Vol.2020-HPC-177 No.3
2020/12/21

Adaptive Allocation of Computing Resources for Multiple Distributed Deep Learning Tasks

Liang Wei[1] Kazuyuki Shudo[1]

Abstract: To improve the performance of deep learning tasks, the models of Deep Neural Network (DNN) could be trained in a distributed way. That means, the DNN models are trained in a cluster environment, where the parameters for the models are refined by multiple nodes in the cluster parallelly. A number of previous works that studied distributed learning focus on improvement of one single task. In such a situation, the whole hardware resources can be fully utilized for that task. However, in a real system, it is usual to have several learning tasks running in the same cluster. So in this paper, we propose an adaptive allocation of computing resources for multiple learning tasks, with the knowledge of current learning phase for each task. In experiments we train two deep learning tasks with VGG16 networks using CIFAR10 dataset on a GPU cluster. The younger task is set to begin 200 seconds later than the start of the older task. The results show that with an adaptive adjustment for computing resources, the training time can be reduced by 4.70% with an unchanged per task batch size, and 12.76% and 7.26% for the older and younger task, separately.

1. Introduction

For the past decade, Deep Learning has become a significant tool to solve once unsolvable problems. Since training a deep learning model is a time-consuming task, it is usual that a DL task be solved distributedly using GPU clusters. Distributing a Deep Learning task over multiple GPUs on multiple nodes can accelerate the learning outstandingly compared to using CPUs on one node. However, it is still extremely hard to train a large dataset like ImageNet [1] using limited GPU resources, unless one can utilize tens of hundreds of GPUs [2]. Therefore, it is still meaningful to devise good algorithms to accelerate the training speed.

With distributed learning, the computational resources can be fully utilized, although the network cost will be incurred due to the necessity of averaging the models over nodes. Many factors about deep learning will affect the eventual performance of learning tasks, such as batch size [3] [4] [5] [6] and learning rate [9] [10].

Previous works mostly focus on one simple learning task operating in a cluster, attempting to adjust some hyper-parameters like batch size or learning rate to speed up the convergence of that task. In this paper, we found that when multiple learning tasks are at different learning phases, it is possible to accelerate the convergence of all the tasks by adjusting computing resources allocated to each task. Besides, we developed an allocator to adjust the computational resources dynamically.

The remainder of this paper will be organized as follows. Related work is discussed in Section 2. In Section 3, we introduce the background of this paper. In Section 4, we give an overview of the impact of multiple learning tasks executed in a system in the same time. And dynamic adjustment of process number will be discussed in Section 5. In Section 6, we present the experiments and results. Conclusions and future work are given in Section 7.

2. Related work

There are several existed studies about the acceleration of the learning process of deep learning.

Learning rate decay (lrDecay) is a technique for training deep neural networks. An initial large learning rate is set, and then it will be decayed by a certain factor after certain epochs. Modern DNN models are trained by SGD with lrDecay, such as ResNet [9] and DenseNet [10].

Another popular approach to accelerate deep learning is to dynamically change the batch size during the learning process. Devarakonda et al. [5] propose that with varying batch size, the learning process can achieve a speedup of 6.25x to reach almost the same test error, compared to the approach which uses fixed batch size for the whole training.

When the DNN is trained distributedly, network communication will be a factor that must be taken into account. Lin et al. [11] propose a method that before communication is conducted in all the workers across the deep learning cluster, several parameter updates on the local neural networks can attain efficiency gains. In a computational heterogeneous cluster, Chen et al. [12] show that a batch size fitting which considers the heterogeneity across the workers in real time, can achieve a speedup of training time.

3. Background

First of all, we provide an introduction to the overview of distributed deep learning. Then, the concept of learning phase will

[1] Tokyo Institute of Technology
This is an unrefereed paper.

be discussed. Lastly, we change the number of processes allocated for one single learning tasks, and present the impact on performance.

3.1 Distributed deep learning

Image classification is one of the problems that were considered unsolvable by computer programs until the birth of deep neural networks. In a discriminative way, we use a training dataset to feed the deep neural network model, updating the parameters for the network gradually based on the loss rate, and aim to obtain a network model that could achieve high accuracy on a test set.

The update of parameters is performed by Gradient Decent (GD) method to find the optimal parameters. The traditional way to utilize the GD method is to make the model traverse all the dataset, then compute the prediction label for each image sample. At last, the gap between computed labels and real labels will be measured using a loss function, and the parameters of the model will be updated using the GD method based on the loss function.

Since the huge number of data samples in datasets like CIFAR10 and Imagenet, it is unfeasible to compute the loss function for the entire dataset in one round of parameter update. Therefore, the mini-batch Stochastic Gradient Decent (mini-batch SGD) is a more fashionable way to train models on big datasets.

In mini-batch SGD, the objective is to minimize an loss function $F(\theta)$ with respect to model parameters, which is denoted by θ. The training set will be divided into several components: $\mathcal{S} = \{s_1, s_2, ..., s_n\}$, where each s_i is one batch, and the size of the batch $|s_i|$ is called batch size. In one iteration of deep learning, a mini-batch ξ_k will be sampled from training set, then the model parameters are updating using the rule:

$$\theta_k \leftarrow \theta_{k+1} - \eta \cdot \nabla F(\theta)$$

where η denotes the learning rate, and $\nabla F(\theta)$ denotes the first-order gradient of loss function.

In distributed deep learning, the computation is parallelized across multiple processors in parallel computing environments. There are two different ways to parallelize the learning task: data parallelism and model parallelism. Data parallelism is the parallelism where the dataset is distributed across different nodes. On the other hand, the neural network model itself will be distributed in a model parallelism way. We focus on data parallelism in this paper.

Distributed deep learning is different from traditional deep learning in that each node in a cluster is only responsible for a part of the dataset, the models on different nodes must be shared with other nodes. The common way is that each node sends the gradient of the model of itself using all-reduce, meanwhile receive the gradients of the models on other nodes. Then the gradients will be averaged, and be used to update the parameters. Although some researchers have shown that before communicating with other nodes, updating the local model several times will reduce the total time, we choose to do the all-reduce at each end of iteration for the sake of simplicity.

3.2 Learning stages

Liu et al. [6] presented adapting different batch sizes for different period in the learning task can result in faster convergence. The reason is that at the begining of training, the value of loss for the model is quite large, so the descent of loss could be sharply no matter how many batch size is used in one round of update. Thus, because of the smaller batch size, the time spent to finish the same number of iterations could be reduced.

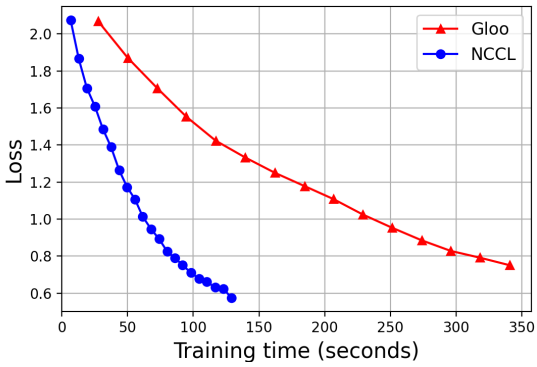

Fig. 1 Comparison of training speed of Gloo and NCCL, sampled by every epoch.

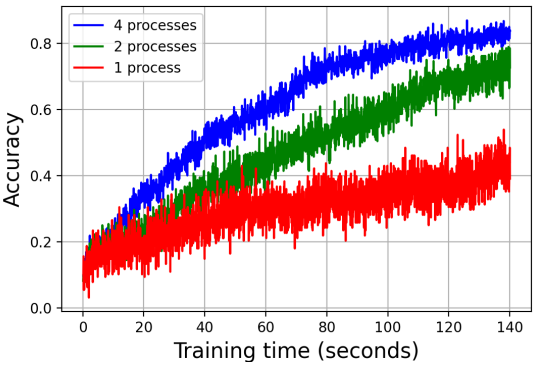

Fig. 2 Accuracy of training time with varying number of processes.

On the contrary, when the training process lasts for a certain period, the descent of loss function becomes smaller and smaller, the correct update of parameters is demanded, comparing to the time consumed in one iteration. That means although the time for one update could be speeded up, due to the lack of generalization incurred by the selection of small batch size, updating the parameter to the optimal would become very hard. As a result, reaching the convergence state would take more time conversely. Therefore, switching to a larger batch size helps to alleviate this problem.

3.3 Number of processeses for single task

When training a deep learning model on a GPU cluster, the number of processes is usually set to the same number of GPUs. However, under certain conditions, the number of total processes being trained can exceed that of the GPUs.

To do all-reduce across all the processes of the same learning task, we need some library to do all the low-level communication and data transfer. Deep learning frameworks provide several such libraries. For example, **Gloo** and **Nccl** are two communication libraries prepared in Pytorch [7], which are called backends in the

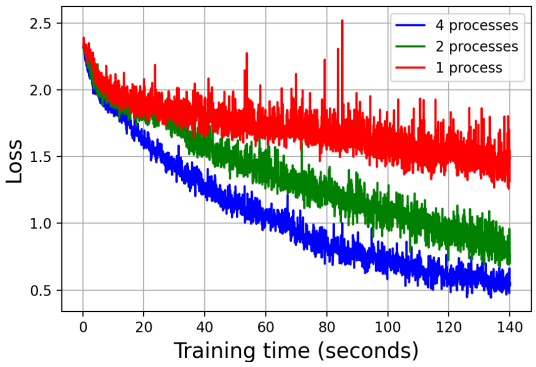

Fig. 3 Loss rate of training time with varying number of processes.

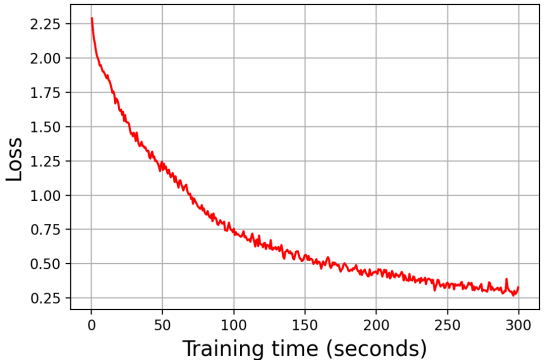

Fig. 4 Single task when trained by 4 GPUs.

Pytorch context.

The main difference is that Nccl is highly optimized for direct GPU communication, so it currently provides the best distributed GPU training performance. While Gloo is available to be utilized in GPU training too, the current implementation in PyTorch does not utilize all the computational ability of GPU cards, and that results in the slower speed of Gloo compared to NCCL. In experiments, with 4 processes on 4 GPUs, each process only use about 30% of the power of one single GPU card. Fig. 1 shows the learning curve of Gloo and NCCL in which a VGG16 [8] network is trained on a 4-GPU cluster using CIFAR10 dataset and the task was parallelized by 4 processes. The batch size for each process is set to 128.

4. Multiple Learning Tasks

As Liu et al. [6] revealed, for two tasks which are in different learning stages, the task at the early stages could use a smaller batch size to reduce the time for one iteration, while a larger batch size is needed for the task at late stages to perform an accurate update of parameters.

Based on this thought, we make two deep learning tasks start at different times, and change the number of allocated processes for each task when the younger task starts training. we assume that the cluster has 4 GPUs and there will be 4 processes. Fig. 4 shows the loss rate of the VGG16 network when training using 4 GPUs.

The gap between the younger task and the older task is set to the time of 200 seconds. Besides, we set the criterion of the convergence of models to the loss rate achieved when the model has been trained for 300 seconds. When the younger task begins its training process, since the GPU resources are necessary, the older task has to release some GPU cards. At first the system will make an effort to split up the GPU cards between each learning task. In this scenario, each task will be allocated 2 GPUs, respectively. Since the available GPUs for each task has reduced, the learning processes suffer performance degradation to some extent. After the older task finished, the younger one can make use of all the GPU cards to continue the remaining learning process. The loss rate with time consumption is displayed in Fig. 5.

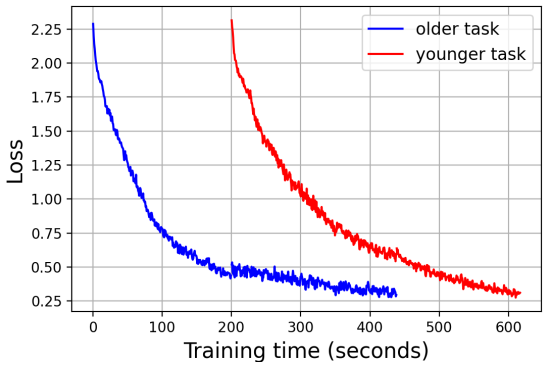

Fig. 5 Two tasks when trained by 4 GPUs, each task is allocated 2 GPUs when they are being trained simultaneously. Batch size for each process is set to 128.

5. Adaptive adjustment of number of processes

Adjustment of the number of the process for each task effectively changes the respective usage of GPU resources. In this section, we propose a naive adjustment method for the number of processes, which automatically determine the adequate number applied to each task. The basic idea of automatic adjustment is that the need for computational resources varies when the learning process goes on. As concluded in Section 2, the learning task which is at the later stage tends to utilize more data samples to complete one update of the parameters. This will be verified in experiments where the batch size for each process remains a constant number. Furthermore, by varying per process batch size, but keep total batch sizes equal for all the tasks, we can accelerate the convergence of tasks which begin its training process earlier. Then the trade-off occurs to shorten the overall training time. Therefore, we aim to design a GPU resource allocator that has access to the information of the ongoing tasks and make decisions on the number of processes based on the collected information.

The motivation to overlap the execution of multiple learning tasks instead of finishing the training one after another is from the assumption that when lesser GPUs are used for training, the overhead of communication among the processes will be reduced.

To figure out at which stage the current task is, the allocator will use information about the delta of loss rate. According to Fig. 4, the delta of loss rate differs across different phases in the learning process. Generally, the delta of loss tends to decrease while the training continues. In the gradient descent method, the

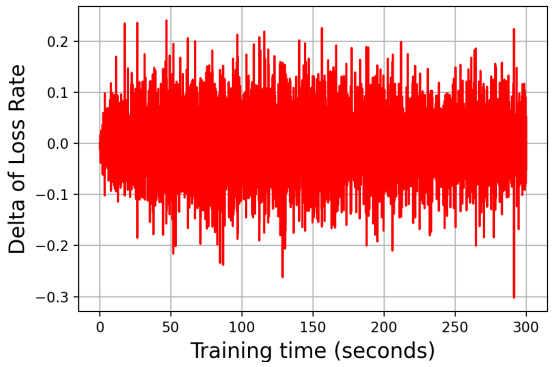

Fig. 6 Delta of loss rate in each iteration, number of processes is 4.

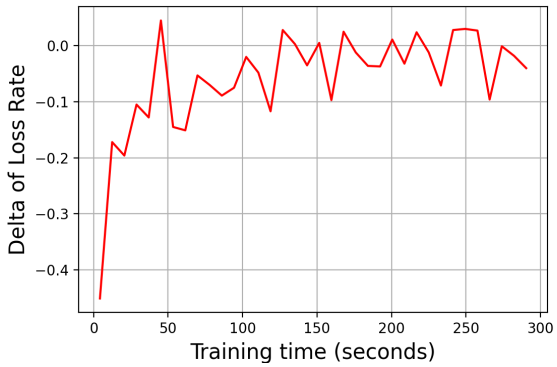

Fig. 7 Delta of loss rate in 100 iterations, number of processes is 4.

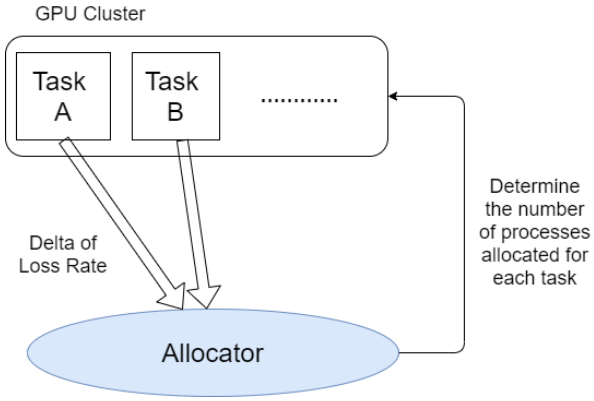

Fig. 8 Architecture of the allocator

loss rate is used to measure how far the targets of each batch data samples computed by the neural network model is from the actual labels. As shown in Fig. 6, the delta of loss rate in each iteration is a random number and is not sufficient to use to determine the learning stage. However, since the loss rate is destined to decline after all, during a long enough time, we can observe the difference in different periods.

When we set the period equal to 100 iterations of training, the contrast will be more clear. Fig. 7 shows the delta of loss rate with time consumption, while the delta is the sum of every 100 iterations.

The architecture of the allocator is displayed in Fig. 8. The whole deep learning system is managed by the allocator, which means that every time a new task prepares to start training, the allocator should be notified. Besides, the allocator needs to have access to the details of the situation of each task (e.g., loss rate, time consumption, allocated number of processes).

This brings about the first limitation of our research. As a consequence of the necessity that the allocator should obtain all the information of the ongoing tasks, an outside task which is not under the charge of the allocator will fail the system, since there exists the possibility that different tasks use the same GPU card. Another limitation is that we only concern about the scenario that a new task using non-pre-trained models while the ongoing task is about to converge.

Actually the proposed method in this paper is only able to handle 2 learning tasks at most. The method consists of a couple of steps.

(1) When a new task is about to begin training, the allocator is notified.
(2) The allocator continues the training process of each task for 100 iterations, and calculates the delta of loss rate.
(3) Based on the information collected, the allocator will judge which task is older and which is younger. This determination is conducted simply by checking which task's loss delta is larger.
(4) The allocator will decide to allocate approriate number of processes for all the tasks.

Regarding step 4, since the cluster we use in experiments has 4 GPU cards, therefore we can only apply the ratio of number of processes for older-younger task of 3 to 1 and 2 to 2 to verify the effect of accelerate the execution of older task. In fact, there are occasions that a machine has more than 4 GPU cards attached to itself, thus a different GPU cards ratio other than 3 vs 1 and 2 vs 2 is possible. How to determine the ratio for computational resources is considered a future work.

6. Experiments

In this section, we will describe the experimental setup. Then, the results will be shown with different policy for batch size. Besides, the reasons for the results will be discussed.

6.1 Experimental setup

The experiment environment is described in this subsection. Table 1 lists the hardware used in all the experiments. All the experiments are conducted on a single server which has 4 GPU cards attached to itself. PyTorch framework is used to implement the system. The neural network model to be used in experiments is VGG16. The loss function we used to calculate the loss rate of the model is cross entropy loss function. Dataset is CIFAR10. Since the communication between nodes does not happen, the network configuration is omitted.

Table 1 Experimental setup

CPU	Intel Xeon CPU E5-2698 v4 @ 2.20GHz
GPU	Tesla V100-PCIE-32GB
OS	Ubuntu 16.04 TLS Linux-4.4.0

With only 4 GPUs are available in the server, there are 3 ways to allocate the processes to older task and younger task: 1-3, 2-2, 3-1. While *i-k* means ratio of allocated processes to older task

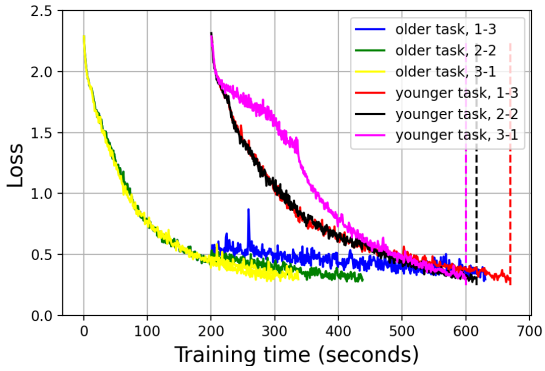

Fig. 9 Learning curves with batch size set to 128 per process.

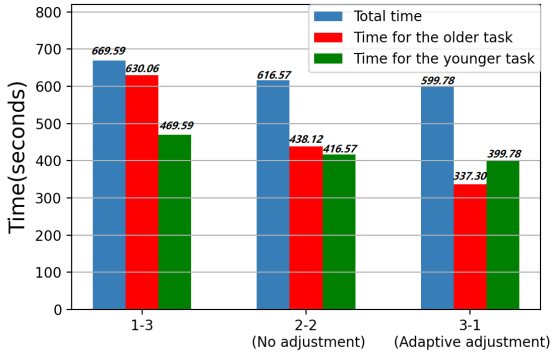

Fig. 10 Time comparison with batch size set to 128 per process.

and younger task is set to i to j. In our experiments, we use ratio of **3-1** and **2-2** to indicate training **with** or **without** adjustment, respectively. The experiments of ratio of **1-3** are conducted for completeness.

In addition to adjustment of computing resources, the dataset will be reallocated for each process as well. more, the batch size will also have to be adjusted if neccessary.

6.2 Batch size equal across processes

In experiments, the older task will begin training at first. Then the younger task will start after 200 seconds. At the meantime, the allocator will determine which task need more GPU resources, and allocates 3 GPUs to that one.

After the younger task joins the system, allocator will observe the loss rate of both tasks for the first 100 iterations. Then the allocation of GPUs is carried out by allocator based on the delta of loss rate.

First of all, we set the batch size, which is 128, be equal across all the processes. That means for tasks which have different number of process will have different batch size settings. Certainly, applying the same batch size for each process leads to the fact that adjustment of computating resources will cause adjustment of total batch size for the tasks at the meantime, which makes it ambiguous that which kind of adjustment contributes more to the performance improvement.

Because the curve for loss rate is quite random, in order to determine whether one task converged or not, we use the average loss rate value for 10 iterations.

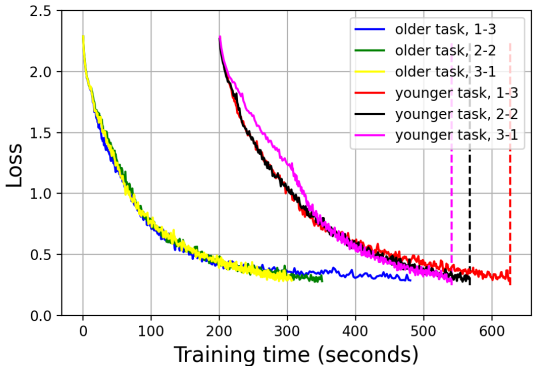

Fig. 11 Learning curves with batch size set to 512 per task.

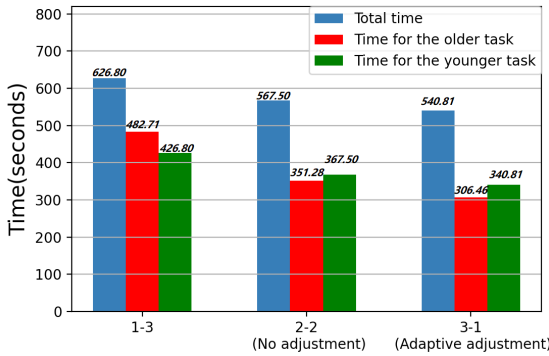

Fig. 12 Time comparison with batch size set to 512 per task.

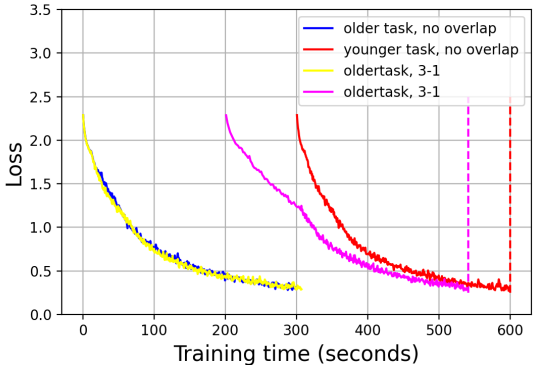

Fig. 13 Learning curves comparison between overlap and non-overlap training, with batch size set to 512 per task.

Fig. 9 shows the learning curves under 3 different process ratio. Fig. 10 shows the time consumption in each part. During the period when two tasks coexist, if the allocator allocates more processes to the older task (3-1), it will reach to the convergence point 100.82 seconds earlier, compared to the one with no adjustment (2-2). As a result, the training time can achieve acceleration of time efficiency of 2.72%. For each task, the training time is shortened by 23.01% for older task, and 4.03% for younger task, respectively.

This proves that our intention to enlarge the actual batch size for older task by changing the computing resources ratio is workable to shorten the overall training time.

Table 2 Experimental results of time reduction

	Reduced by	
	Batch size 128 per process	Batch size 512 per task
Total	2.72%	4.70%
Older task	23.01%	12.76%
Younger task	4.03%	7.26%

6.3 Batch size equal across tasks

In this group of experiments, we keep the batch size equal even the numbers of processes for tasks are changed. Concretely, the batch size will be set to 512, which is the number of batch size for one single task. In our experiments, the batch sizes for task which has three processes will be set to 170, 171 and 171. There are some occasions that the batch size is not divisible by the number of processes. In that case, the batch size for one specific process is adjusted to assure that 512 be the sum of all the batch sizes. Thus, the impact of batch size alteration on performance is eliminated.

Fig. 11 and 12 show the learning curve and time comparison when training with 3 different process ratio. The total training time is reduced 4.70% from 567.50 seconds to 540.81 seconds. For each task, the training time is shortened by 12.76% for older task, and 7.26% for younger task, respectively. Fig. 11 also shows that total training time is extended with lesser computing resources allocated to older task. Furthermore, Fig. 13 shows that training by overlapping two learning tasks while applying adaptive computing resources adjustment is faster than training without overlapping.

Therefore, our assumption that the trade-off for shortening the execution of older task could improve overall performance is proven by the results. In addition, by overlapping the execution of multiple tasks, the hardware is fully utilized with lower overhead.

At last, the experimental results of reduction of training time under different settings of batch sizes, compared between training with adaptive adjustment and without adjustment, is shown in Table 2.

7. Conclusion

We showed that in a deep learning system which have multiple ongoing learning tasks, the tasks will affect the performance of each other. Specifically, due to the finite number of GPUs and the fact that one GPU cannot be shared among multiple learning tasks, older tasks have to withdraw the occupation of some GPUs younger tasks are able to begin its training process using GPU. Then we showed that the training time both for two tasks and for only the older task can be shortened by changing the number of allocated processes for each task, based on the information of learning process. Furthermore, we proposed a naive method to detech which phase the specific task in on. We conducted experiments with batch size equal across all processes and across all learning tasks. The results of experiments showed that with adaptive adjustment of number of processes, the decrease in training time for all the tasks can be achieved.

We proved that the trade-off for shortening the execution of older task could improve overall performance. In addition, by overlapping the execution of multiple tasks, the hardware is fully utilized with lower overhead.

In this paper, we only investigate the scenario in which VGG16 networks are trained using CIFAR10 dataset. The solution for different models to be trained using different datasets is left for future work. The number of learning tasks is limited to two in our research. Actually, the number of tasks being executed simultaneously in a cluster may be more than that. The method to allocate appropriate computing resources for more than two tasks is neccessary. Moreover, the GPU cards used in this research reside on the same node. For deep learning tasks which utilize mulitple nodes connected by Ethernet, the time for network communication will also affect the performance. This is another future work. Besides, it is also future work to consider the number of more than 4 GPUs.

Acknowledgments This work was supported by New Energy and Industrial Technology Development Organization (NEDO).

References

[1] Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei: Imagenet: A large-scale hierarchical image database, *In 2009 IEEE conference on computer vision and pattern recognition*, pp. 248-255, IEEE (2009).

[2] Yamazaki, Masafumi, Akihiko Kasagi, Akihiro Tabuchi, Takumi Honda, Masahiro Miwa, Naoto Fukumoto, Tsuguchika Tabaru, Atsushi Ike, and Kohta Nakashima: Yet another accelerated sgd: Resnet-50 training on imagenet in 74.7 seconds, *arXiv preprint arXiv:1903.12650*, (2019).

[3] Keskar, Nitish Shirish, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang: On large-batch training for deep learning: Generalization gap and sharp minima, *arXiv preprint arXiv:1609.04836*, (2016).

[4] Balles, Lukas, Javier Romero, and Philipp Hennig: Coupling adaptive batch sizes with learning rates, *arXiv preprint arXiv:1612.05086*, (2016).

[5] Devarakonda, Aditya, Maxim Naumov, and Michael Garland: Adabatch: Adaptive batch sizes for training deep neural networks, *arXiv preprint arXiv:1712.02029*, (2017).

[6] Baohua Liu, Wenfeng Shen, Peng Li, and Xin Zhu: Accelerate Mini-batch Machine Learning Training With Dynamic Batch Size Fitting, *In 2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8. IEEE (2019).

[7] Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen et al: Pytorch: An imperative style, high-performance deep learning library, *In Advances in neural information processing systems*, pp. 8026-8037 (2019).

[8] Simonyan, Karen, and Andrew Zisserman: Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv:1409.1556*, (2014).

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

[10] Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger: Densely connected convolutional networks, *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700-4708 (2017).

[11] Lin, Tao, Sebastian U. Stich, Kumar Kshitij Patel, and Martin Jaggi: Don't Use Large Mini-Batches, Use Local SGD, *arXiv preprint arXiv:1808.07217*, (2018).

[12] Chen, Chen, Qizhen Weng, Wei Wang, Baochun Li, and Bo Li: Fast distributed deep learning via worker-adaptive batch sizing, *In Proceedings of the ACM Symposium on Cloud Computing*, pp. 521-521 (2018).