Accelerating I/O for Traditional HPC and Modern ML Workloads on Emerging HPC Systems

KTH Royal Institute of Technology, Stockholm, Sweden

Steven W. D. Chien *wdchien@kth.se* Advisors: Stefano Markidis, Artur Podobas *{markidis, podobas}@kth.se*

KTH ROYAL INSTITUTE OF TECHNOLOGY

The Problems and Motivation

- High-Performance Computing (HPC) systems emerged as a good choice for large-scale parallel Deep-Learning relative to cloud computing or locally setup clusters.
  - A lot of accelerators without virtualization
  - Fast interconnect for reduction
- Machine Learning (ML) workloads have very different data ingest patterns than HPC applications
  - Existing I/O infrastructures are optimized for HPC applications
  - Read intensive vs Write intensive
  - Independent I/O vs collective I/O
- HPC I/O subsystem is becoming **increasingly heterogeneous** and disaggregated.
  - It is unclear how both traditional and emerging workloads can take advantage of these systems.

Research Questions

We attack the problems from two angles:

- How do emerging workloads (ML/AI) differ from traditional HPC applications, what are the challenges and how to overcome them?
- How can HPC applications exploit emerging HPC systems with heterogeneous I/O?

What are the challenges of running modern AI workloads on HPC systems?

Characterizing I/O in DL workloads

- ML frameworks are optimized for the cloud without specific support to HPC I/O infrastructures (for example MPI-IO). **What are their techniques and how well do they perform?**

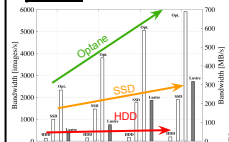


Fig.1

- **Optimization #1. Threading**
  - ML I/O pipelines are embarrassingly parallel. I.e. to fetch samples, preprocess them, and accumulate.
  - These pipelines (producers) can be executed in parallel independently.
- **Optimization #2. Prefetch**
  - Assuming the I/O pipeline (producer) on the CPU runs slower than the training on GPU (consumer), the GPU will be idle while the producer prepares a new batch of data.
  - By running the pipeline in parallel, the GPU can draw a new batch as soon as it has finished training the current batch.
  - **In certain cases, prefetch can result in a complete overlap of I/O and training (Fig.3).**
  - **Prefetching results in higher overall bandwidth and less idling time (Fig.2).**
  - **However, bandwidth contention can become a bottleneck** for both threading and prefetching, depending on the underlying hardware (Fig.1).

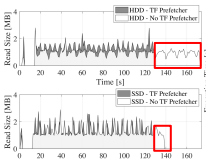


Fig.2

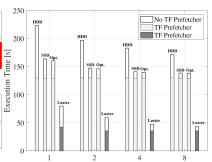


Fig.3

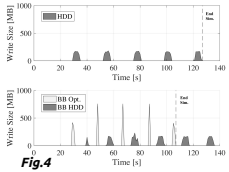


Fig.4

- **Optimization #3. Fast Checkpoint**
  - While ML I/O is read-intensive, blocking checkpointing of networks can also harm performance.
  - We create a prototype burst buffer that saves and sync a checkpoint to a fast Intel Optane SSD and asynchronously (through file system cache) copy the checkpoint to a hard disk. (Fig.4)
  - **We show that the burst buffer can effectively absorb the write I/O and significantly reduce the idle time (~2.6× improvement relative to a checkpoint in slower storage).**

tf-Darshan

- Our previous work gives coarse-grained characteristics of ML I/O workloads, but it does not answer the question on how the I/O is affected by the underlying workloads.
- To further understand the bottlenecks and ML workloads, **we develop a tf-Darshan, an I/O profiler that is tightly coupled with the new TensorFlow profiler** (Fig.5).
- We combine the data collection capability of the Darshan I/O profiler and compute information from tf.Profiler to give a fine-grained I/O characterization.

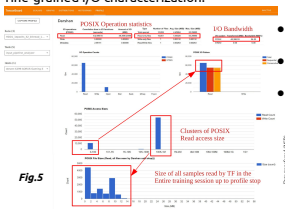


Fig.5

- **One major contribution of tf-Darshan is the ability to perform runtime instrumentation to allow snapshot profiling of a long training session** (Fig.7).
- The **lack of co-design between workloads and I/O (such as MPI-IO and PFS) can significantly hurt I/O** performance (e.g. only ~3 MB/s when reading a lot of small images from HDD).
- We find that TensorFlow performs a large number of small reads, and zero-sized reads to signal end of file (illustrated and annotated in red in the timeline in Fig.6 below), which translate to expensive POSIX I/O operations.

Fig.6

File name	Timeline (POSIX I/O Activity)
image_1.jpg	Read, Read
image_2.jpg	Read
image_3.jpg	Read, Read, Read

- We use the cluster of reading size collected from tf-Darshan to perform optimizations.
- **By staging a small portion of data with small file sizes to a fast storage device, we can improve bandwidth by up to 19%.**
- Our work highlights the importance of co-designing I/O and underlying workloads.

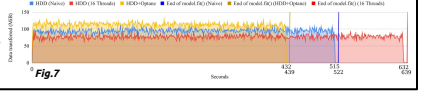


Fig.7

How can HPC applications take advantage of emerging I/O technologies?

Emulating Object Storage on HPC system

- POSIX I/O with strong consistency and coupled metadata management has long been considered a limiting factor for scalable I/O. *Object storage* has attractive characteristics that can potentially become a POSIX I/O replacement. However, it is unclear how HPC applications can exploit this. Here, we show how *object storage as an I/O semantic can bring immediate benefit even existing systems*.
- We design an object storage emulator to emulate the key characteristics of object stores: **immutability**, **flat namespace**, and **eventual consistency**. These are the key factors that leads to a high-performance in I/O.

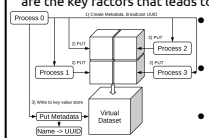


Fig.8

- To support HPC applications, we specifically support applications that use *domain decomposition* (Fig.8).
- **Our emulator allows applications to compose objects using one HDF5 file per process**, conceptually similar to the *subfiling* technique.
- To support object composition, **we use HDF5 Virtual Dataset (VDS) to link all objects together** to provide a global view.

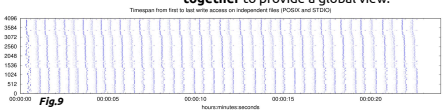


Fig.9

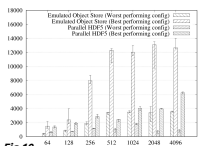
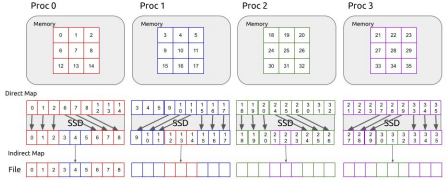


Fig.10

- We emulate metadata operations by writing serialized Protocol Buffers (protobuf).
- To simulate weak consistency, only root rank writes metadata through POSIX atomic rename. Processes do not need to synchronize after writing their partial objects (Fig.9).
- **We profile a benchmark application with Darshan and show significant bandwidth improvement relative to Parallel HDF5 (~80% reduction in time)** (Fig.10).

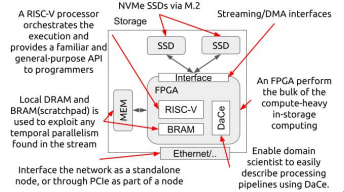
Work in Progress

A programming model for distributed I/O



- ***Research Question***: Existing parallel I/O models (such as MPI-IO) optimize the writing of shared files between processes (e.g. data sieving, non-contiguous access). These implementations are often co-designed with parallel file systems that are visible globally. *Yet, emerging I/O systems in the upcoming HPC systems are often local and disaggregated.* **How can applications use fast and distributed node-local storage to accelerate the writings of shared files?**
- ***Approach***: We envision a library that enables shared file I/O through distributed storage and global memory abstraction. By using a mmap-like memory allocator, we can translate memory operations into remote I/O that stage many files on node-local storage. They can be easily combined into a common file when staged to the parallel file system.

In-storage compute for ML preprocessing



- ***Research Question***: While existing I/O optimizations for ML (such as threading and prefetching) can partially mitigate I/O bottlenecks, issues such as small file sizes and bandwidth contention remain a problem. *At the same time, both I/O and preprocessing take up considerable CPU resources.* Our work highlighted the importance of co-designing I/O and workloads. **Can we co-design the hardware in a similar manner to also benefit preprocessing workloads?**
- ***Approach***: We propose a new kind of accelerator in form of FPGA that is attached directly to the storage system. Both I/O and preprocessing compute workloads can be offloaded accelerated by the FPGA. Data batches can be aggregated before being transferred to compute nodes for training.

Conclusions

- We researched and developed tools that characterize and explore ML I/O workloads that have very different I/O patterns comparing to traditional HPC applications.
- Our work highlights the importance of co-designing between I/O and underlying workloads and our tools provide the knowledge required to enable such optimizations.
- We also highlight the challenges of using emerging I/O technologies and how HPC applications can adapt to new programming models.

References

1. S. W. Chien et al., "Exploring Scientific Application Performance Using Large Scale Object Storage," In: High Performance Computing. ISC High Performance 2018.
2. S. W. D. Chien et al., "TensorFlow Doing HPC," *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019.
3. S. W. D. Chien et al., "Characterizing Deep-Learning I/O Workloads in TensorFlow," 2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS), 2018.
4. S. W. D. Chien et al., "tf-Darshan: Understanding Fine-grained I/O Performance in Machine Learning Workloads," 2020 IEEE International Conference on Cluster Computing (CLUSTER), 2020.
5. S. Narasimhamurthy et al., 2018. The SAGE project: a storage centric approach for exascale computing: invited paper. In *Proceedings of the 15th ACM International Conference on Computing Frontiers (CF '18)*.

Acknowledgements: This work received funding from the European Commission H2020 program under grant agreement No. 801039 (EPiGRAM-HS), grant agreement No. 800999 (SAGE2), and grant agreement No. 800904 (VESTEC). The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at PDC and HPC2N and partially funded by the Swedish Research Council through grant agreement no. 2018-0597.