Initial Characterization of I/O in Large-Scale Deep Learning Applications

Fahim Chowdhury[†] Jialin Liu[‡] Quincey Koziol[‡] Thorsten Kurth[‡] Steven Farrell[‡]
Suren Byna[‡] Prabhat[‡] Weikuan Yu[†]
[†]Florida State University [‡]Lawrence Berkeley National Laboratory
{fchowdhu, yuw}@cs.fsu.edu {jalnliu, koziol, tkurth, sfarrell, sbyna, prabhat}@lbl.gov

I. Introduction

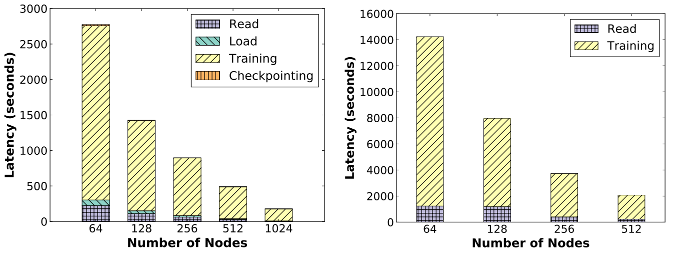
In recent times, the emergence of Deep Learning (DL) is reflected by its use in various recent applications to solve critical real-life problems in the industry and research world. For example, High Energy Physics Deep Learning Convolutional Neural Network Benchmark (HEPCNNB) [3] and Climate Data Benchmark (CDB) [1] developed and maintained by the National Energy Research Scientific Computing Center (NERSC) are some of the important applications for the research on high-energy particle physics and atmospheric science respectively. All of these DL applications are both compute- and data-intensive, i.e., demand the use of large-scale computing facilities for fast execution and high accuracy. Particularly, these large-scale DL applications require efficient I/O support in their data processing pipeline. The exploration and characterization of these I/O patterns are the indispensable prerequisites to developing optimized and efficient I/O modules. The goal of this work is to examine and extract interesting I/O patterns from multiple state-of-the-art DL applications running on HPC systems at NERSC and develop optimization strategies to overcome the possible I/O bottlenecks. So far, we have collected and organized I/O specific information of *HEPCNNB* and *CDB* that provide us with the knowledge to carry the research forward.

II. Profiling Approaches

We initiate a deep dive into the details of I/O requested by two important DL application benchmarks, i.e. *HEPCNNB* and *CDB*, developed at NERSC using TensorFlow [4] and Horovod [6]. *HEPCNNB* is used to generate particle events from a 496 GB dataset of 2048 HDF5 files, and *CDB* is developed to detect extreme weather condition patterns from 3.5 TB dataset of 62738 HDF5 image files. We build a tool named *TimeLogger* to perform I/O focused profiling of the internal executions of these DL benchmarks. We log the start and end time of basic training phase components, i.e. `Read`, `Load`, `Training` and `Checkpointing`. Later, we merge common time intervals per component and determine latency from the merged interval set. We deduce the bandwidth through dividing the size of fetched data by read latency. The results from this instrumentation help to comprehend the impact of global shuffling and dataset size on the I/O of DL applications. In addition, the results inspire to dive deeper into more precise profiling for better perception of the I/O patterns in applications like *CDB* that has high parallelism via multi-threading in TensorFlow and multi-processing in Python. Hence, we explore the TensorFlow *Timeline* module and TensorFlow Runtime Tracing Metadata Visualization (TRTMV) [5] to extract the metadata generated by internal executions of TensorFlow and combine the data with the results generated by *TimeLogger*. We are working on this integration of metadata collected from application and framework layer to understand the possible overlapping of I/O and training, and profile the complicated I/O pipeline more accurately. The current status of the work including all the related documents is available on NERSC github repository [2].

III. Initial Results

We leverage the *TimeLogger* tool and auxiliary utility modules to generate the time breakdown of training phase in both *HEPCNNB* and *CDB*. We measure the latency of each component and I/O bandwidth by scale-out tests on Cori, the Cray XC40 supercomputer at NERSC. We utilize the nodes with Intel Xeon Phi Knight's Landing (KNL) processors and keep the dataset on Lustre mountpoint with stripe size 1 MB and stripe count 1.



(a) *HEPCNNB* Five Epochs (b) *CDB* Three Epochs

Fig. 1: Scale-out Latency

As depicted in Fig.1(a), in case of five epochs experiment on *HEPCNNB*, the read latency is 8.01%, 7.72%, 6.83%, 6.16% and 1.49% of training time for 64, 128, 256, 512 and 1024 nodes respectively. Before we implemented global shuffling in *HEPCNNB*, the read time used to be 3.60%, 3.08%, 3.17%, 2.91% and 1.44% of training time for 64, 128, 256, 512 and 1024 nodes respectively. Hence, if adding global shuffling can increase the I/O percentage by almost two times for only five epochs and a small dataset, we can deduce that, it will degrade I/O performance even more for real-life scenario where there can be thousands of epochs with larger datasets. According to Fig.1(b), for three epochs experiment on *CDB*, the read latency is 8.73%, 15.05%, 10.63% and 11.04% of the training time for 64, 128, 256 and 512 nodes respectively. The larger dataset size is the reason behind this higher percentage of I/O latency compared to *HEPCCNB*, even though *CDB* takes more time for training due to its much denser neural network.

IV. Future Work

With a view to exposing the information on the possible overlap of I/O and training in applications like *CDB*, we are working on integrating the results from *TimeLogger* and *TRTMV*. Besides, we plan to optimize *HEPCNNB* by incorporating prefetching technique in the data pipeline. In the long run, we aim to design and incorporate an optimized cross-framework I/O strategy to overcome the possible I/O bottlenecks, and eventually speed-up DL applications by leveraging HPC infrastructure.

Acknowledgment This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

References

[1] Climate Data Benchmark. https://github.com/azrael417/ClimDeepLearn.
[2] DL-Parallel-IO. https://github.com/NERSC/DL-Parallel-IO.
[3] High Energy Physics Convolutional Neural Network Benchmark. https://github.com/NERSC/hep_cnn_benchmark.
[4] M. Abadi, A. Agarwal, P. Barham, and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.
[5] S. H. Hashemi. Tensorflow runtime tracing metadata visualization. https://github.com/xldrx/tensorflow-runtime-metadata-visualization, 2018.
[6] A. Sergeev and M. D. Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.