cluster of a top-5 cloud provider consisting of 1800+ GPU machines. This open-source datasets can be available at [4]. We had three observations: 1) the utilization of user-requested resources is only about 50%; 2) the deadline-urgent jobs are often delayed; and 3) 65% of the DLT-jobs repeat over 5 times.

These insights motivate our work. These current situations cause the DLT jobs running in the cluster with low resource efficiency and the high risk of the QoS violations. We think the optimizations can involve: 1) automatically configuring reasonable resources for DLT jobs can improve resource utilization and client convenience; 2) Both QoS satisfaction and resource efficiency should be optimized; and 3) Repeatable DLT job executions make data-driven approaches feasible.

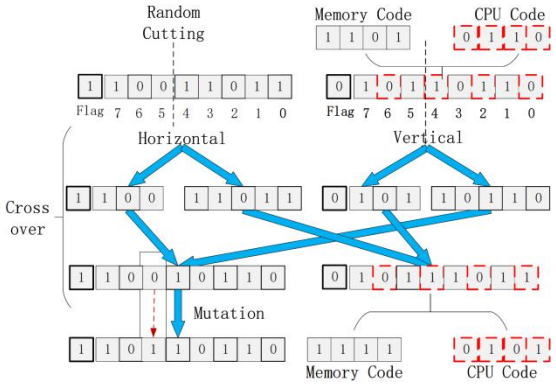
As for the choice of the methodology, we use the data-driven machine learning when configuring the resource. These methodologies is chosen because: 1) too many factors need to be considered when configuring the resources in dynamic clusters, including the characteristics of DLT jobs, the resource contention among DLT jobs and the dynamic resource availability and efficiency of the clusters; 2) we can extract the general and inherent features of DLT jobs from their behaviors; 3) we can collect plenty data of DLT jobs traces, which applies to the data-driven approach.

For the heuristic techniques, they are mainly used in the adjuster. This is because that the bi-objective optimization problem in our work is NP-hard according to [8]. Meanwhile, the

solution of the adjusting can be divided into (i) the selection of target jobs and (ii) the quantification of the adjustment size of the resource. In this situation, heuristic techniques, such as genetic algorithm adapt to searching the local optimal solution well.

In the adjuster, We use the genetic-algorithm heuristic model in the adjusting. We code the adjusting way (horizontal as 1 and vertical as 0) and the adjusting size of the resource in a binary format and use the crossover-mutation methodology to search the solution with better fitness.

In particular, We use a 9-bit binary code to encode candidate solutions. The highest digit of the code represents the two adjustment methods. The lower 8 bits are constructed by the sub-codes of each resource type alternately. For example, The following figure shows the genotype code when considering resource type, CPU and memory:



We maintain a set of candidate solutions for each target job τ_i, and update the set iteratively based on the fitness of solutions. We define the fitness of candidate solution u to τ_i according to the four cases, as shown in Eq.(15) in section IV.D.2) of the paper manuscript.

In addition, all the selected approaches in Qore-DL are light-weight and made trade-off between the performance and the resource consumption of this genetic algorithm execution.

> The contribution made by this work is relevant. Its positioning could be improved.

In our revision, we will summarize more state-of-the-art research and add the description of our work compared to these works.

As for the positioning of our work, we will summarize the innovations of the Qore-DL. DLT-job users and cluster provider expect to complete the DLT job within the required makespan and precision in a high resource efficiency in production [2,4,5,6]. As we analyzed the traces of 1.2 million DLT jobs on a production cluster of a top-5 cloud provider consisting of 1800+ GPU machines. We had three observations: 1) the utilization of user-requested resources is only about 50%; 2) the deadline-urgent jobs are often delayed; and 3) 65% of the DLT-jobs repeat over 5 times. Then we can find some long-standing limitations: 1) automatic and reasonable resource configuration of DLT jobs is kind of absent in practice; 2) QoS-aware optimization of the DLT jobs is often insufficient but is user-concerned; 3) the prior researches often focus on the single stage of the DLT jobs but lack the joint optimization in the lifecycle of DLT jobs; 4) both QoS satisfaction and resource efficiency should be optimized. The Qore-DL address these limitations and consider the bi-objective optimization of resource efficiency and QoS satisfaction to realize the joint optimization of DLT jobs in clusters.

5. Committee Comments

> The paper seems to over rely on equations and assumption.

(1) Assumptions

In summary, we made some assumption when define and solve the problem. These assumption about both the resources involved and the applications.

We make two assumptions on resource to quantify the job resource requirements and the cluster utilization, and these two assumptions are the formalized description of running realistic DLT jobs and the real system.

As for the assumptions of the application, we utilize the real characteristics of DLT jobs, including iterativeness, convergence and decline rate. These are not assumptions, as verified by many works (such as Gandiva[2], Optimus[5] and SLAQ [6]).

We explained these assumptions in the part: Reviewer 3. (1) and Reviewer 3. (2). The assumption about resource is mainly the weight cluster utilization score U and dominant resource utilization of a host(machine) as U_l. The weight cluster utilization score U is the metric to quantify the pressure of the DLT-job workload but not the arithmetic average utilization of the cluster. We give a larger weight to the machines with lighter workload. This is because that we tends to utilize the resource from the machines with lighter workload. Therefore, this metric can reflect the resource availability in Qore-DL better. As for the dominant resource utilization of a host, we refer the idea of the DRF[1] algorithm. The dominant resource means the resource type whose quota or consumption has the largest percentage of the total real cluster resource. when running DLT jobs. For example, CPU/GPU resource can be regarded as the dominant resource of the CPU/GPU cluster when running DLT jobs.

As for the assumption of applications, such as iterativeness, convergence, effective ratio between the number of PSs and workers and decline rate. However, these characteristics are not assumptions, which can be explained in two aspects:

1) theory: we analysis the inherent features and training process of the realistic DLT jobs in part Review 1.(1).1) in this document. Since the execution process of DLT jobs is the repeated iterations of computing gradient, updating parameters and the communication with instances in the same DLT job, the iterativeness, convergence are the inherent feature of the DLT jobs. Some prior research, such as Gandiva [2] and convergence theory [3] can support our definitions of iterativeness and convergence respectively.

2) Insights: we ran 2,000+ realistic training DLT jobs in the real clusters. And then we can discover the insight as shown in the Figure 2,3,4 in the Section IV of the paper manuscripts, which can reflect these characteristics as well.

Therefore, our assumptions of the resource and application are based on the behaviors and the real characteristics of real DLT jobs, or are the quantification of the real clusters and DLT jobs. We will revise our paper and present the behaviors of DLT jobs in the real clusters firstly and summarize the characteristics of the DLT jobs before the system design. Meanwhile, we will try to use more clear text description to present the system design and remove the unnecessary equations.

better to explain how to make parameter choice and apply them in real system/simulations. Some weighting parameter in the equations seems to be chose based on unrealistic assumption

(2) Parameter Settings

The parameter settings is based on four aspects: 1) the inherent characteristics and training process of the DLT jobs; 2) the collected resource and training trace of the realistic training DLT jobs in our testbeds; 3) the designed approach of Qore-DL; 4) some open-source dataset of the large-scale production clusters. The parameter settings below are for our evaluation. Since these parameters are general for the real clusters and real DLT jobs, they can be tuned when running on the production cluster to satisfy the user requirement in real production. However, some profile of the production clusters may be needed.

1) alpha1 (α_1) and alpha2 (α_2) in the Section V.A (Setup) of paper manuscript

We are sorry that we do not explain the parameter alpha1 and alpha2 clearly in Section V.A.2) (Parameter Setting) of the paper manuscript. Actually, we consider the resource type: CPU, GPU and Memory in our experiments. In order to consider the multiple resource types in a metric, in our evaluation, we compute the dominant resource utilization of host H_l, U_l in the Eq.(17) of the paper manuscript. alpha1 is the wight of normalized utilization of CPU / GPU in the CPU / GPU cluster respectively and alpha2 is the weight of the normalized utilization of Memory in the cluster. alpha1 and alpha2 are used in the Eq.(17) and Eq.(7) in the paper manuscript to compute the normalized utilization of the dominant resource of hosts and the normalized resource requirement of DLT jobs.

As for the value of alpha1 and alpha2, we set it as 0.7 and 0.3 in our evaluations. It can be attributed to three reason: 1) we ran the 2,000+ DLT jobs in the cluster consists of 24 hosts using Kubeflow scheduler and we discover that the average cluster utilization of CPU/GPU and Memory is about 46.62%/54.25% and 18.43% respectively. These values show that CPU/GPU is more dominant and the ratio between CPU/GPU and Memory is about 7:3 in our evaluations according to the DRF algorithm[1]; 2) since we have present the training process in the part Review1.(1).1) in this document, we can find that the DLT jobs are often compute intensive for the gradient computation; 3) CPU/GPU is often more costly in the production.

2) Parameters in the machine learning models GBDT and random forest.

The number of the trees in the GBDT models and random forest models are tuned when we training the models. In our experiments, more trees for these models do not bring the valuable accuracy improvement and we do not need to complicate the models with more regression trees to waste the computing resources.

3) β_l in Eq.(1) and Eq.(16)

We define the weight of the U_l as the β_l when computing the weighted cluster utilization score U. The value of β_l is in the range of (0,1) and the sum of the β_l is 1. In our scheduling algorithm, Qore-DL tends to leverage the resource in the hosts (machines) having lighter workload to optimize the cluster efficiency in a balance way. Therefore, we use β_l as the probability of placing the DLT-job container on the host H_l. and the computation of β_l in Eq.(16) of the paper manuscript means that, in our evaluation, the confidence level of placing the jobs on the first third

of hosts with lightest workload is about 95%. And the weight β_l gives larger weight to these first third of hosts can reflect the available resource for scheduling and adjusting more fitting to our approaches. However, we will conduct more evaluations to validate the influence of the β_l.

4) Three threshold of the U: θ_1,θ_2,θ_3

We set three threshold of the weighted utilization score U. These three threshold $\theta_1 < \theta_2 < \theta_3$ are used to describe the pressure of the cluster workload. In different pressure level (status) of the workload, we use different scheduling and adjusting approach to retain the cluster resource efficiency. In order to get closer to the large-scale clusters in production, we analysis the open-source public Alibaba datasets (4000+ hosts) and compute the weighted utilization score U as the definition in paper. Then we get the result shows in the Figure 5 and Figure 6 in the paper manuscript and set the $\theta_1 = 0.35, \theta_2 = 0.55, \theta_3 = 0.7$ in our evaluations. The threshold can be tuned when using different scenarios or production clusters according to the behaviors of the real systems.

5) Submission rate of DLT jobs.

We analysis the submission rate of machine learning jobs for the production cluster according to the public open-source Alibaba datasets. And use a more frequent submission in our testbed to guarantee the workload behaviors are more closer to the real production.

6) Decline rate range.

In our collected data from 2,200+ DLT jobs training different models, the decline rate can reflect the convergence property of the DLT job and we present a threshold range of the decline rate. In the future, we will validate our metric in two ways: using the larger open-source datasets of the training loss of benchmark models to calculate the more robust decline rate range and collect data from more deep learning models, such as conformer and transformer with deploying Qore-DL into larger production clusters to validate and update the decline rate range.

> Need more details on the testbed system and experiments setup. Was the evaluation done on a real system for realistic training studies or on some simulated system using traces running in containers?

(3) Details of Experiments Setup

For details of the testbeds, we will modify the invalid description in the revision. In Section V (EXPERIMENTS) of the paper manuscript, “7-host CPU”, “24-host CPU” means the CPU clusters consist of 7 and 24 cloud hosts (machines) respectively. The “5-host GPU” means a GPU clusters consists of 5 cloud hosts (machines). The detail of the resource capacity of the testbed clusters are presented as the follow table.

TABLE II: Resource configuration of the three clusters

CPU cluster consists of 7 hosts						CPU cluster consists of 24 host						GPU cluster consists of 5 hosts		
Memory(GB) / vCPUs	48	96	128	192	256	Memory(GB) / vCPUs	64	96	128	192	256	GPUs / vCPUs	4	1
8						8						8		×2
16			×1			16	×16		×1			16		
24	×1	×3				24		×4		×1		24		
32			×1		×1	32			×1			32	×3	
64						64					×1	64		

The CPU cluster consisting of 7 cloud hosts has a total of 160 vCPUs and 908GB Memory. The CPU cluster consisting of 24 cloud hosts, has 488 vCPUs and 2TB Memory. The GPU cluster consisting of 5 cloud hosts, has 14 V100 GPU cards, each with 16GB GPU memory. For each cluster, we use NFS to share a 3TB SSD storage space, and bandwidth of the LAN is 10Gbps.

(4) real system for realistic training studies

In our evaluations, we built three clusters as the testbeds as we shown above. All the DLT-job workloads are the realistic training DLT jobs. For example, we trained widely used deep learning models, including, VGG, ResNet, DenseNet and Xception. By varying the architecture of the models (e.g., number of layers, number of neural units, size of convolution kernel, activation functions, batch size) of a model, we can also generate thousands of different models to train. In our evaluations, DLT jobs train these models on the cifar10 and imagenet datasets for CPU cluster and GPU cluster respectively. In order to run the DLT jobs as containers, we build the docker image: qoredl/cifar and qoredl/imagenet, which are available at dockerhub. These workload submit to the real cluster and the instance of the real DLT jobs runs as a container. Some data are available at:

https://github.com/qore-dl/qore-dl-code

(5) Deploying the Qore-DL into prodcution cluster

Now we are developing the Qore-DL to the production cluster of an industrial company. We have build a origin version in our private GPU cluster and have schedule tens of DLT jobs for the real production. Some code are available at:

https://github.com/qore-dl/qore-dl-code/tree/main/qoredl_project

In addition, the Qore-DL in the production cluster can support the DLT jobs in non-PS framework and we also build the docker image: qoredl/wenet-k8s-torch to support to run the DLT jobs which are in non-PS framework or coding with Pytorch.

Reference

[1] Ghodsi, Ali, et al. “Dominant Resource Fairness: Fair Allocation of Multiple Resource Types,” in Nsdi. Vol.11, 2011, pp. 24-24.

[2] Xiao, Wencong, et al. ”Gandiva: Introspective cluster scheduling for deep learning,” in Conf. OSDI, 2018, pp. 595-610.

[3] ALLEN-ZHU, Zeyuan; LI, Yuanzhi; SONG, Zhao. A convergence theory for deep learning via over-parameterization. In: International Conference on Machine Learning. PMLR, 2019. p. 242-252.

[4] Weng, Qizhen, et al. “MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters” in Conf. NSDI, 2022.

[5] Peng, Yanghua, et al. ”Optimus: an efficient dynamic resource scheduler for deep learning clusters,” in Conf. EuroSys, 2018, pp. 1-14.

[6] Zhang, Haoyu et al. ”Slaq: quality-driven scheduling for distributed machine learning,” in Conf. SoCC, 2017, pp.390-404.

[7] Klein, Aaron, et al. "Learning curve prediction with Bayesian neural networks." (2016).

[8] Ullman, Jeffrey D. ”NP-complete scheduling problems,” Journal of Computer and System sciences, vol.10, 1975, pp. 384-393.