Xonar: Profiling-based Job Orderer for Distributed Deep Learning

Changyong Shin, Gyeongsik Yang, Yeonho Yoo, Jeunghwan Lee, Chuck Yoo
Department of Computer Science and Engineering, Korea University
{cyshin, ksyang, yhyoo, jhlee21, chuckyoo}@os.korea.ac.kr

***Abstract*—Deep learning models have a wide spectrum of GPU execution time and memory size. When running distributed training jobs, however, their GPU execution time and memory size have not been taken into account, which leads to the high variance of job completion time (JCT). Moreover, the jobs often run into the GPU out-of-memory (OoM) problem so that the unlucky job has to restart all over. To address the problems, we propose Xonar to profile the deep learning jobs and order them in the queue. The experiments show that Xonar with TensorFlow v1.6 reduces the tail JCT by 44% with the OoM problem eliminated.**

***Index Terms*—GPU cloud, distributed deep learning, parallel training, GPU utilization, job completion time**

I. Introduction

Deep learning (DL) models continuously grow bigger to enhance the prediction accuracy and cover more diverse applications from image classification to advertisement [1]–[3]. To train the big DL models, distributed DL training (DDLT) is widely used. As DDLT requires a number of GPUs and networking between GPU nodes, the GPU cloud is often utilized to facilitate better resource utilization and consolidation [1], [3]–[5]. We denote each individual training on a GPU as a "job." In a GPU cloud, the representative cloud orchestration platform, such as Kubernetes, places jobs in the GPU run queues. Once a job is placed, GPU executes the jobs serially (serial training) [4]. Although serial training is a de facto standard in existing GPU clouds, GPUs are exclusively used and so underutilized. To enhance GPU utilization, quite a few research efforts were made to run the jobs in parallel on a GPU (parallel training) [2]–[4], [6]. They just receive jobs in the run queue and execute them; thus, they have two severe limitations as follows.

The first limitation is that their JCT is highly variable. Suppose that a GPU runs two jobs together. Between the two jobs, one job can occupy the GPU when the other job is idle. So, the ideal situation is that the active duration of one job's utilization overlaps the idle duration of another job. However, it is well-known that the mini-batch time extremely depends on deep learning models [3]. Accordingly, the active and idle duration also depends on the models. To get some sense, we measure the active and idle durations of the image classification models (e.g., AlexNet, VGGNet, ResNet, Inception, and DenseNet) of TensorFlow (TF) benchmark. We observe that the active and idle durations range from 5 ms to 5 s, which are of 1000× difference. Once the jobs are placed on a run queue, their JCT becomes dependent on the enqueued order. Thus, it is quite evident that their JCT becomes highly variable (up to 1.5× difference in our experiments, §II-B2).

The second limitation is that jobs are placed in the run queue without considering the GPU memory sizes. When the jobs run parallel training, they often run into the GPU out-of-memory (OoM) problem. The OoM problem happens when the required memory of the jobs is greater than the GPU memory's capacity. When the OoM problem occurs, all training jobs halt and should be restarted from the beginning. In our experiments, we train two jobs in parallel using the image classification models used above by TF (v1.6) benchmark with V100 GPU of 32GB GPU memory. We observe that 15.2% of the parallel training results in the OoM problem (§II-B3).

To address these limitations, this paper proposes Xonar, a profiling-based job orderer for DDLT. Xonar first profiles the jobs (offline) and finds the GPU execution time and memory size. Specifically, we instrument TF v1.6 library to measure the active and idle durations of GPU execution. Also, we obtain the GPU memory size (consumption) during the active duration. Xonar then uses the information to order jobs in the run queue. Experiment results show that Xonar reduces the 99% tail JCT by 44.1% and 32% than serial training and parallel training, respectively. In addition, Xonar entirely removes the occurrence of the OoM problem.

II. Background and Motivation

A. Related Work

Kubernetes [7] is the representative container orchestration platform. Kubernetes places jobs on the GPU servers. Each GPU then serially trains the placed jobs. Specifically, the jobs are trained on a first-come first-served (FCFS) basis, which means that job training follows the enqueued order. To enhance the poor GPU utilization of serial training, several studies propose the following approaches. HiveMind [2] merges eight individual jobs into one bigger job. Also, KubeShare [4] runs all enqueued jobs at once in parallel. Ebird [6] runs multiple inference jobs on a GPU and sets an upper limit for the number of co-running jobs. Note that HiveMind, KubeShare, and Ebird load the jobs on the FCFS basis to run in parallel without any

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation funded by the Korea government (Ministry of Science and ICT) (2015-0-00280, (SW Starlab) Next generation cloud infra-software toward the guarantee of performance and security SLA) and by Basic Science Research Program through National Research Foundation of Korea funded by the Ministry of Education (NRF-2021R1A6A1A13044830). Co-corresponding authors: Chuck Yoo and Gyeongsik Yang.

consideration of the order of jobs in the run queue. Thus, they end up showing highly varying JCT. Furthermore, these studies do not handle the OoM problem during training. When the OoM problem occurs, all the training jobs halt, and the training of the jobs should be restarted from scratch.

AntMan [3] also runs jobs in parallel. While running jobs, AntMan obtains the mini-batch time from the DL libraries (e.g., TF) as the DL libraries commonly report them. AntMan then increases the number of jobs that run in parallel as long as the mini-batch time does not go over its threshold. When loading the jobs, AntMan follows FCFS, and so the order of training is the order of jobs enqueued. For the OoM problem, AntMan introduces a mechanism that uses the host memory together with the GPU memory. However, the total memory capacity is limited, so the OoM problem still remains. Yet, AntMan does not address the OoM problem.

B. Motivating Experiments

1) Experiment setup: : Unless stated otherwise, we use the following setup throughout this paper. Rather than compositing custom DL models, we use the de facto image classification DL models (e.g., AlexNet, VGGNet, ResNet, Inception, and DenseNet) provided by the official TF (v1.6) benchmark [8]. From the benchmark, we make 600 cases of DDLTs through combinating DL models, hyperparameters (e.g., optimizers, fp_16, and batch size of 32 to 512), and datasets (e.g., CIFAR-10 and ImageNet). The experiments are conducted with a GPU server equipped with Intel Xeon Skylake processor (16 cores), 128 GB memory, and two V100 GPUs that have 32 GB GPU memory each. All the DDLTs are with data parallel and parameter server strategy. We configure two parameter servers and two GPU workers, and each worker trains with a separate GPU. To evaluate parallel training, each GPU runs two jobs at once.[1] Each worker repeats the training for 100 iterations.

2) Highly variable JCT problem: To see the varying JCT of parallel training, we conduct the following experiment. From the 600 cases, 10 jobs are randomly selected and enqueued. We permute the enqueued order of the jobs. Then, we check the JCT variance to see the effect of the enqueued job order. If the OoM problem occurs, we stop and serially train the jobs that caused the OoM problem. The other jobs in the queue are then trained in parallel.

We repeat this experiment three times (experiments A to C in the x-axis of Fig. 1). In each experiment, 10 jobs are randomly and freshly selected, and they are trained 25 times by changing the enqueued order of the selected jobs. Fig. 1 shows the JCT distribution of the three experiments, which is normalized to the minimum JCT in each experiment. In the results, JCT shows high variability from 1.2× (experiment A) to 1.5× increase (experiment B). The results show that the training (enqueued) order makes a big difference in JCT.

3) OoM problem: We test the OoM problem. Similar to the JCT problem, we randomly choose 10 jobs from 600 cases.

[1]We empirically determine the number of jobs to run together as two considering the 600 DDLTs' peak memory consumption and GPU capacity.

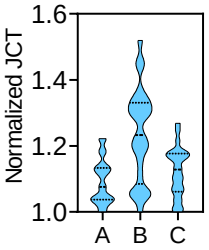


Fig. 1: Highly variable JCT.

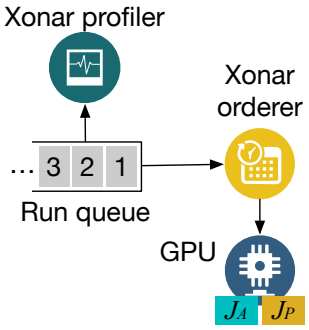


Fig. 2: Xonar components.

By training two jobs in parallel, we check how often the OoM problem happens. Out of 396 parallel training, we observe that the OoM problem happens 60 times, which is about 15.2%. This is a non-trivial ratio as the previous studies have no solution to the OoM problem.

III. Design

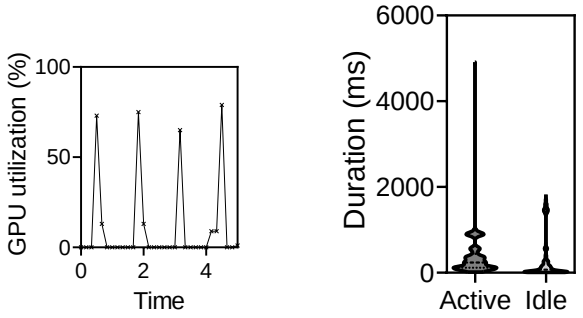
Fig. 2 shows the components of Xonar. Xonar consists of profiler and orderer, detailed as below.

A. Xonar Profiler

Xonar profiles the active and idle durations of GPU utilization. Existing GPU profiling tools, such as NVML and pyNVML, can measure the GPU utilization; to our knowledge, however, they cannot distinctly measure the active and idle durations. So, we design "Xonar profiler" based on the in-depth analysis of the TF v1.6 internals. We have analyzed the internals of TF v1.6 in details as follows: 1) high-level API codes that translate operations like the convolution layer into various computations (called op, such as matrix multiplication), 2) the ops that become a computational graph, 3) the internal implementation of the detailed ops for devices (GPU), and 4) the training pipeline that executes the ops of the graph one-by-one according to the device. From this detailed analysis, Xonar profiler instruments the source code of the TF library and clocks the start and end of each op that TF executes. By distinguishing the devices (i.e., GPU) where the ops are executed, Xonar profiler accurately measures the active and idle durations per iteration.

To address the OoM problem, we especially focus on the peak memory consumption. The peak memory consumption is obtained as follows. First, for each iteration, Xonar profiler regularly measures the amount of GPU memory that a job holds. Then, the maximum memory consumption is recorded in an iteration. Xonar profiler then selects the highest value from the maximum consumptions of multiple iterations as the peak memory consumption. In measuring the GPU memory consumption of a job, Xonar profiler leverages NVML [9]. The interval of measurement is 1/6 s, which is the minimum interval for measuring GPU memory by NVML.

With Xonar profiler, we profile 600 cases to investigate the GPU utilization. Fig. 3a is the GPU utilization snapshot of the VGG16 model trained by ImageNet dataset for 5 s.



(a) VGG16 example. (b) Duration differences.

Fig. 3: GPU profiling

The figure includes four iterations, and the GPU utilization repeatedly shows active and idle durations. Also, the active and idle durations look quite constant across iterations. Note that the idle durations occur due to the networking in DDLT for updating gradients and parameters. By profiling, we observe that 600 cases follow trends similar to what Fig. 3a shows.

Second, we investigate the idle and active durations. We run 600 cases and get the distribution of the active and idle durations per iteration.[2] Fig. 3b shows the results where the active duration ranges from 18 ms to 4.9 s, and the idle duration from 5 ms to 1.8 s. Assume two jobs running together on a GPU. If the idle duration of a co-running job is short, the other job has less opportunity to utilize the GPU. On the opposite, where the idle duration is long, the other job will have more opportunities to utilize the GPU. Xonar provides the active and idle duration time of all the 600 cases to job orderer explained in §III-B.

Xonar profiler takes about 63.4 s on average for the profiling of each case. In this study, we use the pre-profiled data (offline) to show the feasibility of Xonar. However, considering that the jobs of existing public GPU clouds usually wait for their turn for training for up to hundreds of minutes [1], we believe the profiling time can be overlapped with the waiting time. Further, we plan to develop prediction methods for profiling metrics to remove the job profiling step.

B. Xonar Orderer

JCT highly varies by the training (enqueued) order of jobs as seen in Fig. 1 (§II-B2). Also, we observe that the jobs show very different active and idle durations. Xonar orderer determines the proper order of training jobs by considering the active and idle durations as follows. First, Xonar orderer picks a job from the run queue. We denote this job as J_A. To find another job (J_P) to run together with J_A, Xonar orderer scores the degree of fitness in terms of two factors. The first factor is the GPU memory capacity. Xonar orderer first filters out the jobs in the run queue that require memory larger than the current available GPU memory that is the GPU memory capacity subtracted by the peak memory consumption of J_A. In filtering, we use the peak GPU memory consumption to avoid the GPU OoM problem. For the filtered jobs ($Q_{filtered}$),

[2]As the durations are constant across iterations, we empirically average the durations from 10 iterations.

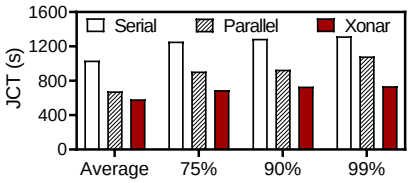


Fig. 4: JCT evaluation

the second factor is the fitness between the idle and active durations between jobs. For each job of $Q_{filtered}$ (J_n), Xonar orderer calculates the active-idle ratio (r_n) by dividing the active duration of J_n by its idle duration. Let us assume that the r_A is 6/4. Then, the proper J_P would be the job whose r_n is the inverse of the r_A (i.e., 4/6) because J_P's active duration can overlap with the idle duration of J_A. Thus, Xonar orderer selects J_P using the following Equation 1.

$$J_P = \min_{J_n \in Q_{filtered}} |r_A \times r_n - 1| \quad (1)$$

IV. Evaluation

We implement Xonar profiler and job orderer using Python with TF v1.6. We compare Xonar with serial training and parallel training in the experiment setting of §II-B1.

Varying JCT. Our evaluation runs10 jobs randomly selected from 600 cases. We compare serial training, parallel training, and Xonar. The experiment is repeated 50 times. Fig. 4 shows the JCT with the average, 75%, 90%, and 99% tail JCTs (x-axis). For the average JCT, Xonar reduces JCT by up to 43.6% and 13.7%, compared with serial training and parallel training, respectively. The improvements are greater for the tail JCT. Xonar bounds its 99% tail JCT within 736 s, while serial and parallel trainings reach up to 1317 s and 1083 s, respectively. This means that the 99% tail JCT of Xonar is 44.1% and 32% lower than those of serial and parallel trainings. These results show that Xonar reduces the JCT variability significantly.

OoM problem. In §II-B3, we have noticed that 15.2% of parallel training encountered the OoM problem. We evaluate Xonar in the same experiments and find that Xonar entirely avoids the occurrence of the OoM problem.

V. Conclusion and Future Work

This study proposes Xonar, a profiler-based job orderer for DDLT. We show that Xonar enhances the tail JCT by up to 44% while entirely avoiding the OoM problem. As future work, we extend Xonar to cover parallel training for more than three jobs. Also, we plan to apply Xonar to other DL libraries (e.g., PyTorch), different DT strategies (e.g., model parallel and all-reduce), various DL workloads (e.g., language models), and various GPUs. Furthermore, we plan to cover job placement over multiple GPU servers with Xonar as well.

References

[1] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 595–610.

[2] D. Narayanan, K. Santhanam, A. Phanishayee, and M. Zaharia, "Accelerating deep learning workloads through efficient multi-model execution," in *NeurIPS Workshop on Systems for Machine Learning*, vol. 20, 2018.
[3] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "AntMan: Dynamic scaling on GPU clusters for deep learning," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 533–548.
[4] T.-A. Yeh, H.-H. Chen, and J. Chou, "KubeShare: A framework to manage GPUs as first-class and shared resources in container cloud," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, 2020, pp. 173–184.
[5] M. Kang, G. Yang, Y. Yoo, and C. Yoo, "TensorExpress: In-network communication scheduling for distributed deep learning," in *2020 IEEE 13th international conference on cloud computing*, 2020, pp. 25–27.
[6] W. Cui, M. Wei, Q. Chen, X. Tang, J. Leng, L. Li, and M. Guo, "Ebird: Elastic batch for improving responsiveness and throughput of deep learning services," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 2019, pp. 497–505.
[7] Kubernetes. Accessed: 2022-03-12. [Online]. Available: https://kubernetes.io/
[8] benchmarks/scripts/tf_cnn_benchmarks. Accessed: 2021-12-18. [Online]. Available: https://github.com/tensorflow/benchmarks
[9] NVML API reference guide. Accessed: 2021-10-24. [Online]. Available: https://docs.nvidia.com/deploy/nvml-api/structnvmlUtilizationt.html