HEP CCE Input/Ouput Storage (IOS)

Saba Sehrish for the HEP CCE IOS team

CSAID Roadmap meeting November 2023

Acknowledgement

These are the slides that were presented at the DOE review in July (few ATLAS-specific slides removed), and the last slide is from the next phase of CCE planning.

Input/Output and Storage: Fermilab Participants

Philippe Canal
Patrick Gartung
Ken Herner
Christopher Jones
Kyle Knoepfel
Saba Sehrish
Elizabeth Sexton-Kennedy

CCE IOS Activities

Measuring performance of ROOT I/O in HEP workflows on HPC systems

- Darshan (a scalable HPC I/O characterization tool) has been enhanced (including fork safety) and used to monitor HEP production workflows
- Use Darshan to monitor I/O for various HEP production workflows on HPC

Mimicking framework for understanding scalability and performance of HEP output methods

- Experiment agnostic tool allows scaling I/O beyond what is currently accessible by production and has uncovered/fixed bottlenecks in ROOT and frameworks

Investigate HDF5 as an intermediate event storage for HPC processing

- Relying on ROOT to serialize complex Event Data Model used in Simulation/Reconstruction workflows
- Implementing Collective Writing to avoid potential merge step

HPC friendly Data Model

- Together with PPS team started investigating efforts to make data model more suitable for offloading to accelerators and storage on HPC systems

Darshan: Measuring performance of ROOT I/O in HEP workflows on HPC systems

HEP-CCE

Darshan https://www.mcs.anl.gov/research/projects/darshan/

- Darshan is a lightweight I/O characterization tool that captures concise views and entire traces (DXT) of applications' I/O behavior
  - Widely available – Deployed at many HPC facilities
    - LCFs, NERSC, etc. and CVMFS
  - Popular tool for HPC users to better understand their I/O workloads
    - Easy to use – no code changes required
    - Modular – straightforward to add new instrumentation sources

Darshan enhancements for HEP use cases

- **Improved filtering** of what files are tracked, to focus on data of interest
- **Instrumentation of non-MPI jobs** was exercised and improved to meet HEP workflow needs
- **Clean handling of fork()** was implemented to separate child I/O from parent, important for ATLAS (using multi-process AthenaMP) and DUNE (python wrapper)

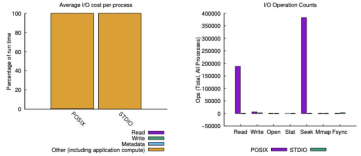
CMS workflow

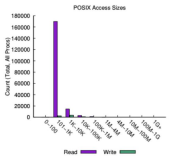
- Created a native build of CMSSW data dictionary libraries on supercomputers for use with CCE-IOS serialization test framework
- Measured ROOT IO for a CMSSW workflows on shared and local file system with Darshan
- Compared the Darshan reports for shared and local filesystem.
- The I/O rates are consistent with those expected for local HDD/SSD and shared file systems.
- Example Darshan report ->

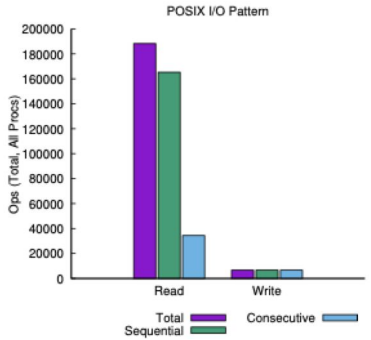
cmsRun (1/28/2021) 1 of 3

jobid: 3698	uid: 72001	nprocs: 1	runtime: 383 seconds

I/O performance *estimate* (at the POSIX layer): transferred 456.7 MiB at 321.38 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.0 MiB at 2.61 MiB/s







Most Common Access Sizes (POSIX or MPI-IO)

	access size	count
POSIX	300	154
	271	108
	322	87
	293	72

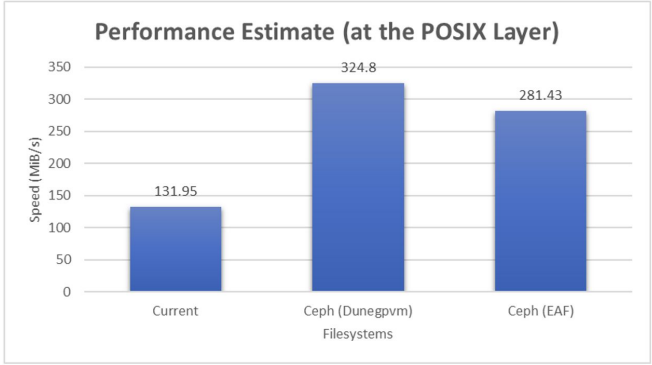
File Count Summary (estimated by POSIX I/O access offsets)

type	number of files	avg. size	max size
total opened	7	269M	1.9G
read-only files	3	619M	1.9G
write-only files	2	13M	26M
read/write files	0	0	0
created files	2	13M	26M

cmsRun step4_PAT_PU.py

DUNE workflow

- Studied some DUNE workflows and built Darshan against usual DUNE software release both on dunegpvms and NERSC (inside usual container)
- Similar access patterns as CMS
- Used this summer as part of benchmarking studies for upcoming Ceph migration. Some unexpected results, including identifying some sub-optimal networking setups on EAF. ->



(numbers from Darshan runs)

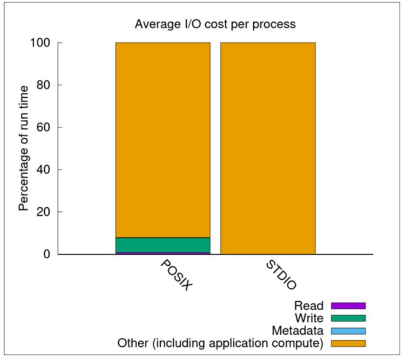


Fig 3. The percentage of time the machine spent on reading, writing, Metadata, and computing for the Ceph Filesystem.

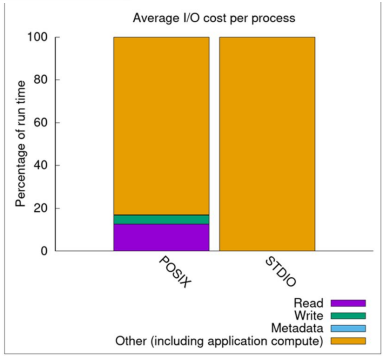


Fig 4. The percentage of time the machine spent on reading, writing, Metadata, and computing for the current Filesystem.

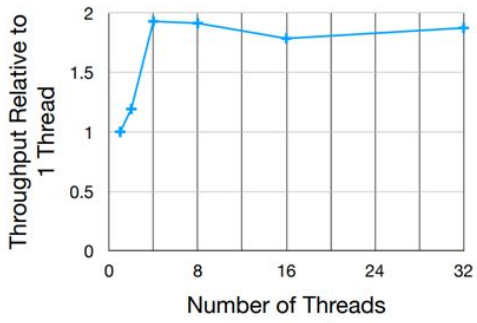
Nehemyah Green, GEM student

Mimicking framework

HEP-CCE

Mimicking framework for understanding scalability and performance of HEP output methods

- Experiment agnostic tool allows scaling I/O beyond what is currently accessible by production
  - Easily try different approaches and I/O technologies
  - With ability to read actual experiment ROOT files
  - Make it easier to study read/write performance and thread scalability
  - Can test input and output formats and approximate HEP job timings
- Has uncovered/fixed bottlenecks in ROOT and frameworks, e.g. ROOT PR 6062

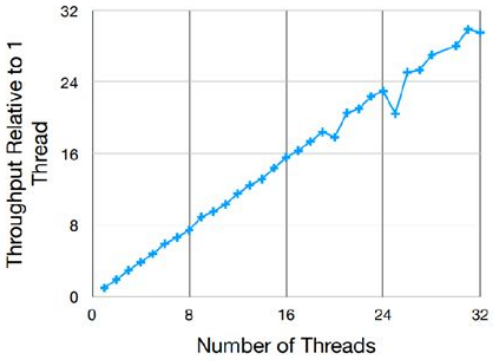


Frequent setting of an atomic value

```
Version_t GetClassVersion() const {
      fVersionUsed = kTRUE;
      return fClassVersion; }
```

Changed to

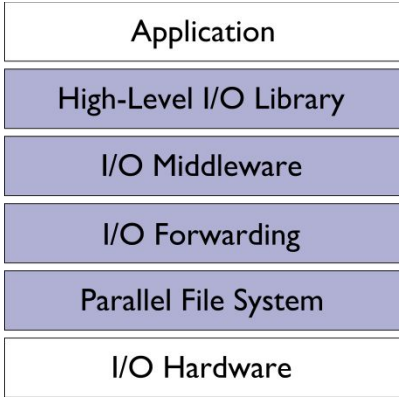
```
Version_t GetClassVersion() const {
      if (!fVersionUsed.load())
         fVersionUsed = kTRUE;
      return fClassVersion; }
```



HPC storage systems

Optimizing HEP workflow I/O for HPC storage systems

- Differences in HTC and HPC storage resources means current HEP computing workflows may use HPC systems suboptimally



- HPCs use parallel file systems for data-storage and access
- HPCs have an established I/O software stack used to support parallel file system
  - High Level libraries can hook into the HPC I/O stack
  - Allows us to take advantages of optimizations such as collective I/O
- **ROOT** has been I/O & storage workhorse of HEP experiments
  - HEP data models are complex
  - Use ROOT to read and write data into ROOT::TTree
- However, storage backends such as **HDF5** are better integrated with HPCs and their expert community
- **Not** an effort to replace ROOT (TTree) storage!
- But augment workflows capability to store intermediate data in HDF5 for HPC processing

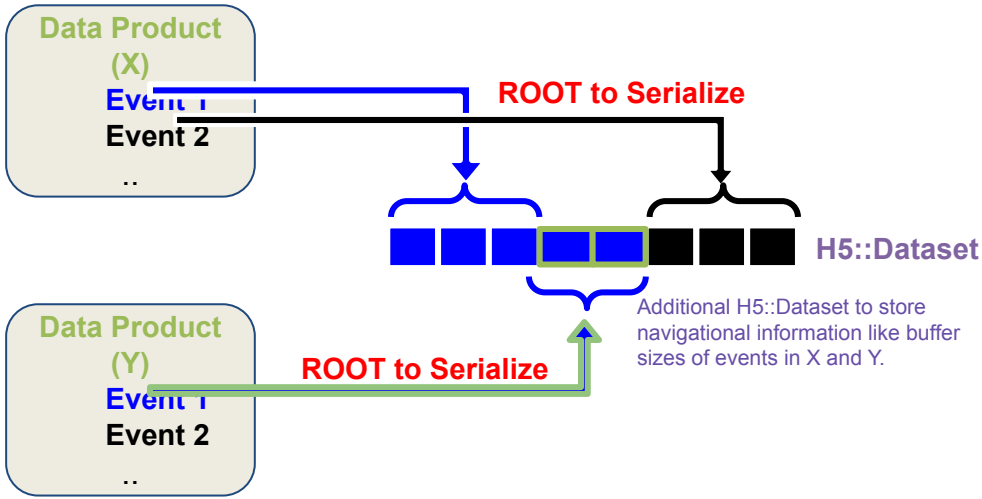
Investigate HDF5 as intermediate event storage for HPC processing

HEP-CCE

Use ROOT serialization to hide HEP data model complexity from HDF5

- Keeps using one of ROOT's proven strength that has been developed as part of the HEP community
  - and on which much of HEP data model were build
- Means HDF5 only has to manage binary data (BLOB)
  - could be seen as an obstruction for final data, but this is intended for intermediate data only

- **Data Products** are experiment specific C++ objects usually written in ROOT format
- Use **ROOT** as common tool to **serialize** C++ objects into byte stream array buffers
- **HDF5 Datasets** store serialized data products with mapping optimized for parallel I/O. Mapping is independent of experiments
- Mapping can be done either on **Data Product** or **Event** level, depending on access needs

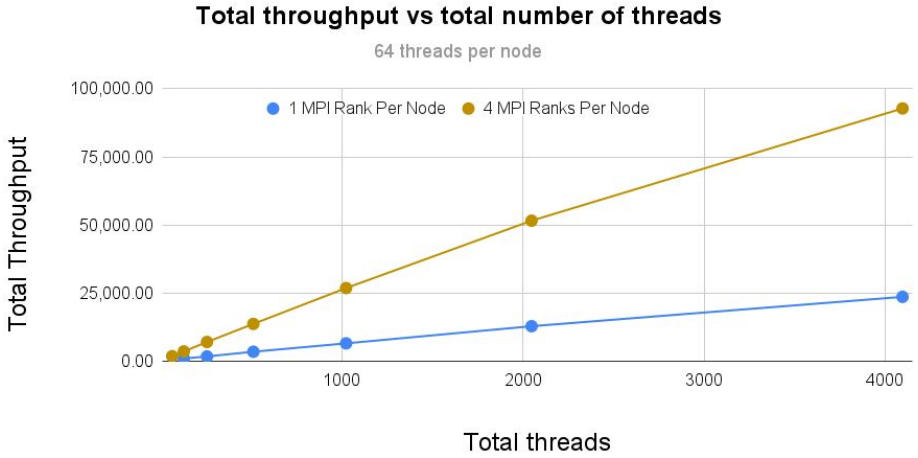


Parallel HDF5 results

HEP-CCE

Parallel HDF5

- In addition to using serial HDF5 and understanding its usability and performance in the HEP application space, IOS also worked on designing and implementing output modules with **parallel HDF5**
- Parallel HDF5 allows multiple processes to write to a **single output file**. It eliminates the need for merging files



I/O Calls	Fraction of Total I/O Time
MPI calls (external to HDF5)	14%
Write data into HDF5 file	32%
Other (including serialization)	54%

Results

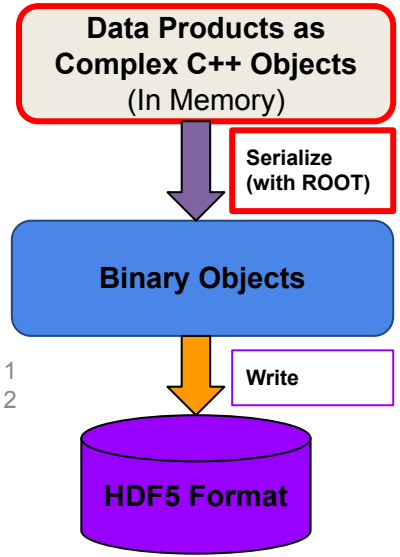
- Studies with ROOT-serialized events on CORI@NERSC showed good scaling
- For collective I/O the MPI call overhead (coordinating data placement) was found to be small

Leveraging HDF5 work for an HPC-friendly data model

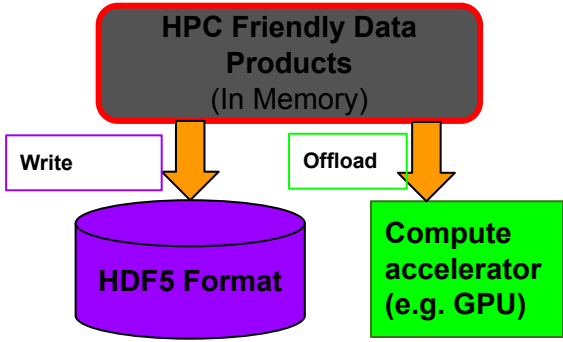
HEP-CCE

Complexity of the HEP data model is a challenge for both efficient offloading and storage

- Solution should address both requirements
- Writing data to HDF5 via ROOT serialization will not allow these objects to be offloaded to GPU



- Continue work on HDF5 and other storage technologies for HEP data without ROOT serialization;
  - ProtoDUNE and DUNE are the primary targets
  - Some data products for ATLAS/CMS do not rely on very complex data models
    - E.g., the structure of analysis products is much simpler and could potentially be stored in HDF5 with direct mapping



HPC friendly Data Model: First Steps *HEP-CCE*

Work on developments for HPC Friendly Data Models for HEP experiments was initiated by IOS & PPS:

- Modern HPCs rely on heterogeneous resources (often CPU+GPU) for compute acceleration
- HEP data models: Heavily object-oriented and therefore
    - typically not GPU (and thus HPC) friendly
    - challenging to store in store backends such as HDF5

- Survey carried out by HEP-CCE on experiments' efforts to make their data model HPC friendly
    - Speakers from **ATLAS, CMS, DUNE, NOvA and EDM4HEP** developers invited to share their experience on developing HPC friendly data models
    - Results recorded: https://github.com/hep-cce2/GPU-DM
    - Goal is not to duplicate these efforts, but augment them to make results more generic and alloc common use

Input/Output and Storage: Summary

Work of the HEP-CCE/IOS team has resulted in:

- Worthwhile insight to I/O behavior of HEP workflows
  - Including on HPC and for scales beyond current production
- Fixes/enhancements to common software and experiments frameworks
  - Darshan included fork-safety and better filtering for I/O.
  - ROOT serialization bottleneck was fixed
  - Patch to resolve the Athena library issue on DSO loading hooks which cause PyRoot crash when running with Darshan
- Prototype development of new functionality in collaboration with experiments:
  - ATLAS developed functionality to store their production data in HDF5

Next CCE: Storage Optimization (SOP) Topics

HEP-CCE

HEP-CCE/IOS is looking forward to optimizing Data Storage for next-generation HEP experiments:

- Applying Lessons Learned to HEP Experiments, Mimicking Framework, Darshan and HDF5
- Tracking and aiding the evolution of ROOT I/O, in particular RNTuple
- Reduced Precision and Intelligent Domain-specific Compression Algorithms
- Object Stores and Strategies for Data Placement and Replication
- Optimized Data Delivery to HPC systems