Understanding I/O behavior of Scientific Deep Learning Applications in HPC system

Hariharan Devarajan Huihuo Zheng Xian-He Sun Venkatram Vishwanath
hdevarajan@hawk.iit.edu, huihuo.zheng@anl.gov, sun@iit.edu, and venkatv@alcf.anl.gov

Argonne NATIONAL LABORATORY

ILLINOIS INSTITUTE OF TECHNOLOGY

ABSTRACT

Deep learning has been shown as a successful machine learning method for a variety of tasks, and its popularity results in numerous open-source deep learning software tools. This had led to its application in scientific domains such as cosmology, particle physics, computer vision, fusion, and astrophysics. Scientists have performed a great deal of work to optimize their computational performance. However, the same cannot be said for I/O performance. As deep learning algorithms rely on big-data volume and variety to effectively train the underlying neural network accurately, I/O is a significant bottleneck on large scale distributed deep learning applications. In this study, we aim to unravel a deep understanding of the I/O behavior of various scientific deep learning applications. We present our findings on scientific deep learning workloads running on a production super computer.

Poster QR

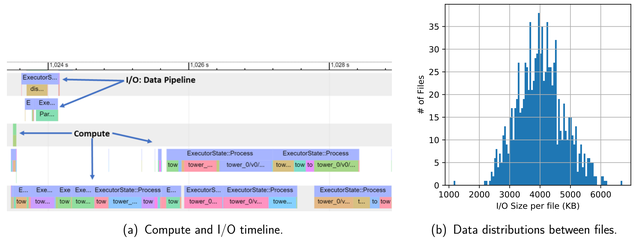
VaniDL QR

METHODOLOGY

1. We explore a collection of scientific deep learning workloads which are currently running at Argonne Leadership Computing Facility (ALCF).
2. We ran the scientific deep learning applications on the Theta supercomputer's KNL nodes with data on lustre filesystem of strip size of 1 MB and stripe count of 48.
3. We use 128 Intel KNL compute nodes, each with 192 GB of main memory, a 128 GB SSD, and 16 GB of MCDRAM in cache mode.
4. We run the tensorflow framework on CPUs with 2 hyperthreads available for a total of 64 threads.
5. We run with 128 node configuration with each node containing 4 processes (total of 512 processes).
6. We configure tensorflow to run with 32 OpenMP threads with 32 intra-threads and 2 inter-threads.
7. We utilize Darshan (with extended tracing) and tensorboard as our profiler to capture I/O behavior of the applications.
8. We developed an in-house analyzer tool, called VaniDL (https://github.com/hariharan-devarajan/vanidl), which merges the low-level Darshan logs with high-level tensorboard logs to generate a more holistic view of the application.
9. We utilize VaniDL software to create summarization, request distribution histograms, and I/O timelines at various granularity to have an in-depth view of the application's I/O behavior.

Imagenet Benchmark (TFRecord)

1. It is an image classification benchmark which contains implementations of several popular convolutional models such as resnet50, inception3, vgg16, and alexnet.
2. Dataset contains 1024 tfrecord files which each file containing 1024 sample image (approx 256 KB in size).
3. The benchmark was configured to use a batch size of 512 images and it ran for 100 steps with ten warm-up steps.



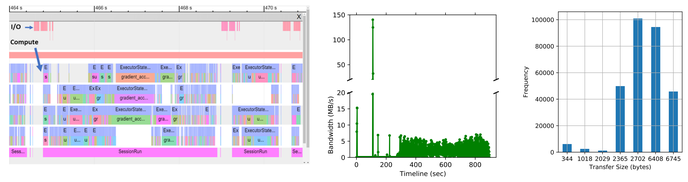
(a) Compute and I/O timeline. (b) Data distributions between files.

Figure 1: Imagenet I/O Characteristics

1. 14.3% of the total time was spent on performing I/O with a bandwidth of 32 GBps.
2. The data pipline (i.e., I/O plus preprocessing) takes 28% of the overall time.
3. The files are read on-demand during the training, and hence the samples per file access are not uniformly distributed among the files.
4. The files are accessed sequentially within the file with most the I/O is overlapped with compute.
5. Data Transfer size for each request is 256 KB which is the default in TFDataset APIs.

Neutrino and Cosmic Tagging with UNet (HDF5)

1. It uses a convolutional network to separate cosmic pixels, background pixels, and neutrino pixels in a neutrinos dataset.
2. We configured the application to run for 150 steps with checkpointing every 50 steps.
3. It reads the HDF5 files using larcv APIs with a batch size of 32 images. The application is run with 12 initial filters with the *adam* optimizer.



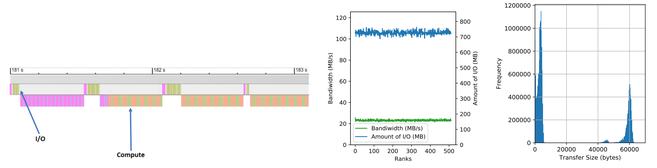
(a) Compute and I/O timeline. (b) I/O Bandwidth Timeline. (c) Data transfer size distribution.

Figure 2: Comsic Tagger I/O Characteristics

1. 22% of time (i.e., 227.24 seconds) was spent in I/O with a bandwidth of 10.43 MB/s.
2. The application reads data sequentially.
3. 22.54 seconds of I/O is overlapped with compute during the application lifetime.
4. I/O occurs throughput the execution of the application before each step executes.
5. The dataset contains images of sizes 40KB - 50KB. However, due to chunking in the dataset, we observe a smaller I/O of 2 KB and 6 KB transfer size per request.

Distributed Flood Filling Networks (HDF5)

1. It is a synchronous instance segmentation of complex and large shapes, particularly in volume EM datasets of brain tissue.
2. The application trains over an HDF5 file using 400 steps with a small batch size of 1 image.
3. It uses a field of view of 32X32X32 and a learning rate of 0.001.



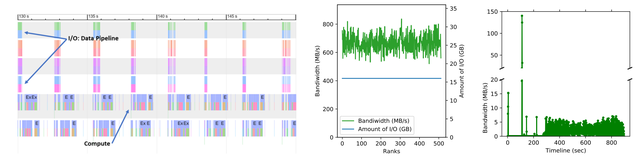
(a) Compute and I/O timeline. (b) Data and Bandwidth distribution across ranks. (c) Distribution of Transfer sizes.

Figure 3: FFN I/O Characteristics

1. 18.4% of the overall time (370 seconds) is spent on I/O with a bandwidth of 10 GBps.
2. The I/O happens just before the compute where each process reads 33% of the dataset with no overlap between I/O and compute.
3. The distribution of the transfer size centers at two places, one at about 62 KB (for reading images), the other at about 100B (for reading metadata). A chunk size of 65KB is used in the data file. That is why the transfer size for reading images is about 62KB.
4. The data access pattern is random to due random selection of image and then sequentially reading of images.

CosmoFlow Benchmark (TFRecord)

1. It is an implementation of the CosmoFlow 3D convolutional neural network.
2. The benchmark runs with 4 epochs with 1024 training files with batch size of one.



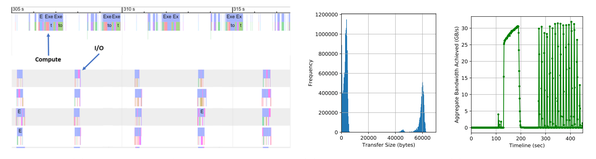
(a) Compute and I/O timeline. (b) Data and Bandwidth distribution across ranks. (c) Distribution of Transfer sizes.

Figure 4: Cosmoflow I/O Characteristics

1. 12% of the overall time (370 seconds) is spent on I/O with a bandwidth of 36 GBps.
2. 23% of the the time is spent on data pipeline (i.e., I/O + preprocessing).
3. 50.16 seconds of the I/O (almost 98%) is hidden behind compute.
4. Data Transfer size for each request on file system is 256 KB which is the default in TFDataset APIs.

Fusion Recurrent Neural Net (NPZ)

1. It implements DL models for disruption prediction in tokamak fusion plasmas.
2. Application uses 1M samples with 100 estimates in a random forest model.



(a) Compute and I/O timeline. (b) Data and Bandwidth distribution across ranks. (c) Distribution of Transfer sizes.

Figure 5: FRNN I/O Characteristics

1. 42% of its total time in performing I/O with a bandwidth of 29 GBps.
2. The signal files are read before and if required during the training with no overlap with compute.
3. Lot of calls happens on the signals metadata file which are few 100 bytes and big I/O happen on signals data files (2MB) size. dataset is accessed sequentially within the file. The files are accessed in random.

CONCLUSIONS

1. We observe the *tf.data* pipeline enables efficient I/O access in HPC environment.
2. However, scientific apps which utilize data formats such as HDF5 and NPZ are not optimized.
3. HDF5 library is not optimized to enable efficient I/O access for Deep Learning workloads and doesn't hide I/O cost behind computations of the applications.
4. As future work, we envision to build a library which can optimize HDF5 workloads for Deep Learning applications.