Accelerating Training of DNN in Distributed Machine Learning System with Shared Memory

Eun-Ji Lim, Shin-Young Ahn, Wan Choi
High Performance Computing Research Group
Electronics and Telecommunications Research Institute (ETRI)
Daejeon, South Korea
{ejlim, syahn, wchoi}@etri.re.kr

Abstract— **In distributed DNN training, the speed of reading and updating model parameters greatly affects model training time. In this paper we investigate the performance of deep neural network training with parameter sharing based on shared memory for distributed machine learning. We propose a shared memory-based modification of the deep learning framework. In our framework, remote shared memory is used to maintain global shared parameters of parallel deep learning workers. Our framework can accelerate training of DNN by speeding up the parameter sharing in every training iteration in distributed model training. We evaluated our proposed framework by training the three different deep learning model. The experiment results show that our framework improves training time for deep learning models in distributed system.**

***Keywords*—Machine learning, deep learning, shared memory, distributed machine learning**

I. INTRODUCTION

In recent years, the deep neural network (DNN) models have become larger and larger and the size of the training datasets have been increasing. Training the DNN models demands more time and computations. The training of DNN models can be significantly accelerated with distributed system. The frameworks for distributing the computations across multiple machines have been recently developed [1,2,11,12]. There are two approaches to paralleling and distributing the training computation: data parallelism and mode parallelism. While model parallelism can work well in practice, data parallelism is the preferred approach for distributed systems and has been the focus of more research. There are two different approaches to parameter updating of data-parallel training: synchronous and asynchronous. Synchronous training improves the efficiency of one iteration, and asynchronous training improves the convergence rate.

Many distributed training system employs parameter server [3,4,5,6], which is responsible for the aggregation of parameter updates and parameter requests coming from different workers. It manages shared state that is updated by a set of parallel workers. In distributed training, workers simultaneously read and update the model parameters from parameter servers in every training iteration. The speed of reading and updating parameters greatly affects model training time. This effects is greater when the number of parameters is large.

Many effective trained models have a large number of parameters. For example, the current state-of-the-art image classification model, ResNet, uses 2.3 million floating-point parameters to classify images into one of 1000 categories [8]. The One Billion Word Benchmark has a vocabulary of 800,000 words, and it has been used to train language models with 1.04 billion parameters [9]. For these models with a large number of parameters, the influence of parameter reading and updating speed on training performance is great.

We propose a method of improving the speed of reading and updating parameters using remote shared memory which can be accessed across multiple workers. We will describe our shared memory-based modification of TensorFlow framework.

II. REMOTE SHARED MEMORY

In this paper, we use remote shared memory that is accessible by multiple workers simultaneously to maintain parameters of distributed deep learning model. We modified the remote memory extension system we developed in our previous study[7,10]. Memory server connected via high-speed network exports its own main memory, which is accessible through low-latency and high-speed RDMA and shared across multiple worker nodes. The remote memory region is mapped on virtual address space of application process. Figure 1 shows our remote shared memory and an example of remote memory allocation and mapping. The SM device driver allocates remote memory at the request of the application process and maps the allocated pages to the virtual address space of the process. The application can read and write data to remote memory like its own local memory because the SM library and SM device driver hides the details of accessing the remote memory. In order to map the remote memory region to virtual address space, SM device driver uses temporal memory region in the physical memory of the worker nodes. When an application process try to access the mapped remote memory region, SM device driver handles page fault. It transfers data from remote memory to the page in its temporal memory region and maps the page to the virtual

address space of process by updating the page table. The role of temporal memory region is similar to that of page cache in kernel.

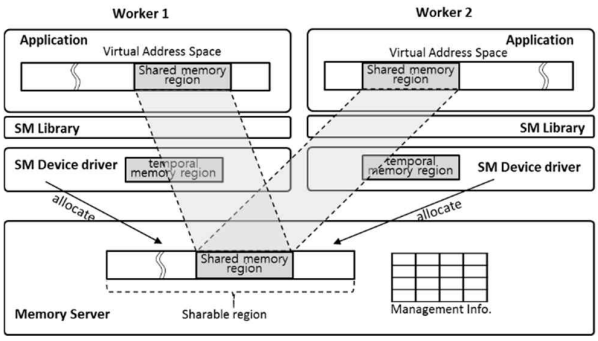

Figure 1. Remote shared memory and memory allocation

Table 1 shows APIs provided to applications to use remote shared memory. An application can allocate remote shared memory via mb_shm_alloc() and may release the memory via mb_shm_free(). Applications in multiple workers can share identical memory region by using the same key when they allocate memory. T

TABLE 1 APIS FOR REMOTE SHARED MEMORY

API	Feature
int mb_shm_alloc(key_t key, size_t size, int shmflg);	Allocate memory server's centralized memory on the process' s virtual address space Return shared memory identifier or -1 if it fails to create
void mb_shm_free(void *ptr);	Deallocate the allocated shared memory
int mb_rsync(void *ptr, size_t size);	Read from remote shared memory to local memory
int mb_wsync(void *ptr, size_t size);	Write from local memory to remote shared memory

If an application write data to allocated memory, it is applied to its local temporal memory region. Application can issue transferring data from temporal memory region to remote shared memory explicitly by using mb_wsync(). In case of read, it operates in the same way by using mb_rsync().

III. DEEP LEARNING FRAMEWORK SHARING PARAMETERS WITH SHARED MEMORY

In this paper, we propose the distributed deep learning framework that shares parameters by using the remote shared memory described in Chapter II. We modified TensorFlow, an open-source deep learning framework developed by Google. In TensorFlow, parameters are shared between workers through parameter servers in distributed training. TensorFlow uses gRPC(over TCP) protocol to move data between workers and parameter servers. In each training step, workers fetch the parameters from parameter servers via gRPC and the parameter servers receive gradients from workers via gRPC after workers compute gradients. We replace the parameter servers to remote shared memory in TensorFlow in order to accelerate distributed model training by reducing the parameter sharing overhead. In this study, our framework supports asynchronous parameter training of data-parallel approach. We will consider the synchronous training in future work.

TensorFlow distributes and executes operations that are composing a model on multiple devices. Users can specify that operations run on a particular device. TensorFlow supporting device types are CPU and GPU. TensorFlow has device objects corresponding to supported device types in its internal. We added a new device type and device object for remote shared memory. Also, we added several operation kernels that save and update parameters to shared memory. If users specify variables that hold and update parameters to be created on our new device when they build model, the parameters can be updated and read from the shared memory. We have not modified the original TensorFlow architecture and features. Users can take advantage of our framework with minimum change of existing model. Figure 2 shows overall architecture of our distributed deep learning system that shares parameters with shared memory. SMDevice object manages access to shared memory.

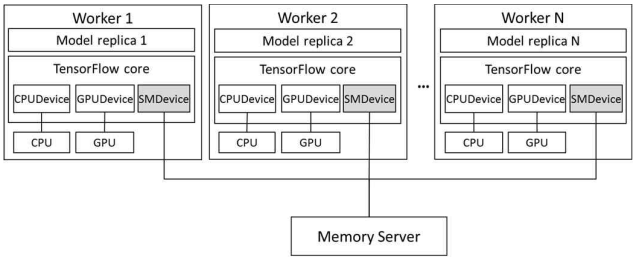

Figure 2. Overall architecture of distributed deep learning system with shared memory

As shown in Figure 3, SMAllocator in SMDevice allocates and deallocates shared memory for parameters from memory server. Newly added operation kernels for SMDevice read and write parameters from (to) allocated shared memory.

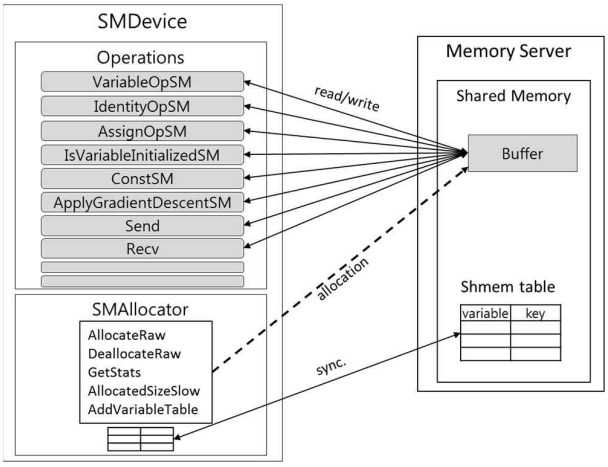

Figure 3. Allocating and accessing shared memory for parameters

In order for multiple walkers to share parameters stored in shared memory, mapping table is required to manage the memory mapping information of the parameters. Because workers should share the mapping table, the mapping table is maintained in the shared memory. Since shared memory is mapped to the virtual address space of a process, workers share and access parameters and mapping table through their respective process address spaces. After a worker updates the

parameter by writing it to shared memory, another worker can obtain the updated parameter by reading it from the shared memory. By sending and receiving parameters with a local memory access approach rather than in the form of a message communication, model training can be accelerated by reducing message processing time and communication overhead.

IV. EXTERIMENTS

We compared the performance of our framework and TensorFlow in distributed model training for image recognition. We measured the training speed of MLP(Multi-layer perceptron) model and CNN model for classifying MNIST dataset in two framework. The MNIST database is a large database of handwritten digits that is commonly used for benchmarking various deep learning algorithms. In addition, we measured the training time of a CNN model for classifying ImageNet dataset. ImageNet is a common academic data set in machine learning for training an image recognition system. We used 2 workers with 1 GPU each. They have two Intel Xeon E5-2690 v4 CPUs with 14 cores, 128GB RAM, and a NVIDIA P100 GPU. We used one parameter server and one memory server, which have 256GB RAM each. All machines are connected via Infiniband FDR (Max bandwidth 56Gbps) network.

In first experiment, we measured the training time of one batch while training MLP model and CNN model for classifying MNIST dataset until they achieve an accuracy of over 80%.

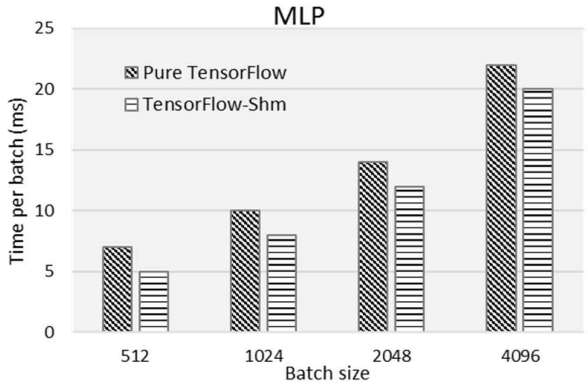

Figure 4. Training speed of MLP model for classifying MNIST dataset

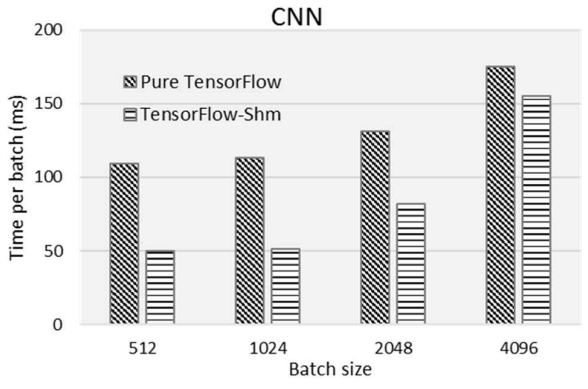

Figure 5. Training speed of CNN model for classifying MNIST dataset

As shown in Figure 4 and 5, our framework achieved about 10~50% improvement of training time in various batch size. Although reduced training time varies by model or batch size, our framework has shown better results in all cases. We expect that the reduced time was varied because the amount of computation and communication was varied in difference cases.

In final experiment, we used Google's Inception V3 model, which achieves 78.8% accuracy in ILSVRC 2012 image classification challenge. In practice, it can take days or even weeks for training this model from scratch. We measured times of executing 2000 training iterations for this model. Our test is a small snapshot from much longer runs that would be needed to train this network from scratch. Measured time is shown in Figure 6. Our framework reduces training time by 14~20% compared to TensorFlow.

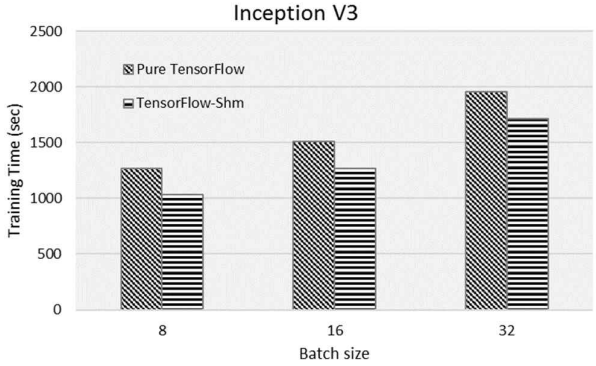

Figure 6. Training time of Inception V3 model for classifying ImageNet dataset.

The experiment results show that our framework improves training time for several DNNs in distributed environment. Sharing parameters with our shared memory have the great influence on the training time in distributed training. From the experiment result, we found that our framework is more effective, when the computation is less. It is because the less the computation, the portion of communication time in training time is larger. In other words, we can expect that if we use more powerful computation device(GPU, etc.), the impact of our framework will become greater. In addition, we can expect that the larger the number of parameters of model, the greater the advantage will be.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we proposed a shared memory-based modification of the deep learning framework in order to accelerate training of DNN in distributed environment. In our framework, remote shared memory is used to maintain global shared parameters of distributed deep learning workers. Remote shared memory can be accessed from distributed workers via low-latency and high-bandwidth RDMA, which can reduce the time of reading and updating parameters in every training iteration. This helps to accelerate training of DNN in distributed system. To evaluate our framework, we compared the performance of our framework and TensorFlow in distributed model training with three different image recognition models. The experiment results showed that our framework improves

training time for three different DNNs in distributed environment. We also found that the impact of our framework will become greater in case of that the portion of communication time in training is large and the number of parameters of a model is large.

For the future works, we plan to evaluate our framework by measuring the time of complete training of Inception-v3 models from scratch. Examining our proposed idea for other models, datasets and computing environment is a research problem worth exploring. In addition, the support for the synchronous training should be considered in future work.

ACKNOWLEDGMENT

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

REFERENCES

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467, 2016.

[2] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. "TensorFlow: A System for Large-Scale Machine Learning." In OSDI, 2016, pp. 265-283.

[3] Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J. K., Gibbons, P. B., et al. "More effective distributed ml via a stale synchronous parallel parameter server.", In Advances in neural information processing systems, 2013, pp. 1223-1231.

[4] M. Li, D. G. Andersen, J. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. "Scaling distributed machine learning with the Parameter Server.", In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 583–598, 2014.

[5] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. "Large scale distributed deep networks." In NIPS, 2012.

[6] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. "Project Adam: Building an efficient and scalable deep learning training system.", In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 571–582, 2014.

[7] Shinyoung Ahn, Gyuil Cha, Youngho Kim, Eunji Lim, "Exploring feasibility of user-level memory extension to remote node", 6th Int'l Conference on Internet, 2014.

[8] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition.", CoRR, abs/1512.03385, 2015.

[9] R. J´ozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. "Exploring the limits of language modeling.", CoRR, abs/1602.02410, 2016.

[10] Eun-Ji Lim, Young-Ho Kim, Shin-Young Ahn, Gyu-Il Cha, "Distributed Memory Integration System for High Performance Data Processing", International Conference on Future Web, 2014.

[11] Yu, D., Eversole, A., Seltzer, M., Yao, K., Huang, Z., Guenter, B., et al. "An introduction to computational networks and the computational network toolkit." Microsoft Technical Report MSR-TR-2014–112, 2014.

[12] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M. et al. "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems." arXiv preprint arXiv:1512.01274, 2015.