DELAYED WEIGHT UPDATE FOR FASTER CONVERGENCE IN DATA-PARALLEL DEEP LEARNING

Tetsuya Youkawa, Haruki Mori, Yuki Miyauchi, Kazuki Yamada, Shintaro Izumi, Masahiko Yoshimoto, and Hiroshi Kawaguchi

Graduate School of Science, Technology and Innovation, Kobe University, Kobe, Japan
E-mail: youkawa.tetsuya@cs28.cs.kobe-u.ac.jp

ABSTRACT

This paper presents a proposal of a data-parallel stochastic gradient descent (SGD) using delayed weight update. A large-scale neural network appears to solve advanced problems, but its processing time increases concomitantly with the network scale. For conventional data parallelism, workers must wait for data communication to and from a server during weight updating. Using the proposed data-parallel method, the network weight has a delay. It is therefore stale. Nevertheless, it gives faster convergence time by hiding the latency of the weight communication for the server. The server concurrently carries out the weight communication and weight update while workers calculate their gradients. The experimentally obtained results demonstrate that, in the proposed data parallel method, the final accuracy converges within degradation of 1.5% compared with the conventional method in both VGG and ResNet At maximum, the convergence speedup factor theoretically reaches double that of conventional data parallelism.

Index Terms— Data synchronous parallelism; Delayed weight update; Distributed learning

1. INTRODUCTION

A convolutional neural network (CNN) imitates a part of the human visual cortex in the cerebrum. The original CNN, named *neocognitron*, was developed for handwritten character recognition [1]. CNNs have been scaled up with numerous synapses and neurons in deeper layers. Recently, a deeper network having more than three layers is generally designated as *deep learning*. Today, deep learning is applied mainly to computer vision applications, but it has general-purpose characteristics and capabilities [2]. Because deep learning has generality with a deeper and larger-scale network, error rates of recognition continue to improve. Accordingly, training times become much longer. AlexNet took 5–6 days to train 90 epochs of 1.2-M ImageNet benchmark datasets [3] on two NVIDIA GTX580 GPUs [4]. Actually, ResNet with 200 layers (ResNet200), which was designed for ImageNet, took three weeks for training even with eight GPUs used in parallel computing [5]. In deep learning, training a single GPU is virtually impossible for a practical benchmark to be processed in a realistic time. Therefore, distributed deep learning has received a remarkable amount of attention.

Two concepts exist for distributed deep learning to shorten the training time for an enormous network [6–7]: (1) Model parallelism has divided dimensions of a model (network). Each worker trains a different part of the model (network). (2) Data parallelism has divided dimensions of data. Each worker trains on the same network, but with a different data example. Data parallelism is mainstream because it is generic and easily implemented with homogeneity, irrespective of the processor.

To apply data parallelism for additional speeding up, multithreaded mini-batch stochastic gradient descent (SGD) is often used. For homogeneous workers, implementing the same software for them is simple. Each worker has the same network, but each processes a different mini batch. In other words, a single network is trained with different mini batches. Each worker calculates different gradients. A server usually updates a weight by averaging the gradients received from all workers. Then, the updated weight is sent back to the workers for the next mini-batch step. The multithreaded data-parallel SGD tends to be less effective in convergence than a single-threaded one because its actual batch size is multiplied by the data parallelism [8–10]. By virtue of layer-wise adaptive rate scaling (LARS) [11], the convergence efficiency is improved even for batches as large as 32,000. Several reports have described that training Resnet-50 with ImageNet dataset can be completed in tens of minutes using more than 1,000 GPUs [12–14].

However, with data parallelism, bottlenecks arise from communication between a server and workers. Particularly in a huge model with many parameters, the communication time increases concomitantly with the number of workers: for learning Resnet-50 with 1,024 Tesla P100 GPUs, the communication time occupies a quarter of the total time,

This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO), Japan.

although parameter communication is optimized to half accuracy [14]. Communication time will be even greater for huge models expected in future applications. In this paper, we propose delayed update in synchronous data parallelism by which all workers calculate gradients using weights delayed by one step. Techniques for updating with old gradients have already been discussed [15-16]. The proposed method allows each worker to communicate new gradients while one is calculating with only one older weights. In addition, this paper explains the speed multiplication factor using this method.

This paper is organized as follows: Section 2 presents a data-parallel SGD using delayed weight update. The experimentally obtained results, including convergence time performance, are explained in Section 3. The last section presents discussion of the proposed method and related issues.

2. DATA PARALLELISM WITH DELAYED WEIGHT UPDATE

Fig. 1 presents a conceptual diagram of data parallelism. After a batch of data (a mini-batch) is given to each worker, every worker calculates its gradient (dW_1, dW_2, …) in the order of forward propagation, loss calculation, and back propagation. The server collects the gradients from the workers and then distributes an updated weight (W) back to the workers after averaging the gradients. The gradients and the weight have the same data size, which increases with the network size; the server collects the gradients of 243 × n MB (where n is the number of workers) in the VGG-F network [17] with the ImageNet benchmark dataset [3]. Then the server calculates the averaged weight from the gradients and broadcasts the updated weight of 243 MB to the workers.

The processing time for the gradient collection, weight update, and weight distribution is regarded as a latency until the next gradient calculation for the workers. It is overhead time in the conventional synchronous data parallelism. This latency increases with increasing parameter size. Especially in a huge model, this latency hinders the scaling up of parallelization. The proposed method hides the latency using a stale (= delayed) gradient. Therefore, the proposed delayed update makes it possible for workers to perform instantaneous processes. Fig. 2 presents timing diagrams of the conventional synchronous data parallelism and the proposed method with the delayed update.

In conventional data parallelism, T_W and T_S are given respectively as a calculation time for workers and a processing time for the server. In T_W, each worker calculates its own gradient, $dW^{(t)}$ for the present batch step, t, with the updated weight, $W^{(t)}$. Then in T_S, the server collects the gradients, by which the weight is updated. The new weight, $W^{(t+1)}$, is distributed back to the workers. In other words, $W^{(t+1)}$ is updated by the very new $dW^{(t)}$, yet there must exist a latency between the gradient and the weight. Workers and the server, with the proposed data parallelism with the delayed update, carry out their tasks concurrently without latency. The updated weight is $dW^{(t)}$ as in the conventional method, but it is for $W^{(t+2)}$, not for the very next. Concurrency exists with no latency, but with a delay in batch steps. Therefore, we designate the proposed method as "delayed update." The proposed method is not a naive SGD algorithm. We must verify its accuracy, as explained below.

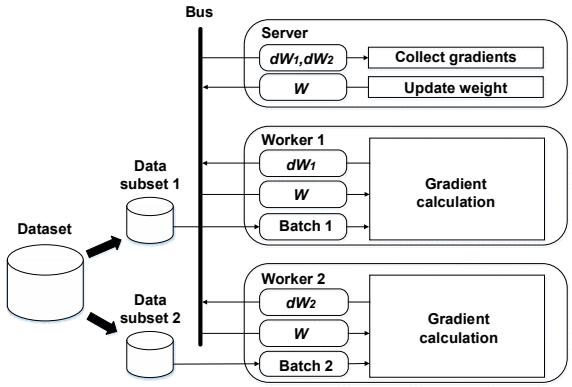

Fig. 1. Conceptual diagram of data parallelism for distributed deep learning (two workers shown for simplicity).

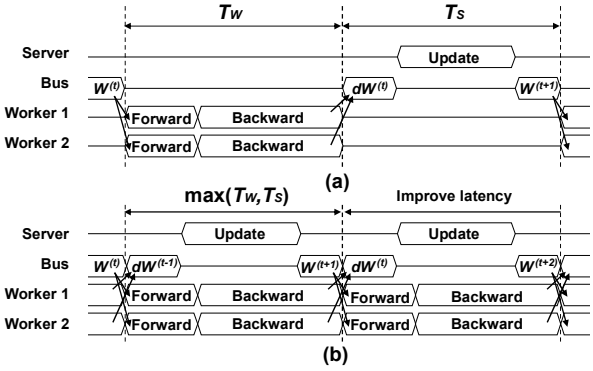

Fig. 2. Timing diagrams of (a) conventional SGD data parallelism and (b) proposed SGD data parallelism with delayed update.

3. EXPERIMENT RESULTS

3.1 Acceleration Ratio: R_{WS}

The convergence time of the proposed method depends not only on its accuracy, but also on the computing system. The conventional data parallelism takes T_W for a worker to calculate a gradient in a batch. It takes T_S for the server to process a new updated weight. $T_W + T_S$ is the time for one iteration. In the proposed method, the iteration time is the longer one of T_W and T_S. For the proposed method, we define acceleration ratio R_{WS} as

$$R_{WS} = (T_W + T_S) / \max(T_W, T_S). \quad (1)$$

Actually, R_{WS} becomes 2 at maximum when $T_W = T_S$, although R_{WS} strongly affects the final convergence speedup. In our experimental environment (Core i7-6700K, NVidia GeForce GTX 1080, Cuda 9.0, CuDNN 6.1, Matlab 2017a, and MatConvNet [18]), we verified VGG-F [17] and ResNet50 [5]. Our mechanical system contains a single GPU. In order to evaluate data parallelism, parallelization is virtually implemented by updating the weight with the accumulated gradient. Fig. 3 shows the gradient calculation times (GCTs) and the weight update time (UTs) achieved after doubling the batch size from one to 128. A GCT is T_W; the sum of a UT and a communication time is T_S. The GCT increases with the batch size. In VGG-F, the GCT becomes almost equal to the UT at a batch size of 16; T_S becomes a bottleneck for batches smaller than 16. TABLE I presents the normalized GCT per image for batch sizes. The experimental environment appears to be well optimized when the batch size is 16–128 in VGG-F.

Communication time is particularly determined by the data size of a weight. VGG-F with three fully connected layers requires as much as 243 MB of weight memory. ResNet50 has 100-MB weight memory. For this reason, the amount of data communication between the server and the worker is important for T_S. Fig. 4 depicts the measured time for transferring gradients and the update weight. The communication time increases with the number of threads. From these setup data, Fig. 5 is obtained. In VGG-F, R_{WS} tends to be a small value because the communication time is dominant over others. R_{WS} can be improved using a wider bus interface. In ResNet50, R_{WS} approaches a better value as the number of threads is increased because the GCT is much larger than the sum of the communication time and the UT. R_{WS} reaches 1.92 when the number of threads is eight and the batch size is 16.

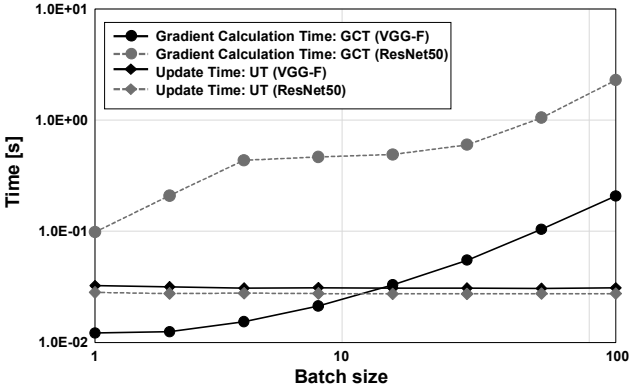

Fig. 3. Measured gradient calculation times (GCTs) and weight update times (UTs) in VGG-F and ResNet50.

TABLE I Gradient calculation time (GCT) per image

Calculation time (ms)	Batch size							
	1	2	4	8	16	32	64	128
VGG-F	12.2	6.24	3.84	2.65	2.06	1.72	1.63	1.62
ResNet-50	98.4	104.5	108.8	58.1	30.6	18.8	16.4	17.9

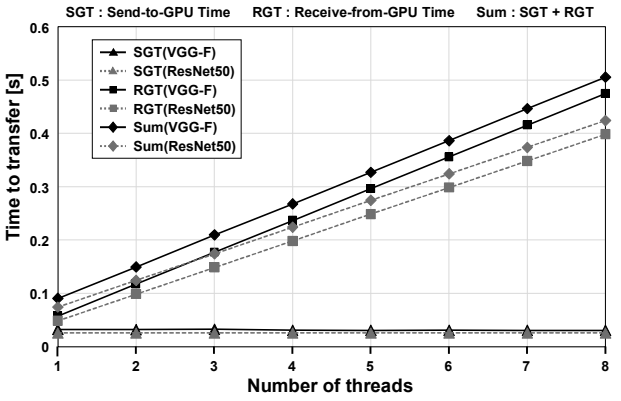

Fig. 4. Measured communication times for sending and receiving data to/from the GPU, and their sums.

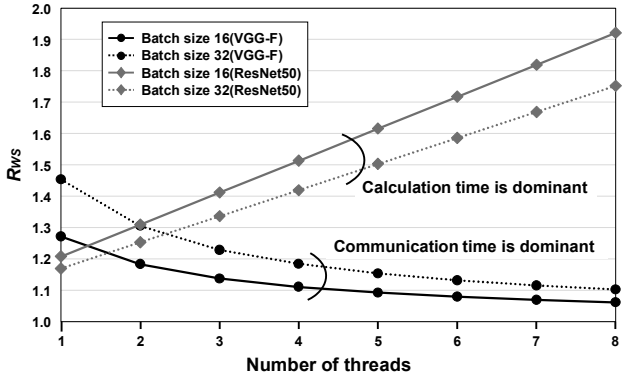

Fig. 5. Change in R_{WS} for various numbers of threads.

3.2 Convergence Speedup

3.2.1. Results obtained using the VGG model

The proposed data parallelism with the delayed update does not strictly carry out the naive SGD because the gradient is calculated using the one-step delayed weight. To verify the accuracy and convergence time, we first set optimal hyperparameters. In data parallelism, the batch size and the learning rate are important and closely related hyperparameters. The actual batch size is proportional to the number of threads. According to the linear scaling rule [6], the learning rate must be increased in proportion to the batch size. Therefore, we set the respective learning rates as 0.001 × *th* (where *th* is the number of threads and the batch size is 16). This learning rate is multiplied by 1/10 at the 20-th epoch and the 30-th epoch. The optimization method is the momentum SGD. The moment is 0.9. The weight decay is 0.0001. Batch normalization [19] is adopted for convolutional layers and fully connected layers, except the final layer. After implementing the proposed data parallelism with the delayed update, we observed its convergence time. Fig. 6 shows a convergence comparison from learning 40 epochs in training a subset of 50,000 images in the ImageNet benchmark dataset [3] (50 images

selected randomly from each category). Table II shows the final accuracy obtained under each condition. In the same *th*, the final accuracy difference between the conventional method and the proposed method is up to 1.5%. When *th* = 8, the proposed method shows slower convergence in the early epochs. In the latter part, however, it converges just as well as other condition of proposed method. It is noteworthy that the proposed method has a speedup factor: R_{WS} The proposed data parallel method can learn faster.

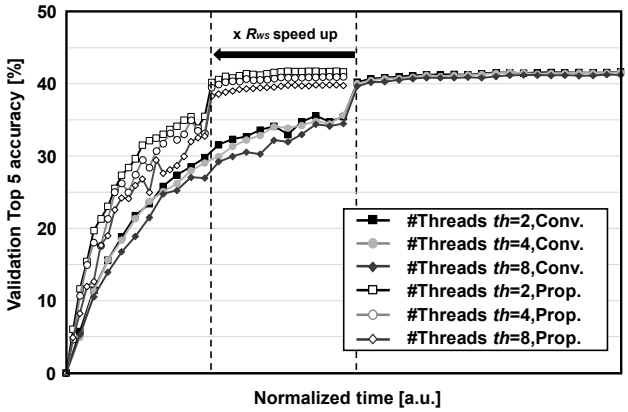

Fig. 6. Convergence comparison of VGG-F between conventional data parallelism and the proposed data parallelism when R_{WS} is 2.

TABLE II Final accuracy of the VGG-F network [%]

Condition	Number of threads		
	2	4	8
Conv.	41.7	41.6	41.2
Prop.	41.7	41.0	39.7

3.2.2. Results obtained using ResNet

Because of GPU memory constraints, we implemented our proposed method using the ResNet-18 model instead of Resnet-50 and then compared the results with those obtained using the conventional method. As with the experiment of the VGG model described above, the optimizer is Momentum SGD. The learning rate is 0.025 × *th*; it is multiplied by 1/10 at every 30 epochs. The moment is 0.9. The weight decay is 0.0001. Batch normalization is adopted for each convolution layer. The batch size is 16. Fig. 7 shows a convergence comparison between the conventional method and the proposed from learning 70 epochs. Furthermore, Table III shows the final accuracy under the different *th* conditions. Similarly to the VGG-F network, the learning curves converge on all conditions; the accuracy deterioration is 1.4% at most. These results from the VGG model and the ResNet prove that the proposed data parallelism can boost the learning speed irrespective of the models. Particularly, ResNet has fewer parameters than the VGG model because of the less fully-connected layer. Therefore, ResNet is a suitable model for data parallelism. This result is important for evaluating the proposed method.

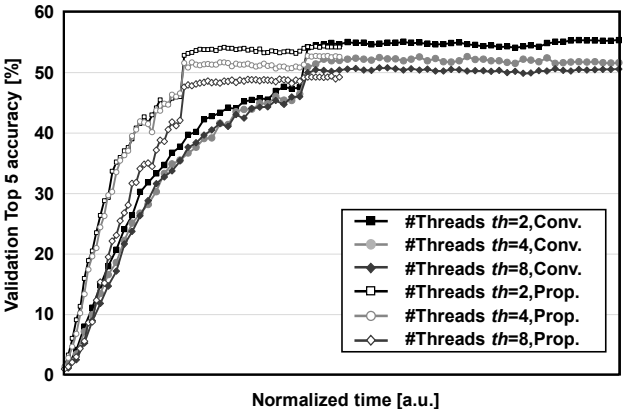

Fig. 7. Convergence comparison of ResNet-18 between conventional data parallelism and the proposed data parallelism when R_{WS} is 2.

TABLE III Final accuracy of the ResNet-18 network [%]

Condition	Number of threads		
	2	4	8
Conv.	55.4	50.6	50.5
Prop.	54.2	52.6	49.2

4. CONCLUSION AND DISCUSSION

The experiment results demonstrate that applying the delayed data parallelism to a general deep learning model improves the convergence speed, although it degrades the accuracy slightly. The proposed method shows great effectiveness when communication costs increase with the model size and the number of parallel workers. In a huge distributed deep learning system including more than 1000 GPUs, the proposed method is expected to be very effective because the communication costs are high. Moreover, the proposed method is useful for low-cost distributed systems with inexpensive communication interfaces.

Furthermore, to generalize the proposed method, we must eliminate accuracy deterioration. Deep learning accuracy is often dependent on a set of hyperparameters and a type of optimization method; they should be optimized through further experimentation. In this paper, the degree of parallelism is up to eight (*th* = 8). Algorithms such as LARS [11] and Warmup [12], which suppress accuracy degradation even when the actual batch is very large, might be countermeasures.

In the proposed method, speed up is difficult if either T_S or T_W dominates the time. When T_S is dominant, the introduction of a high-bandwidth communication interface increases R_{WS}. When T_W is dominant, introduction of a high-performance processor is a solution. Alternatively, reducing the parameter size or communicating half-precision gradients is expected to be another solution. To balance T_S and T_W, optimizing hardware for processors, memory, and communication interfaces is necessary in practical cases. If T_S and T_W are balanced, then the proposed method will help speed up data parallelism.

REFERENCES

[1] K. Fukushima, "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," Biological Cybernetics, vol. 36, no. 4, pp. 93–202, Apr. 1980.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, pp. 436–444, May 2015.

[3] ImageNet at http://image-net.org/

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Proceedings of Neural Information Processing Systems (NIPS), pp. 1097–1105, Dec. 2012.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," Proceedings of European Conference on Computer Vision (ECCV), arXiv:1603.05027, July 2016.

[6] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," arXiv:1404.5997, Apr. 2014.

[7] H. Mori, T. Youkawa, S. Izumi, et al., "A Layer-Block-Wise Pipeline for Memory and Bandwidth Reduction in Distributed Deep Learning," Proceedings of IEEE International Workshop on Machine Learning for Signal Processing (MLSP), pp. 1–4, Sep. 2017.

[8] Y. Bengio, "Practical Recommendations for Gradient-Based Training of Deep Architectures," arXiv:1206.5533, Sep. 2012.

[9] S. Gupta, W. Zhang, and F. Wang "Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study," Proceedings of IEEE International Conference on Data Mining (ICDM), arXiv:1509.04210, Dec. 2016.

[10] J. Keuper and F.-J. Pfreundt, "Distributed Training of Deep Neural Networks: Theoretical and Practical Limits of Parallel Scalability," arXiv:1609.06870, Dec. 2016.

[11] Y. You, I. Gitman, and B. Ginsburg, "Large Batch Training of Convolutional Networks," arXiv:1708.03888, Aug. 2017.

[12] P. Goyal, P. Dollr, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour ," arXiv:1706.02677, June 2017.

[13] Y. You, Z. Zhang, C. Hsieh, J. Demmel, and K. Keutzer, "ImageNet Training in Minutes," arXiv:1709.05011, Sep 2017.

[14] T. Akiba, S. Suzuki, and K. Fukuda, "Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes" arXiv:1711.04325, Nov. 2017.

[15] A. Agarwal, and J. Duchi, "Distributed delayed stochastic optimization," Proceedings of Neural Information Processing Systems (NIPS), pp. 873-881, Dec 2011.

[16] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng, "Large scale distributed deep networks," Proceedings of Neural Information Processing Systems (NIPS), pp. 1232–1240, Dec 2012.

[17] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556, Sep. 2014.

[18] MatConvNet at http://www.vlfeat.org/matconvnet/

[19] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Proceedings of International Conference on Machine Learning (ICML), pp. 448-456, July 2015.