Analyzing Scalable Data Pipeline in Distributed Deep Learning

Fahim Chowdhury[1], Jialin Liu[2], Quincey Koziol[2], Thorsten Kurth[2], Steven Farrell[2], Suren Byna[3], and Prabhat[2]

[1] Department of Computer Science, Florida State University, Tallahassee, FL

[2] NERSC, Lawrence Berkeley National Laboratory, Berkeley, CA

[3] Scientific Data Management Research Group, CRD, Lawrence Berkeley National Laboratory, Berkeley, CA

Overview

- Deep Learning (DL) is applied to solve problems in the industry. e.g. Google's *AlphaGo*, Tesla's *Autopilot*, Baidu's *DeepSpeech* etc.
- DL techniques are rigorously applied in the analysis and research on different fields of science such as High Energy Physics, Atmospheric Science etc.
- DL applications need to train themselves by using a large amount of data to gain the best accuracy and require Supercomputing systems to run in an effective manner.
- Scalable DL requires efficient I/O mechanism.
- While Parallel I/O has been studied extensively for conventional HPC workloads, serious I/O bottlenecks are present in modern DL frameworks.
- The goal of this project is to explore I/O patterns invoked through multiple DL applications running on HPC systems and develop optimization strategies to overcome the I/O bottlenecks.

Benchmarks at NERSC

- **High Energy Physics Deep Learning Convolutional Neural Network Benchmark (HEPCNNB)**
- Uses a dataset of particle collisions generated by a fast *Monte-Carlo* generator named *Delphes*
- Used to generate particle events that can be described by standard model physics and particle events with R-parity violating Supersymmetry
- Can be expanded for multi-class classification or including regression on model parameters
- Uses *Horovod* for Distributed *TensorFlow*

- **Climate Data Benchmark**
- Uses a huge dataset of climate data images
- Used as a image recognition model
- Can be used to detect patterns for extreme weather
- Uses *Horovod* for Distributed *TensorFlow*
- Uses *TensorFlow Dataset API* and python's *multiprocessing* package for input pipelining
- Made of *Tiramisu,* a fully convolutional network for semantic segmentation

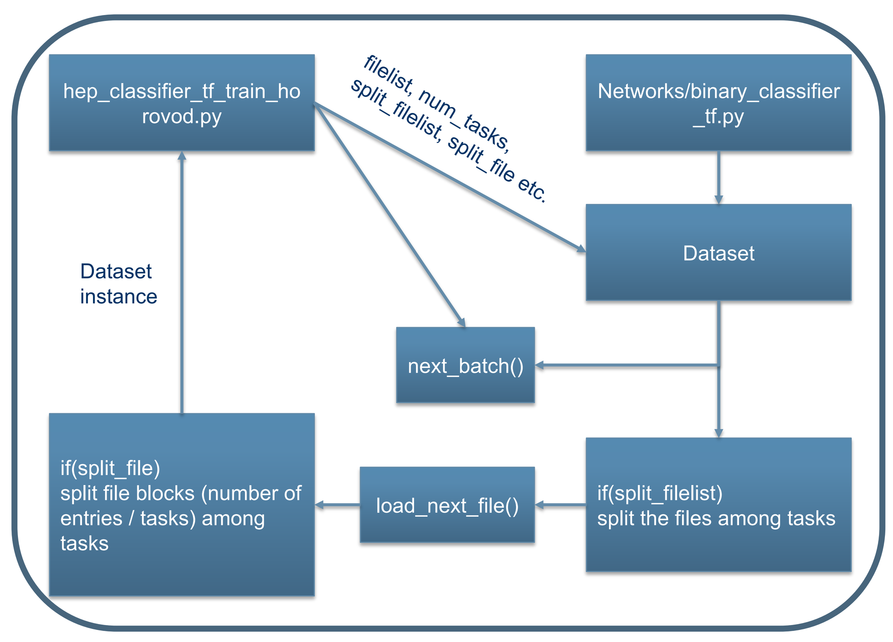
Training Datasets

- **HEPCNNB Dataset**
- 2048 files in HDF5 format
- 1024 training files 408 MiB each
- 1024 validation files 54 MiB each
- *data* – 709 × 3 × 224 × 224 32-bit integers
- *label* – 709 32-bit integers
- *normweight* – 709 32-bit integers
- *psr* – 709 32-bit integers
- Lustre: Stripe Size = 1 MB, Stripe Count = 1 */global/cscratch1/sd/ftc/deep_learning_data/hep_cnn/224x224/*

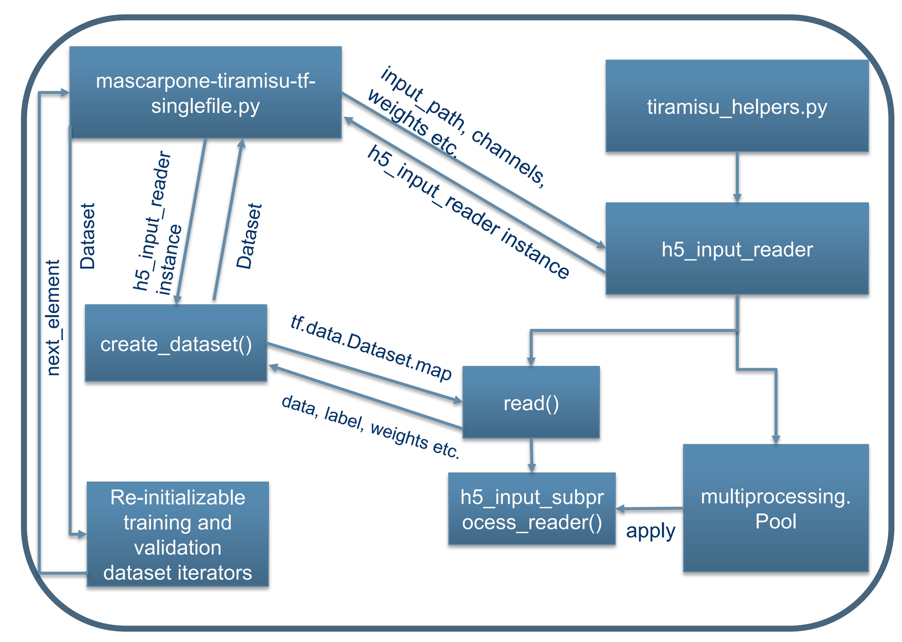
- **Climate Dataset**
- 62738 files of size 3.5 TB in HDF5 format
- Each data file is 58 MiB
- Training files: first 80% = 50190 files
- Validation files: last 10% = 6273 files
- *data* – 16 × 768 × 1152 32-bit integers
- *labels* – 768 × 1152 32-bit integers
- *stats* – 16 × 4 32-bit integers
- Lustre: Stripe Size = 1 MB, Stripe Count = 1 */global/cscratch1/sd/ftc/deep_learning_data/climate_deep_learn/*

Data Pipelines in Benchmarks

HEPCNNB Input Pipeline

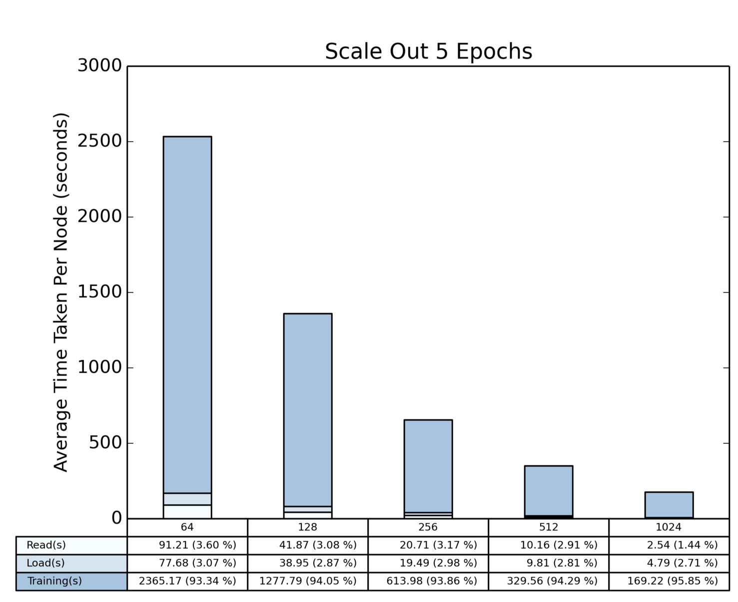


Climate Data Benchmark Input Pipeline

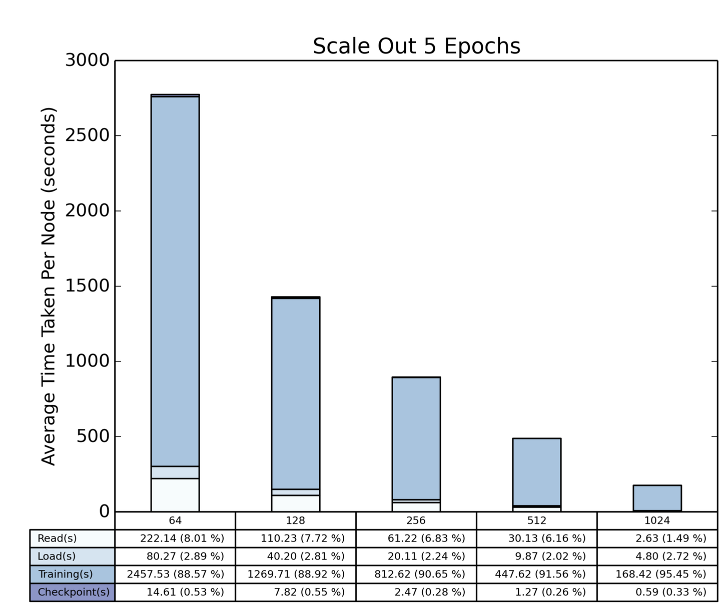


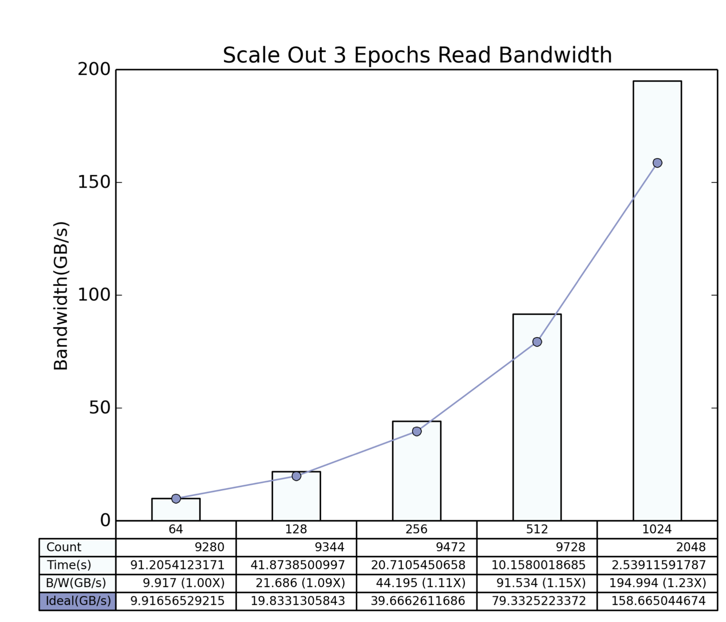
I/O Analysis of High Energy Physics Benchmark

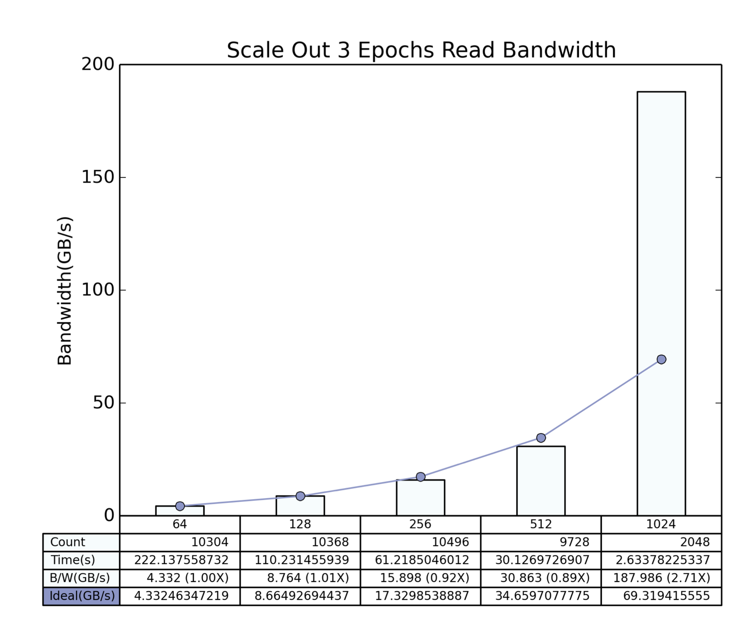
Local Shuffle



Global Shuffle

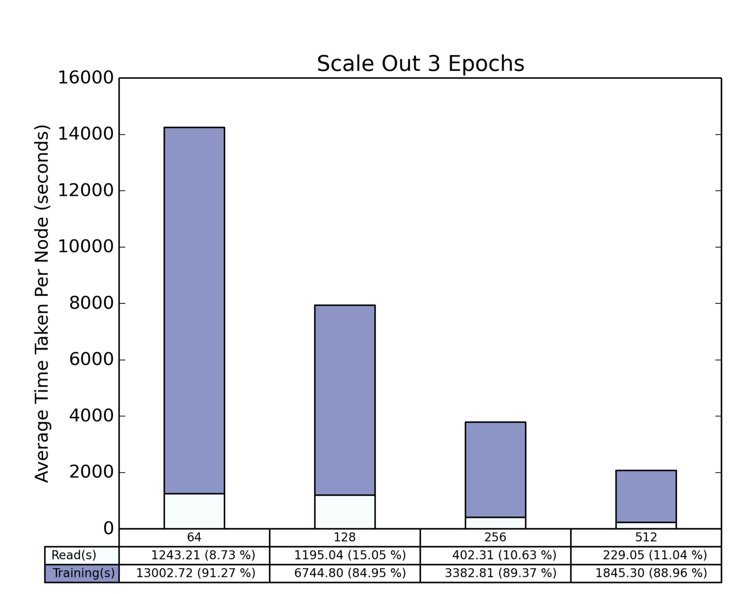


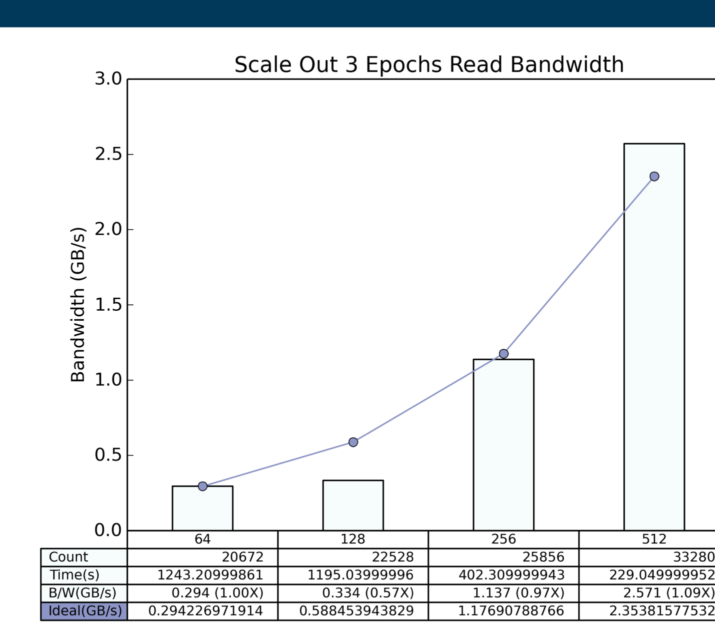




- I/O takes more time when Global Shuffling is introduced
- I/O bottleneck becomes severe with increasing number of epochs
- Global Shuffling affects I/O even if the dataset is small

I/O Analysis of Climate Data Benchmark





- The percentage of I/O in the training process is more when dataset is larger
- The I/O percentage increases with the number of nodes
- Training benefits more from the scaling than I/O

Technology

HOROVOD

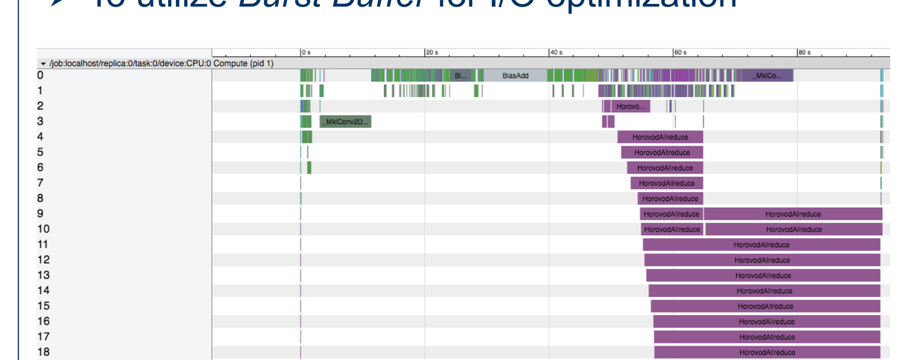
python™

Tools and Instrumentation

- *Darshan* can not profile I/O calls made without directly using *MPI* or *POSIX*
  - The benchmarks use *h5py* on *HDF5-IO*
- Developed an in-house python profiling tool, i.e. *TimeLogger* class in *utility_classes.time_logger* package
  - *TimeLogger* can log node number, epoch number, start time, end time, duration, process id and current thread of any code snippet
- Incorporated *TimeLogger* in the benchmarks
- Ran the benchmarks on *Cori*
- Developed utility scripts to extract *TimeLogger* outputs and plot using *matplotlib* python package
- Explored TensorFlow's *Timeline*
- Looking into using *TensorBoard*

Future Scopes

- To leverage TensorFlow *Timeline* for more detail and accurate profiling
- To explore *TensorBoard* for generating session graphs
- To try TAU for I/O profiling
- To profile I/O and training executions per thread
- To add optimization techniques like prefetching in HEPCNNB
- To utilize *Burst Buffer* for I/O optimization



Conclusion

- DL applications are indispensable for science and industry
- Distributed DL at scale on Supercomputing clusters needs careful analysis of I/O
- Performing careful profiling of I/O is necessary to innovate optimization techniques
- An optimized cross-framework I/O strategy is necessary to speed up training process

Acknowledgement

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.

Github link: https://github.com/NERSC/DL-Parallel-IO

Contact Information

Email: ftc@lbl.gov; fchowdhu@cs.fsu.edu;

Phone: 786-406-2617

Website: https://fahimcsebuet.github.io/website/