An Adaptive Batch-Orchestration Algorithm for the Heterogeneous GPU Cluster Environment in Distributed Deep Learning System

Eunju Yang, Seong-Hwan Kim, Tae-Woo Kim, Minsu Jeon, Sangdon Park and Chan-Hyun Youn
School of Electrical Engineering
Korea Advanced institute of Science and Technology
Daejeon, Korea
{yejyang, s.h_kim, taewoo_kim, msjeon , sangdon.park, chyoun}@kaist.ac.kr

***Abstract*— Training deep learning model is time consuming, so various researches have been conducted on accelerating the training speed through distributed processing. Data parallelism is one of the widely-used distributed training schemes, and various algorithms for the data parallelism have been studied. However, since most of studies assumed homogeneous computing environment, there is a problem that they do not consider a heterogeneous performance graphics processing unit (GPU) cluster environment. The heterogeneous performance environment leads to differences in computation time between GPU workers in the synchronous data parallelism. Due to the difference of the computation time of one iteration, the straggler problem that fast workers wait for the slowest worker makes training speed slow. Therefore, in this paper, we propose a batch-orchestration algorithm (BOA), reducing the training time by improving hardware efficiency in the heterogeneous performance GPU cluster. The proposed algorithm coordinates local mini-batch sizes for all workers to reduce the training iteration time. We confirmed that the proposed algorithm improves the performance by 23% over the synchronous SGD with one back-up worker when training ResNet-194 using 8 GPUs of three different types.**

Keywords- Batch size; distributed training; heterogeneous GPU cluster.

I. INTRODUCTION

Deep learning (DL) has improved the state-of-the-art technologies in various fields such as image processing, natural language processing, etc. [1]. Comparing with other machine learning algorithms, training a DL model takes longer time because it iteratively updates model parameters using gradients calculated by the huge amount of training data. To reduce the time, many distributed training schemes have been studied [2] such as data parallelism and model parallelism with graphics processing units (GPUs) as workers. In recent years, data parallelism, which is relatively easy to implement and has a high degree of scalability, is mainly used to accelerate the training [3]. Data parallelism (DP) is a way in which several workers calculate the gradients of same model replicas using different subsets of the training data, called mini-batch [4]. The gradients computed by the workers, usually GPUs, are sent to a parameter server to update the parameters of the DL model according to mini-batch SGD algorithm.

Synchronous SGD is a DP method that the parameter server synchronously waits for all gradients from the workers, updates the model parameters, and sends them back to workers. The synchronous SGD shows the high convergence rate when a large number of workers are used in training [8]. However, its hardware efficiency (HE) [5], the speed of training one iteration, is low because of a straggler problem. The straggler problem refers that the slowest worker, called straggler, delays the iteration time of the distributed training [7]. Fig. 1 shows the straggler problem in the synchronous training. The synchronous SGD waits for all workers and then proceeds with the next iteration, so the time for computing gradients at the straggler determines one iteration time of the synchronous training. It increases the idle time that fast workers wait for while doing nothing. To improve the straggler problem, the previous study [8] assumed homogeneous performance environment and proposed synchronous SGD with a back-up worker, which does not wait the slowest worker for parameter update. The algorithm showed fast convergence rates by mitigating the staggers caused by dynamic factors such as resource failure or communication congestion in the homogeneous computing environment.

However, unlike the assumption of the previous study, the distributed DL training environment is usually heterogeneous [7]. Especially considering the short release cycle of new GPU types and their high improvement rates of the performance [12], the heterogeneous-performance environment is inevitable for the distributed training. The heterogeneous performance environment causes a static straggler due to the performance difference of workers, unlike the homogeneous environment assumed in the previous study. The performance difference of the workers statically determines straggler worker. Therefore, the main cause of the straggler problem in the heterogeneous performance environment is different from the homogeneous environment, so it is limited to improve the HE by applying the previous approach in the heterogeneous environment.

To solve this problem, we propose the batch-orchestration algorithm (BOA). It aims to improve computation time per iteration for the synchronous SGD by

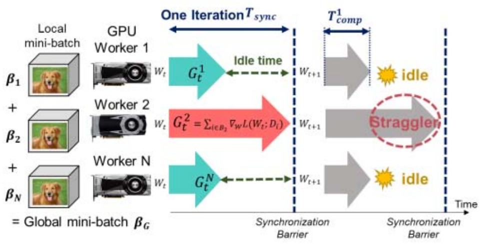


Figure 1. The straggler problem in synchronous SGD

estimating computation time of each worker. Generally, previous studies focused on algorithms for parameter update. However, we address the issue through coordinating the size of mini-batches in a computation perspective, taking into account the experimental results that the synchronous SGD leads to a poor GPU utilization in heterogeneous environments. The proposed BOA finds the estimating function of the computation time of each GPU worker with respect to a local mini-batch size and get the optimal combination of the local mini-batch sizes constraint to a given total mini-batch size.

The remainder of this paper is organized as follows. In section II, we analyze the effect of the straggler according to various heterogeneities. BOA we propose is introduced in section III. Experimental results are discussed in Section IV. Finally, we conclude in section V.

II. Straggling Effect in Heterogeneous GPU Cluster

In this section, we explain the performance heterogeneity in different types of GPUs and empirically show the effect of the straggler in the synchronous DP according to the heterogeneous level [7]. To explain the remaining part, we define two terminologies: local mini-batch size β_i, which is the number of training data used at one training iteration at the worker *i,* and the global mini-batch size β_G, which is the effective mini-batch size used to update the model parameters at the parameter server [10].

The number of CUDA cores and memory size generally determine the performance of GPUs. Different types of GPUs have different number of CUDA cores and memory size. Thus, their computation time is different when training a same DL model [6]. Therefore the cluster with heterogeneous GPUs leads heterogeneous performance, which degrades the straggler problem in the synchronous training.

To verify the relationship between the performance-heterogeneity in a cluster and the training speed of the synchronous SGD, we measured one iteration time of the synchronous SGD T_{sync} when training ResNet-98 [9] with 256 local mini-batch size at all workers for four different heterogeneous environments. Each environment consists of three GPU workers with different types as described in Table II. Table I shows the hardware specification of GPUs used as workers and one iteration time T_i to compute the gradients of the Resnet-98 at each worker *i*. For analysis, we define the straggling effect (SE) as the average value of $T_{straggler} - T_i$ for all workers *i* used in the synchronous SGD, where $T_{straggler}$ refers one iteration time to compute the gradients of the model at the straggler. The straggling effect reflects the average idle time in the parallel training. Additionally, To express the heterogeneity of the environments, we used heterogeneous level [7], which is the maximum iteration time over the minimum iteration time among workers.

Fig. 2 shows the experimental results. The heterogeneous levels increase from *environment 1* to *4*. The *environment 1* is composed of homogeneous GPUs. The *environment 4* has large performance difference among workers as described in the Table I and II, and it shows the high straggling effect. The experimental result shows that (1) one iteration time of the synchronous SGD is determined by the iteration time of the straggler, and (2) the straggler is usually determined by the performance of GPU workers in the heterogeneous environment. The high straggling effect denotes low computation utilization because the straggler causes an idle time of other workers, which leads to low training speed. Therefore, we aim to enhance GPU utilization by coordinating the local mini-batch sizes.

TABLE I. The computation time for workers

	GPU 1	**GPU 2**	**GPU 3**	**GPU 4**	**GPU 5**	**GPU 6**
GPU Type	GTX 1080 🟩	GTX 1080 🟩	GTX 1080 🟩	GTX 1060 🟪	GTX 1060 🟪	Quadro M2000 🟧
CUDA cores	2560	2560	2560	1280	1280	768
CPU	I7-4980 3.6 GHz	I7-4980 3.6 GHz	I7-6700 3.64GHz	I7-3770 3.4GHz	I7-3770 3.4GHz	Xeon® 2.10GHz
T_i **(sec)**	0.48	0.54	0.52	0.68	0.78	1.52

TABLE II. Experiment Environments to verify the SE

	Environment 1	***Environment 2***	***Environment 3***	***Environment 4***
Used Workers	GPU 1 🟩	GPU 1 🟩	GPU 4 🟪	GPU 1 🟩
	GPU 2 🟩	GPU 2 🟩	GPU 5 🟪	GPU 2 🟩
	GPU 3 🟩	GPU 4 🟪	GPU 6 🟧	GPU 6 🟧
***Heterogeneous Level* [7]**	1.125	1.42	2.24	3.17
T_{sync} **(sec)**	0.6184	0.6975	1.527	1.524
Straggling Effect	0.038	0.1645	0.800	1.01

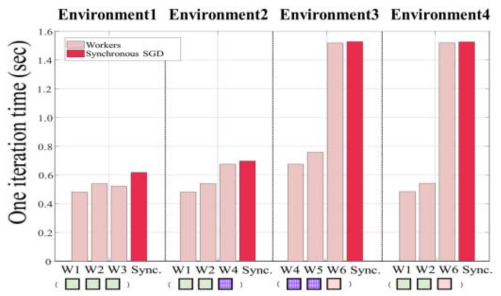


Figure 2. Experimental results of the iteration time of DP and computation time for each worker the heterogeneous GPU cluster.

III. Model Description of BOA

In this section, we introduce BOA, which is the algorithm coordinating the local mini-batch size to mitigate the straggler problem due to the performance heterogeneity. When the global mini-batch size is given as β_G, the traditional DP approaches divide the global mini-batch size by the number of workers N, to compute the same amount of gradients at each worker. However, it makes the performance difference cause the different computation time and the straggler increase the idle time of fast workers. Therefore, we aim to find the optimal local mini-batch size of each worker, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_N)$, based on the function $T^i_{comp}(\beta_i)$ that estimates the computation time of one iteration at GPU worker *i* according to the local mini-batch size β_i. By the coordinating the local mini-batch sizes, BOA mitigates the iteration time of the synchronous SGD. Fig. 3 shows the process of BOA with two major steps: GPU computation time estimation and coordinating the local mini-batch sizes.

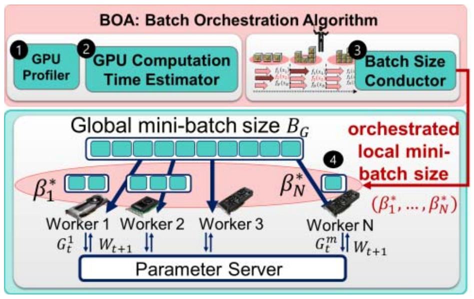

BOA: Batch Orchestration Algorithm
1 GPU Profiler
2 GPU Computation Time Estimator
3 Batch Size Conductor
Global mini-batch size B_G
orchestrated local mini-batch size
4
β_1^*
β_N^*
$(\beta_1^*, \ldots, \beta_N^*)$
Worker 1
Worker 2
Worker 3
Worker N
G_t^1
W_{t+1}
G_t^m
W_{t+1}
Parameter Server

Figure 3. Behavior of the propoed BOA in heterogeneous GPU Cluster

In BOA, computation time of one training iteration $T^i_{comp}(\beta_i)$ is modeled as a linear function of local mini-batch size β. The computation time of a DL model is the sum of computation time of each layer of the model [12]. Convolution layer, which is one of the frequently used layers, is implemented by a lowered matrix multiplication in cuDNN [11]. The convolution is computed using the matrix multiplication with dimension *(K* x *CRS)* and *(CRS* x *βPQ)*, where the mini-batch size β, input channel C, input width W, input height H, the number of filters K, the size of the filter width and height R, S are given respectively. The number of floating points for the convolution is *(2CRS-1)KβPQ*, so the computation time of the convolution layer in GPU with FLOPS$_i$ is the number of floating operations divided by the FLOPS$_i$. Thus, the computation time of the convolution layer is a constant multiple of the local mini-batch size β. Moreover, the computation of a fully connected (fc) layer is also a matrix multiplication, and its computation time is also a linear function of the local mini-batch size. Therefore, BOA puts the function of one training iteration time at GPU *i* as a linear function of local mini-batch size, $A_i \beta_i + b_i$. To find the parameters A_i and b_i of GPU worker *i*, BOA conducts a profiling. Using the profiling results, the parameters of the estimation function are driven by linear regression.

Using the estimation functions, BOA coordinates the local mini-batch size of all workers when the global mini-batch size β_G is given. BOA finds the optimal combination of the local mini-batch size, minimizing the delay of one synchronous iteration due to the straggler. One iteration time of the synchronous SGD is determined by the computation time of the straggler. Therefore BOA minimizes the maximum computation time of one iteration among GPU workers. To find the coordinated solution, BOA uses an integer linear programming (ILP) solver. The pseudo code of BOA is described in fig. 4.

```
BOA: Batch Orchestrating Algorithm
B = [$\beta^1$, $\beta^2$, … , $\beta^m$] ,t = [ ]
A = $\mathbf{I}_{N\times N}$ ,b = [ ]
# Profiling GPU (in system, parallelize the N GPU profiling tasks)
for i in [0, … , N-1]
    for j in range(length(B))
        t[i][j] = profilingGPUTime($i, j$)
    end for
end for
# Find the estimation models $A_i\beta + b_i$.
# Because it is 1-dim linear regression, the parameters are derived by
# $\frac{\partial \hat{R}(A_i,b_i)}{\partial A} = 0$ and $\frac{\partial \hat{R}(A_i,b_i)}{\partial b_i} = 0$.
# Therefore $A_i = \sum_{j=1}^{m}(\beta_i^j - \overline{\beta}_i)(t_i^j - \overline{t_i}) / \sum_{j=1}^{m}(\beta_i^j - \overline{\beta}_i)^2$ and $b_i = \overline{t_i} - A_i\overline{\beta}_i$
for i in [0, … , N-1]
    avg_B = average(B)
    avg_t = average(t[i])
    for j in range(length(B))
        den += (B[j]-avg_B) * (B[j]-avg_B)
        nom += (B[j]-avg_B)*(t[i][j] – avg_t)
    A[i][i] = nom/den
    b[i] = avg_t – W[i][i]*avg_B
    end for
end for
$\beta$ = [$B_G$/N, …, $B_G$/N]
A_eq = [1, …, 1; $\text{zeros}_{N\times N}$]
b_eq = [$B_G$, $\text{zeros}_{1\times N}$]
# Find the coordinated local mini-batch sizes
# $\underset{\beta}{minimize}\left(\max_{i\in[1,N]} A_i\beta_i + b_i\right)$ subject to $\sum_{i=1}^{N}\beta_i = \beta_G,\ \beta_i \in \mathbb{Z}$.
B* = $ILPSolver$(W, b, $\beta$, A_eq, b_eq)
return B*
```

Figure 4. Pseudo code of BOA

IV. EVALUATION AND DISCUSSION

To verify the performance of the proposed BOA, we conducted experiments. Table III shows environments of the experiments. Firstly, two GTX1080s and one GTX1060 were used as GPU workers, and each GPU was installed in one machine connected to 1Gbps network switch. The workers performed distributed training of the ResNet-98 models for CIFAR-10 data. We assumed that the global mini-batch size is 1536. BOA constructed the estimation functions of all workers. To find the functions, three profiling samples were measured for each worker as shown in fig. 5 (left). From the profiling results the estimation functions were derived. Using the functions, the step 2 of BOA found the coordinated combination of local mini-batch size (418, 607, 511). Using the optimal local mini-batch sizes, we trained the ResNet-98 model with the synchronous SGD. To analyze the performance, we compared the synchronous SGD using BOA with the traditional synchronous SGD and the synchronous SGD with one-backup worker [8]. The results of the training in fig. 5 shows that the synchronous SGD with BOA improved the training time to converge up to 22% compared to the traditional synchronous SGD.

We additionally verified the performance of the estimation function of BOA. Table IV shows the percentile error between the estimated time and the measured time for the optimal local mini-batch sizes returned by BOA. It can be seen that the percentile error of the estimation is reasonable, and the measured iteration time for the optimal combination of the local mini-batch reduces the idle time.

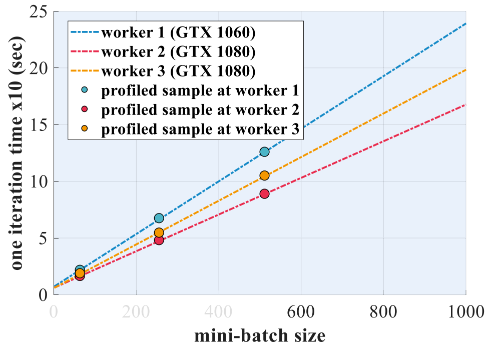
For the second experiment, we used 8 workers and made more heterogeneous environment by adding one more GPU type. The Quadro-M2000 GPU workers increased the heterogeneity of the cluster because their performances are

TABLE III. EXPERIMENTAL ENVIRONMENT

	Experiment 1	Experiment 2
Training model	ResNet-98	ResNet-194
The number of workers	3	8
Used GPU workers	2 GTX1080, 1GTX1060	4GTX1080, 2GTX1060, 2Quadro-M2000
Global mini-batch size	1536	2048

TABLE IV. VALIDATION OF THE ESTIMATION FUNCTION OF THE EXPERIMENT 1

	Orchestrated local mini-batch size	Estimated time (sec)	Measured time (sec)	Percentile Error (%)
Worker 1	418	1.041	1.035	0.57
Worker 2	607	1.042	1.032	0.93
Worker 3	511	1.042	1.020	2.17



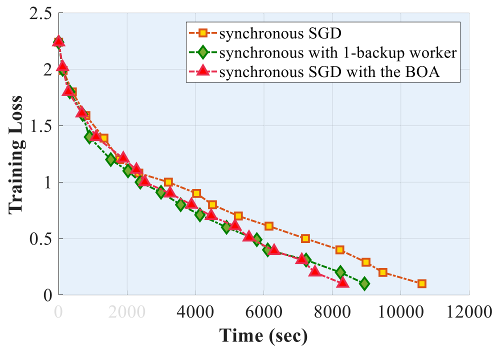
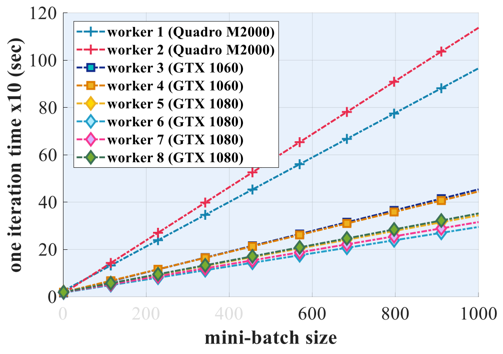


Figure 5. Results of the experiment 1: the result of BOA step 1 (left) and training loss of the experiment 1. BOA found the orchestrated local mini-batch size, (418, 607, 511). The synchronous SGD with BOA showed 22% performance improvement compared to the baseline synchronous SGD.



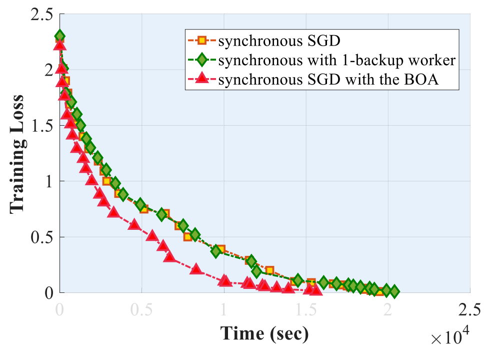


Figure 6. Results of the experiment 2: the result of BOA step 1 (left) and training loss of the experiment 2. (right) The proposed algorithm using optimal mini-batch size (49, 50, 121, 125, 155, 157, 175, 192) showed 23% performance improvement compared to using one back-up worker.

much slower than other GPU workers. The result of the second experiment is shown in fig. 6. Due to the obvious performance differences between the workers, the synchronous SGD using one back-up worker was rather inferior to the synchronous SGD. We could confirm that the synchronous SGD with BOA improves the time to converge when the heterogeneity is high in a heterogeneous GPU cluster environment.

V. Conclusion

In this paper, we proposed BOA to mitigate the straggler problem in a heterogeneous GPU cluster. To verify the performance of the algorithm, we used 8 GPU workers of three different types. The experimental results showed that by giving more local mini-batch size to GPU with better computational performance, it can improve the training speed of the synchronous SGD in the heterogeneous performance GPU cluster. Therefore the synchronous SGD with BOA showed better performance compared to other synchronous SGD. When the number of different types of GPU increases, BOA can provide good convergence speed at the synchronous SGD.

Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2016-0-00087, Development of HPC System for Accelerating Large-scale Deep Learning)

References

[1] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature* 521.7553 (2015): 436-444.

[2] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le et al., “Large Scale Distributed Deep Networks,” *NIPS 2012 Neural Inf. Process. Syst.*, pp. 1–11, 2012.

[3] P. Goyal, P. Dollar, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola et al. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour." *arXiv preprint arXiv:1706.02677* (2017), unpublished.

[4] D. Das, S. Avancha, D. Mudigere, K. Vaidynathan, et. al., “Distributed Deep Learning Using Synchronous Stochastic Gradient Descent,” 2016.

[5] S. Hadjis, C. Zhang, I. Mitliagkas, et. al., “Omnivore: An Optimizer for Multi-device Deep Learning on CPUs and GPUs,” 2016.

[6] “Deep Learning Benchmarks of NVIDIA Tesla P100 PCIe, Tesla K80, and Tesla M40 GPUs_Microway.” [Online]. Available: https://www.microway.com/hpc-tech-tips/deep-learning-benchmarks-nvidia-tesla-p100-16gb-pcie-tesla-k80-tesla-m40-gpus/.

[7] J. Jiang, B. Cui, C. Zhang, and L. Yu, “Heterogeneity-aware Distributed Parameter Servers,” *Proc. 2017 ACM Int. Conf. Manag. Data - SIGMOD ’17*, pp. 463–478, 2017.

[8] J. Chen, R. Monga, S. Bengio, R. Jozefowicz, G. Brain, and M. View, “Revisiting Distributed Synchronous SGD,” 2016, no. 2012, pp. 1–5.

[9] K. He, X. Zhang. S. Ren and J. Sun, "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[10] J. Keuper, and FJ. Preudt, "Distributed training of deep neural networks: theoretical and practical limits of parallel scalability." Machine Learning in HPC Environments (MLHPC), Workshop on. IEEE, 2016.

[11] S. Chetlur, C. Woolley, P, Vandermersch, J. Cohen and J. Tran, "cudnn: Efficient primitives for deep learning.", unpublished.

[12] Qi, Hang, Evan R. Sparks, and Ameet Talwalkar. "Paleo: A performance model for deep neural networks." (2016).