Addressing I/O Bottleneck for Large-Scale Deep Learning Applications On HPC Systems

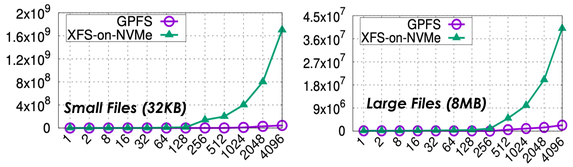
Awais Khan, Soon Hwang, Chris Zimmer, Sarp Oral, Youngjae Kim
Email: {khana, zimmercj, oralhs}@ornl.gov, {soonhw, youkim}@sogang.ac.kr

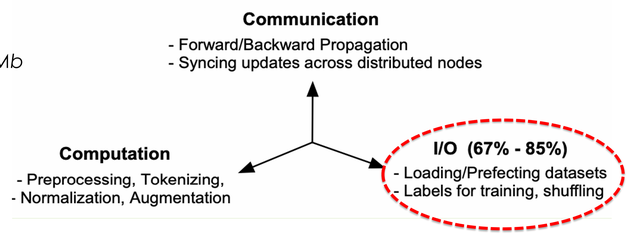
Introduction

- Deep Learning (DL) applications are becoming an increasingly important workloads on HPC systems such as ***Summit and Frontier***
- To efficiently ***Run and Scale DL Applications*** to leverage state-of-the-art HPC system remains challenging
- Existing prefetching and caching solutions pose non-trivial challenges such as
  - ***Fail to fully utilize the compute node-local NVMes,***
  - ***Require modifications to applications or input I/O pipeline***
  - ***Incur additional metadata overhead and bottlenecks***

Motivation and Research Challenges

- I/O optimization for DL Applications is non-trivial challenge
  - ***Dataset characteristics, DL Access patterns, and I/O properties***
- DL applications running at large-scale training environments spend
  67-85% of their ***execution time performing I/O to PFS***
- MDTest on Summit for DL workload access patterns
  - ***(Open-Read-Close)*** *with two different file sizes, 32Kb and 8Mb*



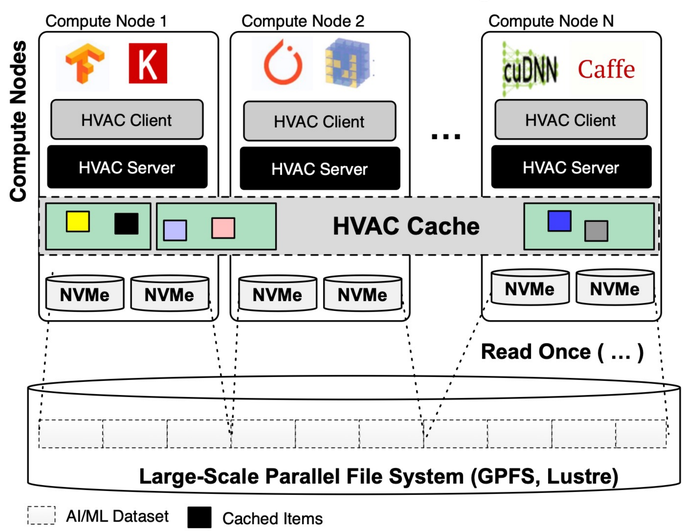


- ***Opportunity to exploit node-local or near-node local storage on compute nodes and solve I/O Scalability limitations by layering a caching system***
  - With diverse deployment model, portable and POSIX support, no metadata slowdown, no repeated rereads from PFS

An overview of High-Velocity AI Caching System

"To Scale on thousands of compute nodes on leadership-class supercomputer such as Summit and Frontier without modifying DL applications and additional metadata bottlenecks and storage overhead"

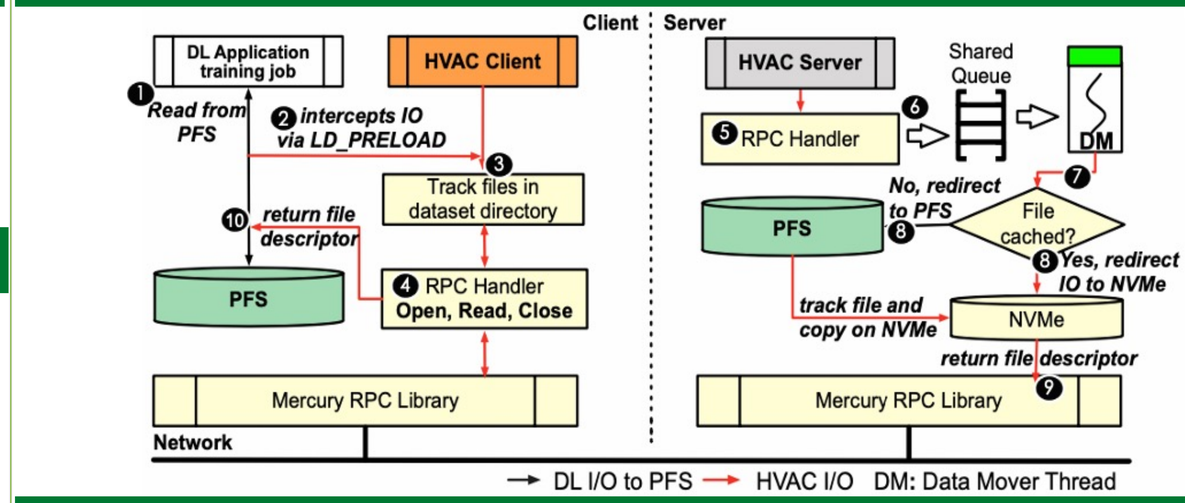
- A simple, lightweight and transparent library intended to accelerate I/O accesses for DL applications that utilize read-only data with a ***high re-read rate***
- Architecture agnostic and can be deployed on node-local, near-node local or rack-local storage
- Supports POSIX operations (open, read and close) via ***LD_PRELOAD*** and fully portable, requiring no changes to application or PFS
- Guarantees ***no repeated re-reads*** from underlying PFS, once files are cached in node-local storage
- Consists of two components:
  - ***Client:*** intercepts the application I/O calls
  - ***Server:*** processes the I/O operations and cache dataset locally on node-local NVMe



Acknowledgments

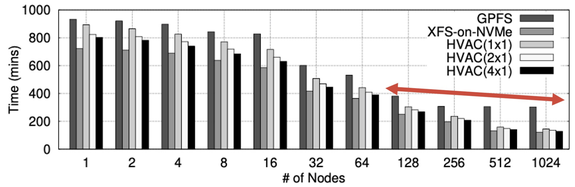
This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the DOE under Contract DE-AC05-00OR22725.

Accelerated I/O Service Flow

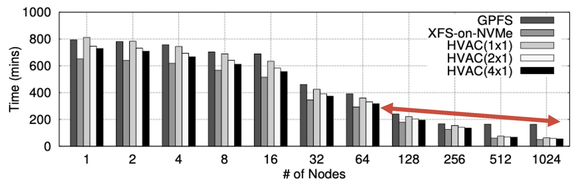


Scalability Study on Summit

- HVAC compared against ***GPFS***, ***XFS-on-NVMe*** (pre-staged whole dataset to each node-local storage)
  - ***HVAC (i x1): no. of HVAC server*** instance(s) running on each compute node
- Scalability analysis with ***increasing no. of compute nodes***
  - Average performance improvement is over 50% for all HVAC variants atop GPFS

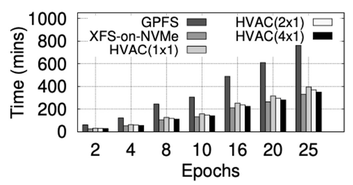


(a) ResNet50 [BS:50, Eps:10, nProcs/node:2]

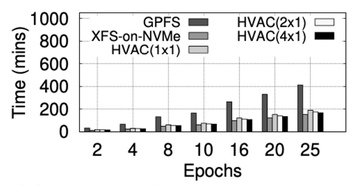


(b) CosmoFlow [BS:80, Eps:10, nProcs/node:2]

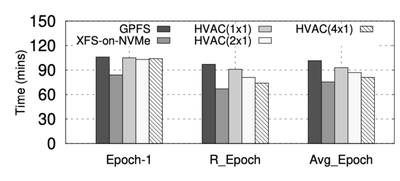
- Scalability analysis with ***increasing no. of epochs***



(a) ResNet50 [BS:80, nNodes:512]



(b) CosmoFlow [BS:4, Eps:10, nNodes:512]



(c) Per Epoch Analysis

Conclusion

- HVAC is a scalable read-only caching system for HPC systems such as Summit and Frontier
- Exploits compute node-local storage and builds an aggregate cache layer atop to accelerate the DL application training
- Scales linearly on 1024 summit nodes and shows an average 25% performance improvement atop GPFS and 9% drop with XFS-on-NVMe

Work In Progress (WIP)

- Open-sourcing the HVAC software library and deployment on Frontier
- Comparison with State-of-art works using large-scale DL models such as DeepCAM, OpenCatalyst
- Prefetching and chunk-granularity caching to fully utilize multiple node-local SSDs in Frontier compute nodes, i.e., two node-local NVMes per compute node