Multi-Client DeepIO for Large-Scale Deep Learning on HPC Systems

Yue Zhu
Florida State University
yzhu@cs.fsu.edu

Fahim Chowdhury
Florida State University
fchowdhu@cs.fsu.edu

Huansong Fu
Florida State University
fu@cs.fsu.edu

Adam Moody
Lawrence Livermore National Laboratory
moody20@llnl.gov

Kathryn Mohror
Lawrence Livermore National Laboratory
kathryn@llnl.gov

Kento Sato
Lawrence Livermore National Laboratory
sato5@llnl.gov

Weikuan Yu
Florida State University
yuw@cs.fsu.edu

1 INTRODUCTION

Deep neural networks (DNNs) have gained tremendous interest due to their potential for solving complex problems. With the growth of computation power of processors and accelerators, such as on leadership High-Performance Computing (HPC) systems, larger datasets can be trained more efficiently with DNNs. However, not many efforts have worked to improve I/O support for DNNs to match the increasing computation power. Typically, on HPC systems, a training dataset is stored on a parallel file system or node-local storage devices. However, not all HPC clusters have node-local storage (e.g., SSD), and large mini-batch sizes stress the read performance of parallel file systems since the large datasets cannot fit in file system caches. To make things worse, large mini-batches have been widely used in DNN training for speeding up training and maintaining accuracy [3–5, 7]. Thus, it is a challenge for DNNs with large mini-batches to achieve high training performance on HPC systems, particularly those with GPUs.

In prior work, we proposed DeepIO [8] to mitigate the I/O pressure when loading datasets from parallel file systems. DeepIO is designed to support the I/O behavior of TensorFlow [2] and leverages on-node memory and InfiniBand between nodes to assist the mini-batch generation. TensorFlow is a popular machine learning framework that supports splitting a larger mini-batch into multiple small pieces or fetching multiple small mini-batches simultaneously to serve different workers, which is critical for achieving high training throughput. However, a key limitation of DeepIO is that it does not support multiple training workers on a single compute node. In this work, we address this gap with the following contributions:

- Modify DeepIO framework to support multiple training clients on a single node.
- Evaluate multi-client DeepIO against state of the art in-memory file systems.
- Compare DeepIO API against TensorFlow API.
- Explore the potential of DeepIO in real DNN training.

2 DEEPIO WITH NEW FEATURES

DeepIO is a temporary in-memory storage system co-located with a distributed DNN training application. DeepIO preloads datasets from backend storage to an in-memory storage buffer and incorporates several techniques to improve mini-batch read performance,

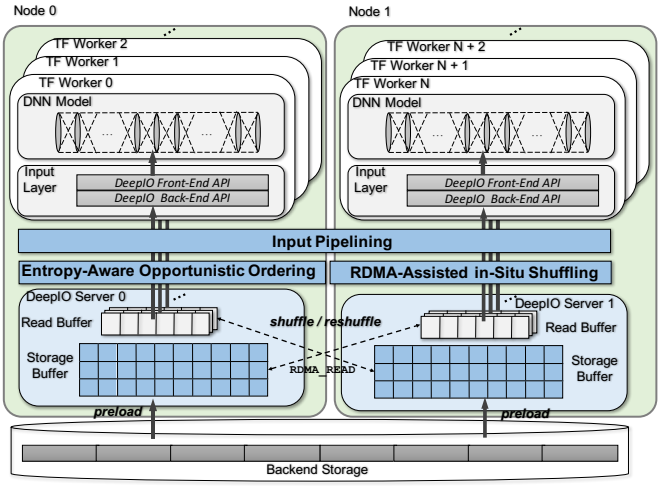


Figure 1: Data flow of reading dataset for TensorFlow.

including RDMA data transfer (*RDMA-assisted in-situ shuffling*), overlapping I/O operations with training iterations (*input pipelining*), and relaxed order of records retrieval (*entropy-aware opportunistic ordering*). DeepIO also includes a DeepIO TensorFlow API to facilitate the DeepIO's integration with TensorFlow.

In this work, to support multiple clients on a single node, we introduce multiple read buffers to distinguish the requests and elements for different training workers on top of DeepIO's old architecture (see Figure 1). Every read buffer is divided into multiple fixed length blocks which are viewed by its assigned client as a ring. Users decide the total count of read buffers, read buffer size, and block size at initialization. The read buffers are created in shared memory and exposed to RDMA for fast element read.

The original DeepIO achieved RDMA-assisted in-situ shuffling by randomly fetching elements of mini-batches from storage buffers via `memcpy` or `RDMA_read` to ensure good randomization of the input sequence and to reduce additional memory footprints in DNN training process. To support multiple clients on a single node, we modify DeepIO so that the DeepIO server on a node dispatches the randomly fetched elements to all read buffers, which are determined by users or DeepIO servers. The independent read buffers can guarantee that

the elements are returned to the correct clients without additional communication cost.

The purpose of input pipelining in DeepIO is to overlap I/O time with training iterations, both when reading datasets from in-memory read buffers or backend storages. To support multiple clients in DeepIO, we modify the in-memory pipeline of DeepIO so that the servers handle requests from all clients in a round robin manner.

DeepIO uses entropy-aware opportunistic ordering to relax the input order while still delivering proper randomization with the input pipeline. Because the training input order needs to be shuffled, we use cross-entropy to estimate the randomization level when loading datasets from backend storage. We do not need to modify DeepIO to support this feature since it is to address randomization level estimation instead of directly interacting with data movement.

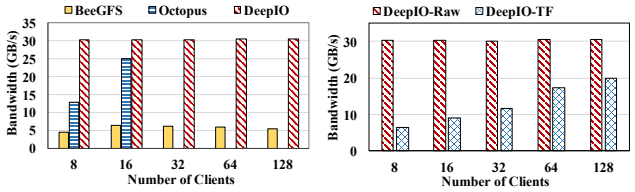
We also modify the DeepIO TensorFlow API for multi-client support, including passing the client ID and reading the individual client read buffer.

3 EXPERIMENTAL EVALUATION

To evaluate our multi-client DeepIO implementation, we measure the aggregate bandwidth when there are multiple training clients on a compute node and use a dummy dataset with random numbers to represent the dataset content for bandwidth tests. The read pattern in all read tests is fully randomized across the dataset, as fully randomized input is critical for DNN training. The results in Fig. 2(a) and Fig. 2(b) are from executions on eight nodes with a 32 GB dataset and a transfer size of 256 KB. During the read tests, we vary the number of clients from 8 to 128, which means 1 to 16 clients per node. In addition, we also explore the potential of DeepIO in real DNN training as shown in Fig. 3.

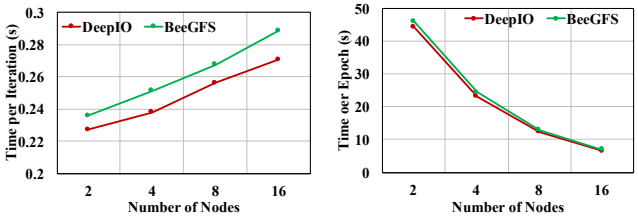
Fig. 2(a) shows the aggregate bandwidth of running different client counts over BeeGFS[1], Octopus[6], and DeepIO. BeeGFS is a traditional parallel file system mounted over tmpfs. Octopus is a state-of-the-art RDMA based memory file system. As BeeGFS delivers relatively low read performance on the fully randomized read, we use the N-to-N sequential read pattern to collect BeeGFS's read bandwidth as shown in Fig. 2(a). Besides, some results are missing for Octopus as we only manage to run Octopus for up to two clients per node without errors. As shown in Fig. 2(a), DeepIO outperforms BeeGFS and Octopus by at least 5.1× and 1.2×, respectively. We also observe that the performance of DeepIO does not increase with the client count, as one DeepIO client per node already saturates the network bandwidth. However, when using DeepIO through our DeepIO Tensorflow API (*DeepIO-TF*), we can see that there is a significant improvement with the larger number of clients, as shown in Fig. 2(b). This counters the non-trivial overheads introduced by the API.

Fig. 3 shows the average time of an epoch and an iteration when training CIFAR10 with CifarNet. During the test, we train CifarNet with DeepIO TensorFlow API over memory-based DeepIO and also BeeGFS over different node counts. As the size of CIFAR10 is relatively small, the BeeGFS tests benefit from the cache. Therefore, the performance difference between DeepIO and BeeGFS is not apparent. We take the test of varying the training dataset size for exploring the capability of DeepIO as our future work.



(a) BeeGFS vs. Octopus vs. DeepIO. (b) Alternative data APIs.

Figure 2: Aggregate read bandwidth of multiple clients.



(a) Time per Iteration. (b) Time per epoch.

Figure 3: Average time of training with CIRFAR10 (batch_size=128).

4 CONCLUSION AND FUTURE WORK

In this study, we enable the multi-client support for DeepIO, an I/O solution for DNN training on the HPC system. Our implementation shows that multi-client support is still crucial for improving I/O bandwidth when training with TensorFlow.

In the future, we plan to evaluate DeepIO with more DNNs and different sizes of datasets, to further scrutinize DeepIO's advantages and disadvantages.

Acknowledgment

This work is performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-ABS-755590). This work is also supported in part by the National Science Foundation awards 1561041, 1564647, and 1744336.

REFERENCES

[1] BeeGFS. https://www.beegfs.io/content/.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint arXiv:1603.04467*, 2016.

[3] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training Imagenet in 1 Hour. *arXiv preprint arXiv:1706.02677*, 2017.

[4] Alex Krizhevsky. One Weird Trick for Parallelizing Convolutional Neural Networks. *arXiv preprint arXiv:1404.5997*, 2014.

[5] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient Mini-batch Training for Stochastic Optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

[6] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. Octopus: an RDMA-enabled Distributed Persistent Memory File System. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 773–785, 2017.

[7] Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32k for Imagenet Training. *arXiv preprint arXiv:1708.03888*, 2017.

[8] Yue Zhu, Fahim Chowdhury, Huansong Fu, Adam Moody, Kathryn Mohror, Kato Sato, and Weikuan Yu. Entropy-Aware I/O Pipelining for Large-Scale Deep Learning on HPC Systems. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2018.