DEIM2021 E14-4

Adaptive Allocation of Computing Resources for Multiple Distributed Deep Learning Tasks

Liang WEI[†] and Kazuyuki SHUDO[†]

† Tokyo Institute of Technology

Abstract To improve the speed of deep learning tasks, the models of Deep Neural Network (DNN) could be trained in a computational node where the parameters for the models are refined by multiple GPUs in the node parallelly. A number of previous works have studied parallel learning focus on the 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 node. 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 VGG-16 models using CIFAR-10 dataset on a node with multiple GPUs, and the results show that with adaptive adjustment for computing resources, the training time can achieve a decrease of 4.70% with unchanged per task batch size. Besides, the experiment that mixing up ResNet-50 and VGG-16 models shows that this computing resources adjustment is effective for different models. Furthermore, the results show that network overhead and the GPU memory limitation could impair the training speed.

Key words Distributed deep learning, GPUs, computing resources

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 parallelly using multiple GPUs. Distributing a Deep Learning task over multiple GPUs 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 parallelization, the computational resources in one node can be fully utilized to speed up the training process. 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.

(注) : This is an unrefereed paper.

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

To accelerate a deep learning task, one feasible way is to change the hyper-parameters dynamically during training process. Learning rate decay (lrDecay) is a technique to adjust the learning rate as the training proceeds. By using different learing rate in different learning phases, the models could achieve a faster convergence speed. Modern DNN models are trained by SGD with lrDecay, such as ResNet-50 [9] and DenseNet [10].

Another popular approach is to adjust the batch size as the training proceeds. 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. This paper researches the relationship between convergence time and computational resources, although one experiment about varying batch size setting is conducted, we mainly focus on the experiments where the batch sizes for the tasks are the same.

There have been number of techniques proposed for computing resources allocation for deep learning tasks. Takahashi et al. [12] proposed a static scheduling framework to profile the processing time for several training tasks, and schedule the tasks across a CPU cluster based on that result. Chaudhary et al. [13] designed a algorithm to share the GPU throughput among cluster users fairly using time-slicing method in a heterogeneous computational environment. The computational resources used in this papre are GPUs, and each has the same performance.

3 Background

First of all, we provide an introduction to the overview of parallel deep learning. Then, the concept of learning phase will be discussed. Lastly, we change the number of processes allocated for one single learning tasks, and present the impact on performance.

3 1 Parallel 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. Since the huge number of data samples in datasets like CIFAR-10 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.

Furthermore, training several models parallelly in a computational node using multiple CPUs or GPUs could speed up the training time. When training parallelly, some communication methods such as gossip or all-reduce are necessary for synchronizing the parameters of different models across processes.

Fig. 1 and Fig. 2 show the accuracy and loss rate curve of training of a VGG-16 network under different settings for number of process.

3 2 Learning phases

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

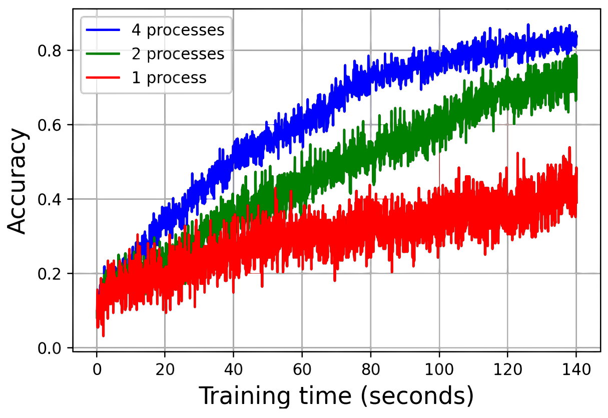

Figure 1 Accuracy of VGG-16 model with varying number of processes.

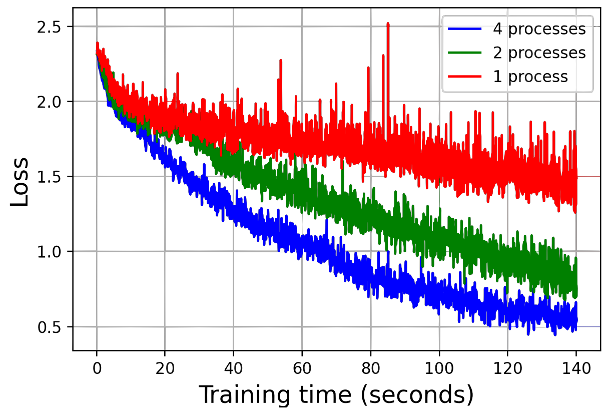

Figure 2 Loss rate of VGG-16 model with varying number of processes.

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.

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 node, the number of processes is usually set to the same number of GPUs.

To do all-reduce across all the processes of the same learn-

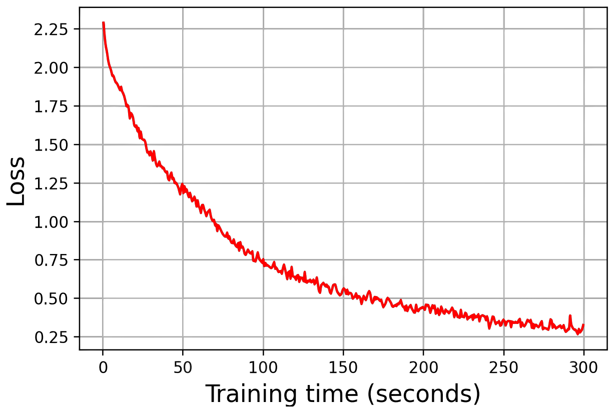

Figure 3 Single task when trained by 4 GPUs.

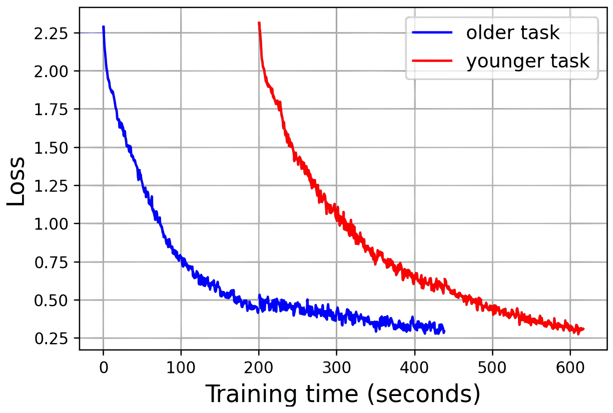

Figure 4 Two tasks when trained by 4 GPUs, each task is allocated 2 GPUs when they are being trained simultaneously. Batch size for each process is set to 128.

ing task, we need some library to do all the low-level communication and data transfer. Deep learning frameworks provide several such libraries. **NCCL** is one communication backend which is prepared in PyTorch [7], and since it currently provides the best distributed GPU training performance, NCCL is used in this paper.

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 node has 4 GPUs and there will be 4 processes. Fig. 3 shows the loss rate of the VGG-16 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 criteria of the convergence of models to the loss rate achieved when the model has been trained for 300 seconds. The reason why we set the loss rate instead of the number of iterations is that, we define the convergence of a neural network model to be the loss rate on datas. 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. 4.

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. 3, 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 loss rate is used to measure how

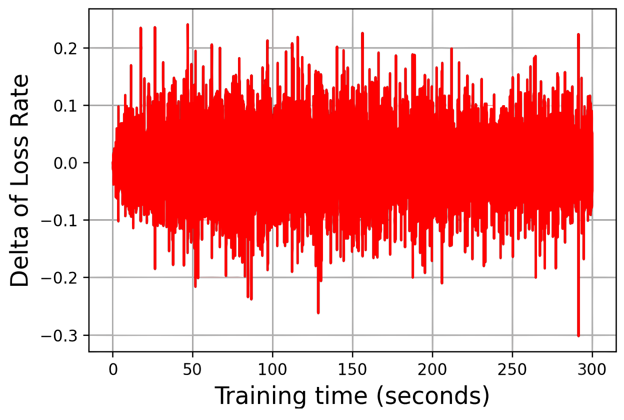

Figure 5 Delta of loss rate in each iteration, number of processes is 4.

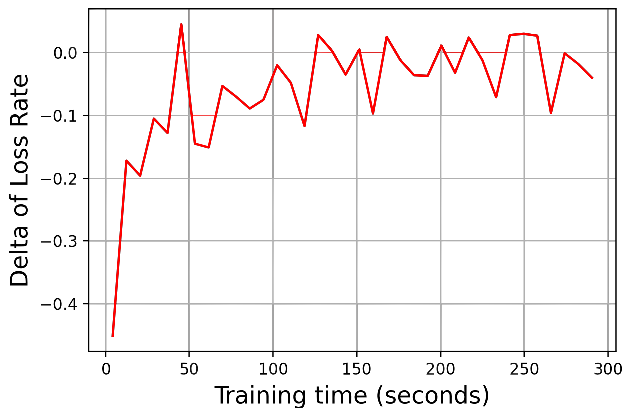

Figure 6 Delta of loss rate in 100 iterations, number of processes is 4.

far the targets of each batch data samples computed by the neural network model is from the actual labels. As shown in Fig. 5, 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. 6 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. 7. 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 ob-

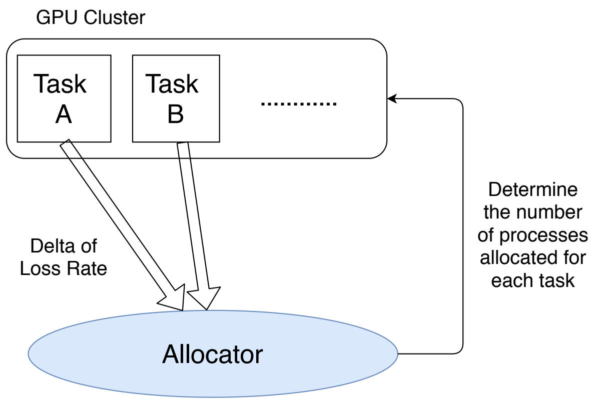

Figure 7 Architecture of the allocator

tain 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.

The proposed 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 node 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 (e.g. 5 vs 3, 6 vs 2 or 7 vs 1) of computational resources for that scenario 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

Table 1 Experimental setup

CPU	Intel Xeon CPU E5-2698 v4 @ 2.20GHz
GPU	NVIDIA Tesla V100-PCIE-32GB * 4
OS	Ubuntu 20.04 LTS Linux-5.4.0

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 VGG-16. The loss function we used to calculate the loss rate of the model is cross entropy loss function. Dataset is CIFAR-10. Since the communication between nodes does not happen, the network configuration is omitted.

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 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. The condition to terminate a learning task is that when the loss rate reaches the same level as one single tasks achieves using all the 4 GPUs to train 300 seconds.

In addition to adjustment of computing resources, the dataset will be reallocated for each process as well. Moreover, 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. 8 shows the learning curves under 3 different process ratio. Fig. 9 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.

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. 10 and 11 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. 10 also shows that total training time is extended with lesser computing resources allocated to older task. Furthermore, Fig. 12 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.

6 4 Tasks using different neural network models

To verify that this adjustment is effective for the scenario where different types of model are used in tasks, we conducted an experiment that Resnet-50 and VGG-16 models were applied to the older and younger tasks respectively.

The batch size setting remains the same as that in the last subsection, i.e. each task computes 512 pieces of images per iteration.

In this experiment, we compared the training convergence speed between process ratio of (2-2) and that of (3-1). The condition used to judge whether each task reaches the convergence state remains the same as previous experiments stated

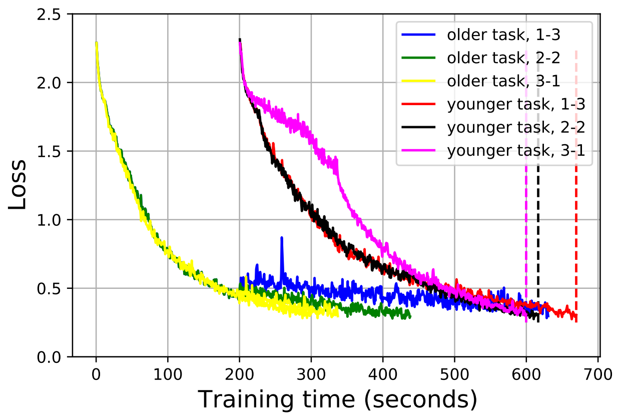

Figure 8 Learning curves with batch size set to 128 per process, VGG-16.

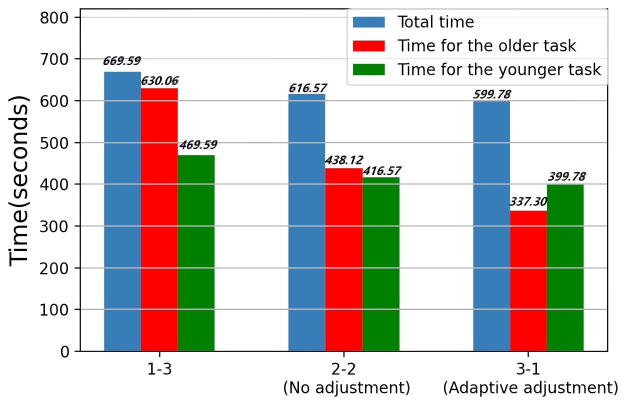

Figure 9 Time comparison with batch size set to 128 per process, VGG-16.

in this paper.

Fig. 13 shows the learning curve of the experiment which the Resnet-50 and VGG-16 models are applied to the older task and the yonger task respectively. The total training time is reduced 7.11% from 604.25 seconds to 561.26 seconds, and the reduction percentage is 11.20% and 10.63% for the older task and the younger task. These results demonstrate that the adjustment is effective for tasks that are using different neural network models.

6 5 Learning between different computation nodes

We introduce the experimental results about distributed deep learning between nodes that are connected by LAN in the last of this section. Two computation nodes are used in this experiment, one of which is identical to the node used in previous experiments. While the other one has the same CPU and OS settings, except for GPU, which has the same computing power as another node's but has only 16 GB GPU memory.

Similar to the previous experiments, after the execution of the older task, all the GPUs are available to it.

The younger task starts 200 seconds after the execution of the older task. However, the dynamic adjustment is not used here, the ratio of number of processes for older and younger tasks are set to **7-1**, **6-2**, **5-3** and **4-4**, as there are 8 GPUs totally.

The condition to terminate a learning task is that when the loss rate reaches the same level as one single tasks achieves using all the 8 GPUs to train 300 seconds.

The results are showed in Fig. 14. Contary to the conclusions obtained in previous subsections, allocating more GPUs for the older tasks has no effect on improving the overall performance. The reason is that, due to the network latency and the time consumed to transit the model data via LAN, the improvement with more GPUs is not that obvious. Besides,

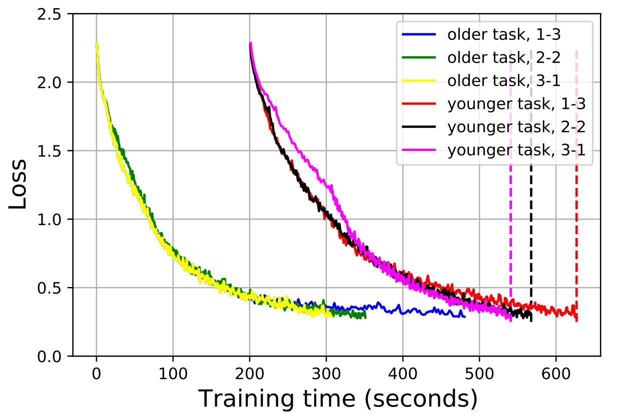

Figure 10 Learning curves with batch size set to 512 per task, VGG-16.

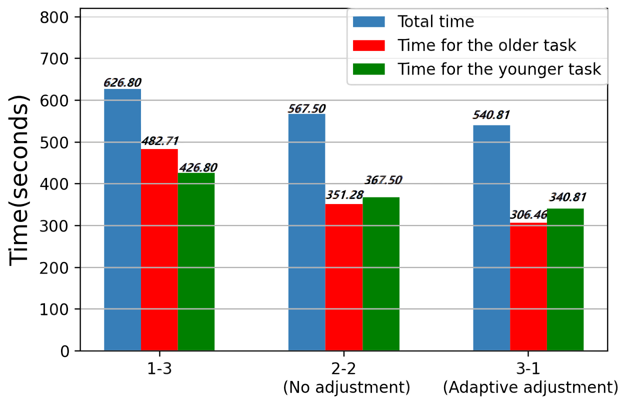

Figure 11 Time comparison with batch size set to 512 per task, VGG-16.

to maintain the same batch size in one task, the younger task, which has less number of GPUs during overlap, has to feed much more data into GPU than the GPU memory can afford. As a result, the younger task need to free the GPU memory every few seconds, which leads to the slow down of

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%

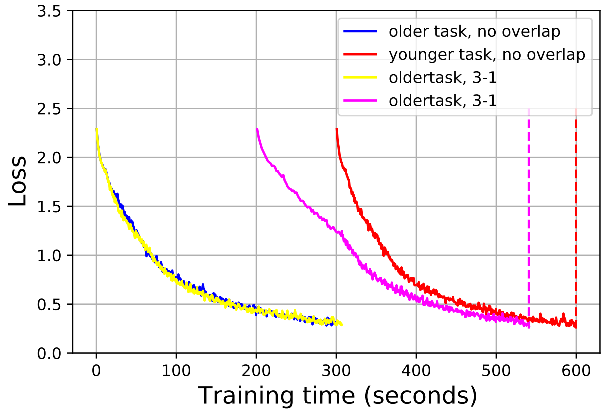

Figure 12 Learning curves comparison between overlap and non-overlap training, with batch size set to 512 per task, VGG-16.

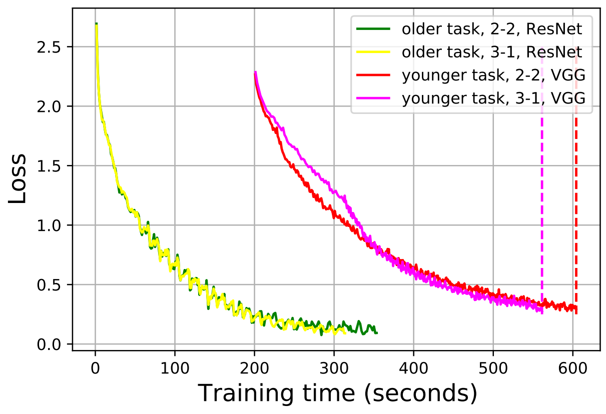

Figure 13 Learning curves with the older task using ResNet-50 and the yonger task using VGG-16, batch size set to 512 per task.

training.

Futhermore, the results show that the older task converges fastest with 4 GPUs. The reason is the same that increase in computational resources is not able to compensate for the overhead incurred in additional network communication.

7 Conclusion

We have showed that in a deep learning system which have multiple ongoing learning tasks, the tasks will affect the performance of each other. Then we showed that this problem can be solved by change the number of allocated processes

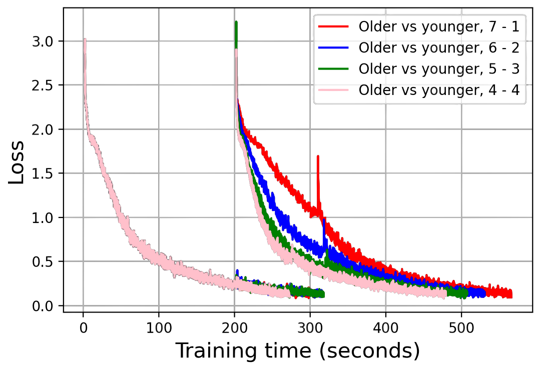

Figure 14 Learning curves of 2 tasks, 8 GPUs totally.

for each task, based on the information of learning process. Furthermore, we propose 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 learning 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.

Furthurmore, we showed that in the computational environments, using GPUs in different nodes connected by network will incur communication overhead. Besides, under the limitation of GPU memory, maintaining the per-task batch size while using less GPUs could slow down the training speed.

In this paper, the number of learning tasks is limited to two in our research. Actually, the number of tasks being executed simultaneously in a node may be more than that. Besides, in a node that has more than 4 GPUs, there's more room for adjustment of computing resources. The algorithm to allocate appropriate computing resources in such scenario is a future work.

Acknowledgement

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 IEEE conference on computer vision and pattern recognition, 2009, pp. 248-255.

[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,' 2019, *arXiv:1903.12650*.

[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," 2016, *arXiv:1609.04836*.

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

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

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

[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, 2019, pp. 8026-8037.

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

[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, 2016, pp. 770-778.

[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, 2017, pp. 4700-4708.

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

[12] Takahashi, Yoshiki, Masato Asahara, and Kazuyuki Shudo, "A Framework for Model Search Across Multiple Machine Learning Implementations," 15th International Conference on eScience, 2019, pp. 331-338.

[13] Chaudhary, Shubham, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha, "Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning," in Proceedings of the Fifteenth European Conference on Computer Systems, 2020, pp. 1-16.