Analyzing the I/O patterns of Deep Learning Applications

Sandra Mendez, PhD.

Researcher, BSC.

External Researcher, HPC4EAS Research Group, UAB.

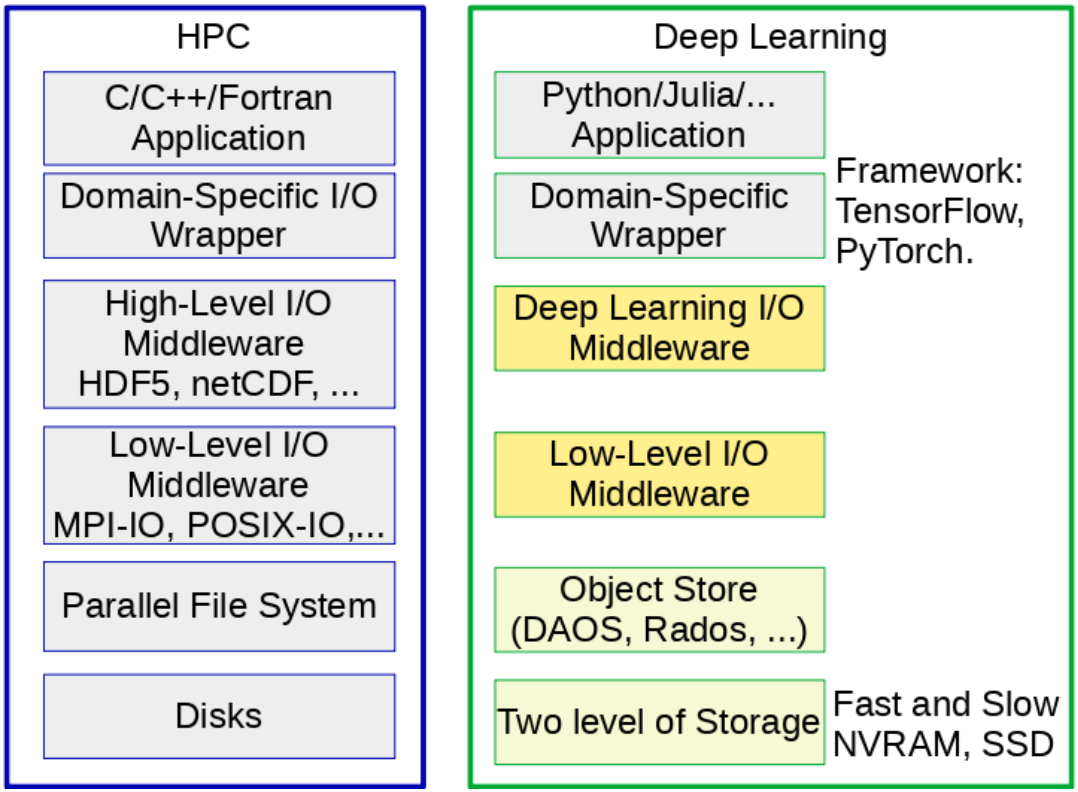
Outline

Introduction

Context

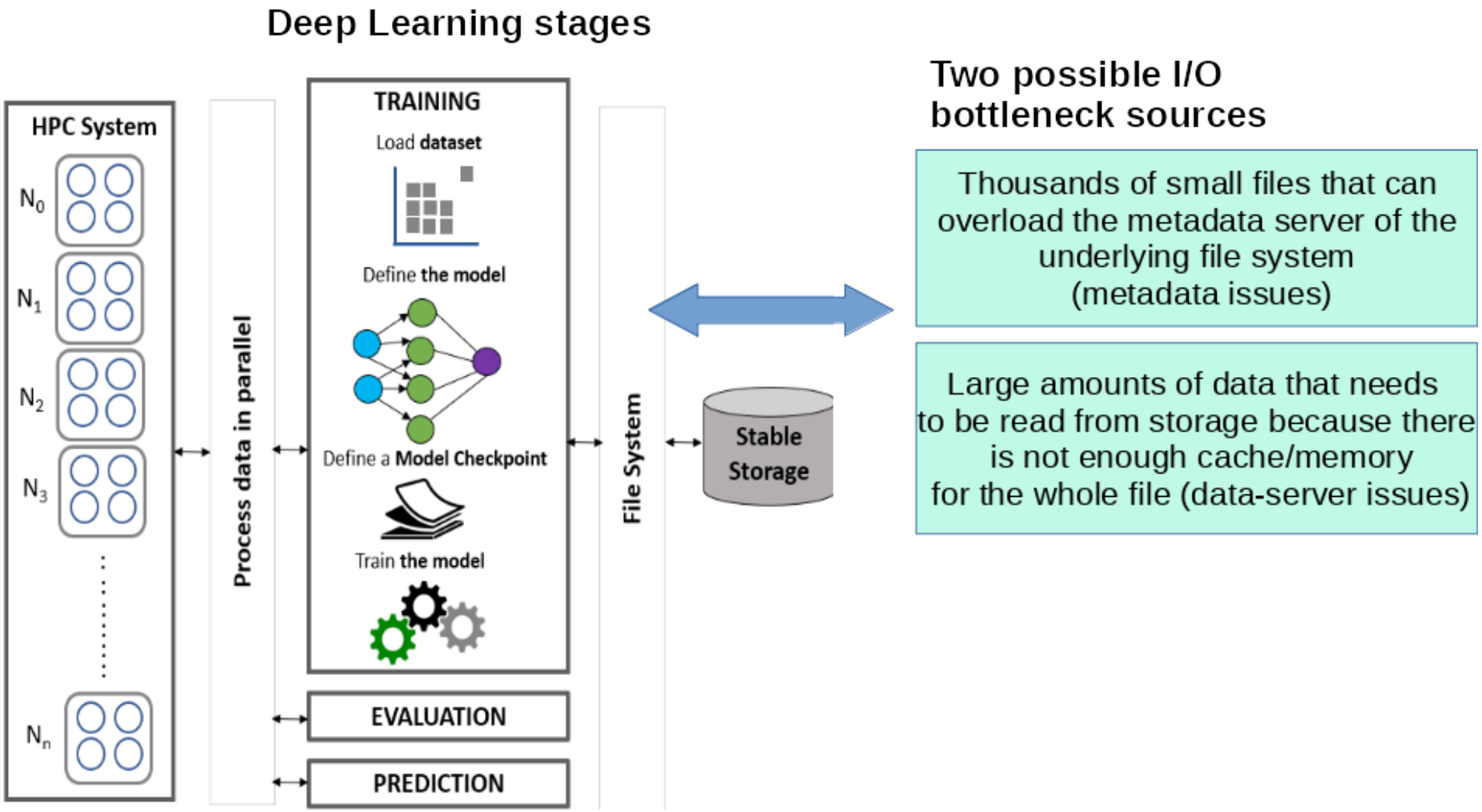
- Deep learning applications are more common in HPC systems.
- The ML/DL applications are usually characterized by read-intensive from a larger number of small files during the training phase.
- For large scaling ML/DL modeling, the checkpointing techniques are needed to be able to use a pre-trained model.
- Unlike the traditional HPC I/O subsystem, the I/O software stack differs from the classical software and tools for scientific applications.
- The evaluation of the I/O performance requires to analyze the DL I/O patterns to understand their I/O behavior on the underlying HPC I/O systems.

I/O Software Stack



The deep Learning I/O software stack is still unclear at I/O middleware level. Although DL frameworks provide different file data formats and methods for dataset manipulation, there is still not a common middleware at high and low level for parallel I/O in DL.

Deep Learning Stage



Methodology of analysis

I/O Characterization of the DL Application

- Identifying data files to trace
- Tracing selected data files

Mainly filtering file related with the application from the file open by python and frameworks. Darshan Tool allows to configure files to traces. We must take care with the maximum number to trace per process (1024 as default)

Analyzing the I/O traces

- Dataset
- Checkpoint

With focus only the Training stage of Deep Learning applications the I/O file are related with datasets and checkpoint (if it is enabled)

Description of the I/O behavior

- Metadata
- I/O Pattern

Mainly focusing on the number of files, amount of data moved on the file system, access mode and temporal and spatial pattern.

Experimental Environment

Environment

- Compute nodes - Processor Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, 24 CORES, 512 GB of RAM.
- GPU: NVIDIA A100 GPUs
- I/O system: GPFS
- Software: Horovod, Tensorflow 2.0, Keras, Python 3.7.

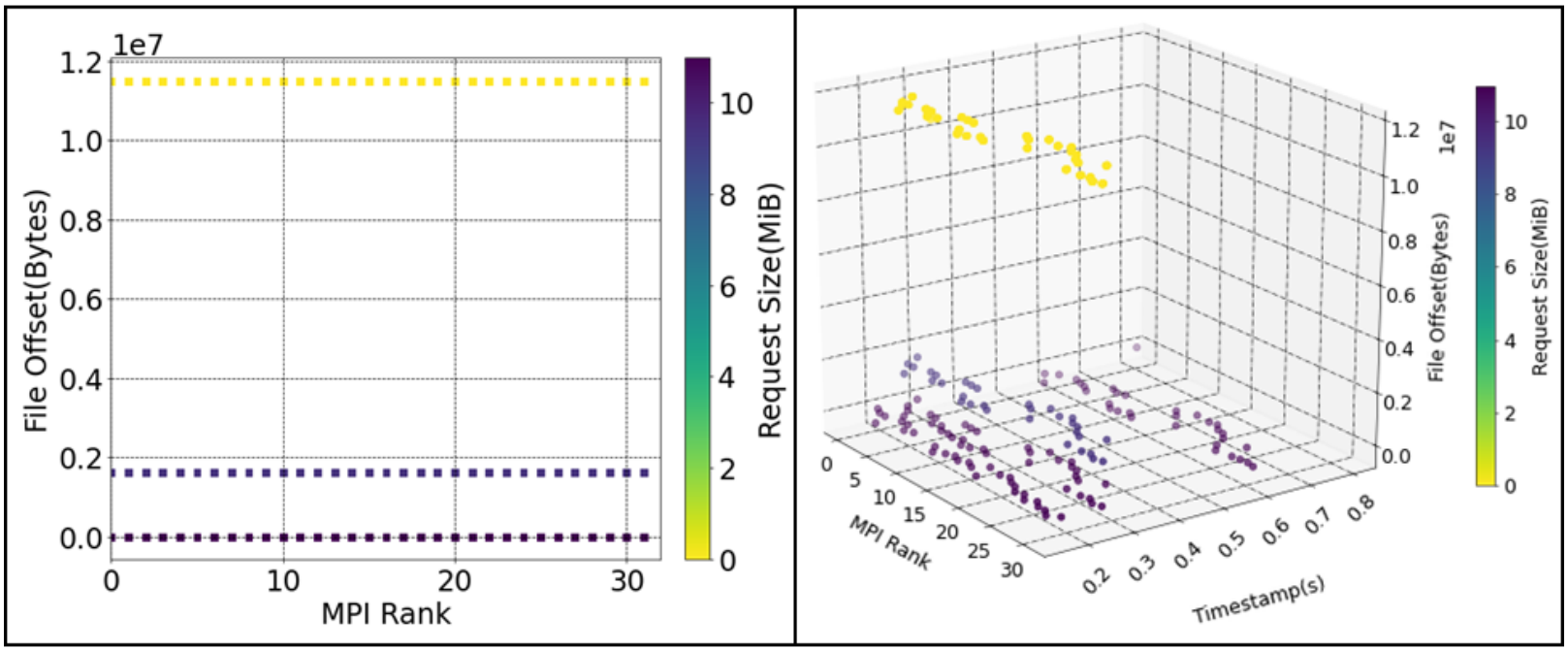
Application

As distributed DL, we select Horovod, which is a distributed deep learning training framework for TensorFlow, Keras, PyTorch, and Apache MXNet. We run horovod with the dataset MNIST.

Tensorlow MNIST (1)

`tensorflow2_mnist.py`

Run for 32 processes in eight nodes. The I/O strategy observed is one file per process, in this case the dataset file is needed for each parallel process.



Tensorlow MNIST (2)

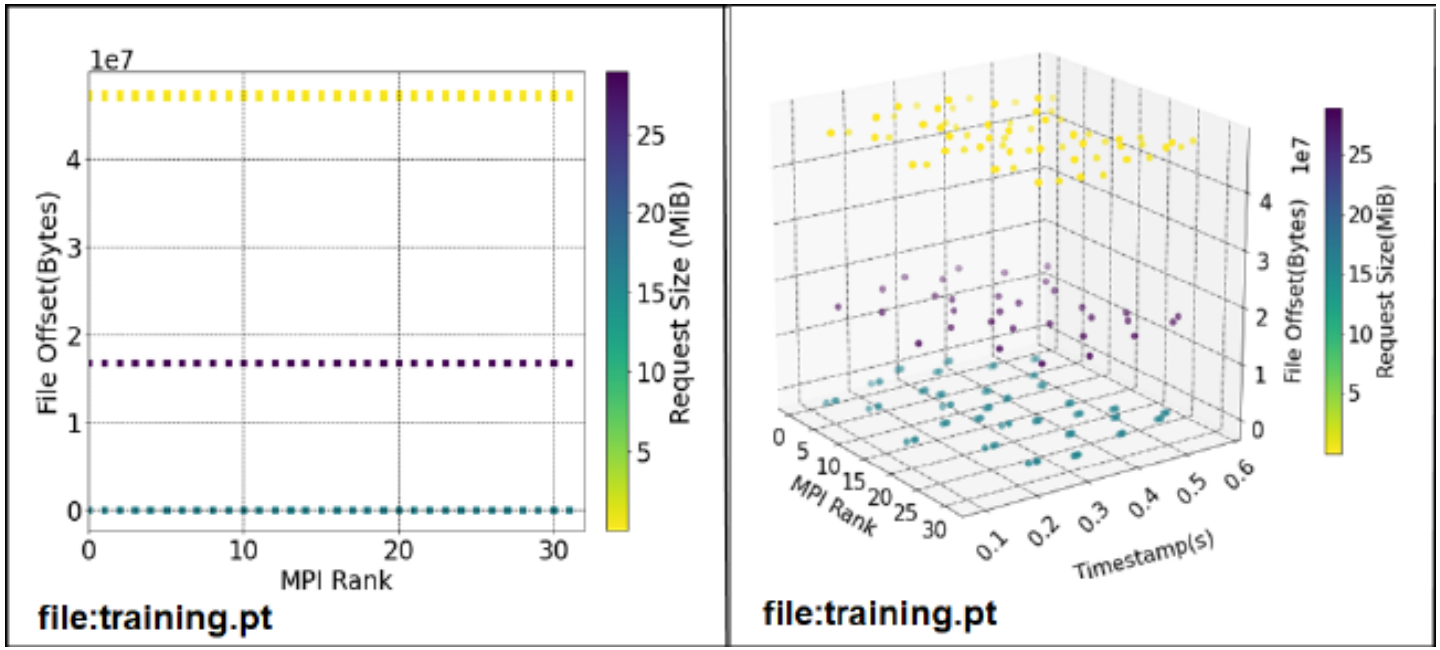
tensorflow2_mnist.py

- The I/O strategy observed is one file per process. The y-axis represents the offset file, x-axis the MPI process and the colorbar represents the request size in a specific offset.
- the left plot shows spatial pattern, where three main sizes can be observed.
- the right plot depicts the temporal pattern (right plot of Figure), each process performs ten readings in the following order: two operations of approximately 10 MiB in the same offset (in dark purple), small reads in different offset (in yellow), one operation of 9.8 MiB (light purple) in a different offset and finally a read of 10 MiB in the same offset of the two first operations. This behavior is similar for the other experiments by using different numbers of processes and mapping.

Pytorch MNIST (1)

pytorch_mnist.py

Run by using 32 processes in 8 nodes and a mapping of four processes per node. It is observed 1 file for training and a file for testing. Both files are opened as shared files for all the processes.



Pytorch MNIST (2)

pytorch_mnist.py

- The I/O strategy is a single shared file. The y-axis represents the offset file, x-axis the MPI process and the colorbar represents the request size in a specific offset.
- For the spatial pattern all the processes perform eleven readings, of which three readings are 16MiB, and one is approximately 28.86MiB. The rest of the readings are very small.
- For temporal pattern each process performs eleven operations in the following order: 2 read operations of 16 MiB at file offset 0 (blue-green), small reads at the end of the file (yellow), one operation of 16 MiB at file offset 0 (blue-green), one operation of 30 MiB at the offset file 16MiB (dark purple) and, finally, a small operation at end of the file (yellow).

Conclusions

- Two different types of applications were analyzed, which were selected because they present different I/O strategies for reading datasets.
- In the case of TensorFlow2, the I/O strategy is one file per MPI process; where each process reads its own file. As in traditional HPC, these kinds of patterns represent a problem for scaling the application.
- In the case of PyTorch, all processes accessed a single shared file, but each one reads the whole file. In this case are small files, but this pattern is a problem for large file that will consume more memory in the training stage.
- The mainly I/O limitation observed for the Horovod DL framework is primarily related to parallel I/O approaches.
- This work presents a first approximation with relatively small datasets and workloads in order to observe in more detail the execution of Deep Learning applications and thus be able to apply it on a larger scale later.