ETL: Elastic Training Layer for Deep Learning

Lei Xie
xie-l18@mails.tsinghua.edu.cn
Tsinghua University

Jidong Zhai
zhaijidong@tsinghua.edu.cn
Tsinghua University

1 INTRODUCTION

Deep learning is playing an important role in addressing challenging tasks [1, 10]. To shorten training time, people usually utilize distributed training and special hardware such as GPU. Therefore, enterprise or cloud providers typically setup a GPU cluster running multiple deep learning training tasks concurrently [3]. However, achieving high resource utilization and training efficiency is challenging. There are at least two major problems in existing clusters, which each provides the potential to further improve the efficiency.

First, schedulers in most existing deep learning clusters use *static task configuration* [3, 9], resources are allocated at the time of task submission and remain fixed during the task execution. The configuration may be optimal at the submission time, but will be sub-optimal during the training procedure. On one hand, tasks using static configuration cannot leverage (e.g. more resources will be available during night or weekend), resulting in resource under-utilization. On the other hand, in a fixed-size cluster, existing tasks may use too many resources so that they prevents tasks with higher priority to execute, resulting in resource abuse and poor quality of service (QoS).

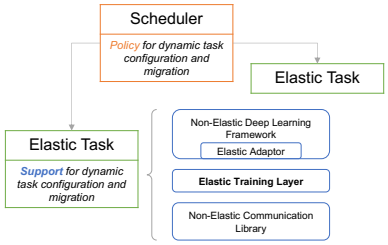
Second, deep learning training tasks are sensitive to GPU locality and multi-task interference [11]. Due to the mismatch between task configuration and hardware configuration (e.g. a machine with 8 GPUs runs a task which requires 4 GPUs) and task queuing time constraint (i.e. schedulers cannot allow a task wait for optimal resources infinitely), there leaves many *resource fragments*. A task that requests a lot of GPU resource may span across a large number of machines, but only uses a small number of GPUs on each machine. This not only harms training efficiency of the task itself due to the poor GPU locality, but also influences other tasks already on the machines due to multi-task interference, resulting in the overall reduction of cluster efficiency.

To achieve high cluster efficiency, running tasks should be dynamically configured, i.e. resources offered to a task can be adjusted at runtime. Besides, task migration should be used as an approach of defragmentation to reduce the influence of multi-task interference for better GPU locality.

To support *dynamic task configuration* and *migration*, we propose ETL, an elastic training layer for deep learning powered by only two essential and efficient mechanisms, *resource addition* and *removal*. With ETL, schedulers can dynamically reconfigure resources of running tasks for overall improvement of efficiency and QoS. Besides, schedulers can also explore the trade-off between queuing time and training efficiency (i.e. ignore GPU locality and multi-task interference for minimum queuing time or wait for the most proper GPU configuration to maximize training efficiency) with migration from ETL for defragmentation.

We detail and evaluate ETL in this paper, and leave dynamic task configuration and migration policy for future work.

Figure 1: Elastic training platform overview.



2 ETL: ELASTIC TRAINING LAYER

In an elastic training platform (shown in Figure 1), the scheduler carries out policy for *dynamic task configuration* and *migration*, while ETL provides elastic training support for training tasks.

2.1 Overview

ETL targets for synchronous distributed training in a data-parallel manner based on Allreduce primitive, which has been widely accepted as the typical way of distributed deep learning in today's GPU clusters [6, 8].

ETL is a lightweight layer residing in between existing static deep learning framework and communication library in order to utilize the optimal performance of both. We have already adapted Caffe [4] and Pytorch [7] and use NCCL [6] as the base communication library.

ETL provides two essential mechanisms, *resource addition* and *removal*. With the two mechanisms, schedulers can add or remove resource as the form of worker at runtime, thus re-scale tasks. Migration can be achieved as adding workers on destination devices and remove workers from source devices at the same time.

2.2 Challenges and Approaches

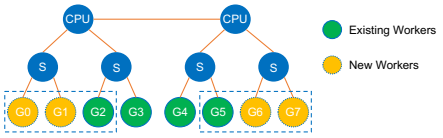
There are three challenges in elastic deep learning training, and we propose several effective approaches for each to address.

How to manage the heterogeneity of deep learning training tasks? Typically, deep learning training tasks not only use massive parallel hardware like GPU for fast computation but also include enormous IO operations and intensive communication. An elastic training system must be aware of and manage such heterogeneity.

How to make trade-off on elasticity and efficiency? To be more elastic means more time devoted to monitoring and managing resource adjustments, thus reduces efficiency. An elastic training system must provide an mechanism to make trade-off on elasticity and efficiency.

How to replicate training state efficiently? Once the worker set changes after resource adjustment, we need to replicate the training state on new workers. The speed of replication determines how often we can adjust resources.

Figure 2: Four new workers are going to join. We perform two broadcast (denoted by dashed box) in parallel to eliminate extra CPU-GPU copy when transferring across CPUs. E.g. G2 broadcasts to G0 and G1. S represents PCIe switch.



To manage the heterogeneity, we investigate and leverage two key characteristics of deep learning training tasks. First, deep learning training is an iterative optimization procedure, which offers the perfect management time point between successive iterations. Second, a deep learning training task can be easily determined by the model parameters and IO states, and all of them are device-independent.

To make trade-off on elasticity and efficiency, we propose a lightweight and configurable *report* primitive: all workers report its willing (to participate in the following training or to exit) every few iterations, and then resources are adjusted respectively. The interval between two successive *report* is configurable to balance elasticity and efficiency.

To replicate training state efficiently, we propose an asynchronous, parallel and IO-free replication mechanism. We execute new workers asynchronously while existing workers are still training models so that the startup overhead of new workers could be hidden. We utilize broadcast primitive directly from device to device, bypass the CPU-GPU copy and IO operations. Besides, we start multiple replication procedure in parallel according to the GPU topology for higher efficiency (See Figure 2 for an example).

3 EVALUATION

We first showcase an elastic training procedure to reveal the potential advantages of elastic training. Then we focus on the inspection of 1) overhead of elasticity from the lightweight and configurable *report* primitive and 2) efficiency of the asynchronous, parallel and IO-free state replication mechanism.

All the evaluations are performed on machines with 8 NVIDIA 1080Ti GPUs each and based on ResNet50 [2] training on Cifar10 [5] by Pytorch.

3.1 Elastic Training of ResNet50 on Cifar10

To demonstrate the advantages of elastic training, we compare the training procedure of 1) static training on 4 GPUs, 2) static training on 8 GPUs and 3) elastic training on 4 GPUs and then 8 GPUs in Figure 3. We train the model for 20 epochs and add another 4 GPUs at 6th epoch when doing elastic training.

It is shown from the loss curve that training performs similar with that on 4 GPUs before 6 epochs and quickly approaches to that on 8 GPUs after resource adjustment. The throughput result also shows that more GPU resources yield higher training throughput and thus leads to higher training efficiency. This case demonstrates the potential speedup when more resources are available.

Figure 3: Elastic training procedure of ResNet50 on Cifar10 from 4 GPUs to 8 GPUs.

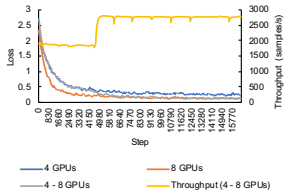


Figure 4: *report* overhead with different numbers of GPU used.

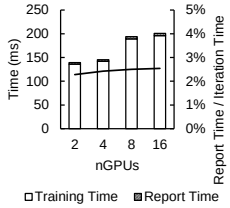


Table 1: Breakdown analysis of checkpoint-based replication (left) and ours (right). We add 4 more GPUs (New 4) into the original ones (Original 4).

Phase		Time (s)
Original 4	New 4	
Save checkpoint		0.245
Shutdown		5.465
Cold restart		64.883
Load checkpoint		2.722
Train		
Total		73.315

Phase		Time (s)
Original 4	New 4	
Train	Cold start	62.374 (hidden)
Report & replication		1.065
Train		
Total		1.065

3.2 Overhead of Elasticity

To make trade-off on elasticity and efficiency, we propose the lightweight and configurable *report* primitive. Figure 4 shows the overhead of *report* primitive with different numbers of GPU used. An iteration consists of training and *report*, whose time is denoted by *training time* and *report time* respectively.

The results shows that the overhead of *report* primitive is less than 2.5% of training time, and is quite stable across 2-16 GPUs. Note that we *report* every iteration for maximum elasticity, while in practice we typically *report* every tens or hundreds of iterations, thus the overhead will be tens or hundreds of times lower.

3.3 Efficiency of Task Replication

We compare with the traditional checkpoint-based replication mechanism [11] to show the efficiency of ours. Table 1 breakdowns the time of both for resource adjustments from 4 GPUs to 8 GPUs.

For checkpoint-based replication, the whole training procedure should all shutdown first and cold restart with the checkpoint. This approach is synchronous, serial and involves IO operations. Except the time of saving and loading checkpoint, the initialization of deep learning framework and model consume a large portion of time while our asynchronous replication mechanism can hide this with training on the original resources, thus results in extreme reduction in replication time. On the other hand, due to the use of parallel and IO-free state replication, we reduce the pressure on storage and consume less time (comparing *Report & replication* time of 1.065s with *Cold restart* time of 2.722s). Note that ResNet50 is a small model with parameters of only 155 MB, replication will benefit more from our approach on bigger models.

REFERENCES

[1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*. 173–182.

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

[3] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. *arXiv preprint arXiv:1901.05758* (2019).

[4] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).

[5] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[6] NVIDIA. 2019. NCCL. https://developer.nvidia.com/nccl

[7] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[8] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[9] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 18.

[10] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[11] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. 2018. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 595–610.