Mapping ROOT RNTuple I/O data structures to DAOS objects

Javier López-Gómez *
Vincenzo Eduardo Padulano **

CERN openlab Technical Workshop, 2022-03-21

* CERN (CH)
** Valencia Polytechnic University (ES)

Contents

RNTuple Goals and Overview

Reminder: HENP Event Data I/O

Why invest in tailor-made I/O sub system (TTree / RNTuple)

- Capable of storing the HENP event data model: nested, inter-dependent collections of data points
- Performance-tuned for HENP analysis workflow (columnar binary layout, custom compression, etc.)
- Automatic schema generation and evolution for C++ (via cling) and Python (via cling + PyROOT)
- Integration with federated data management tools (XRootD, etc.)
- Long-term maintenance and support

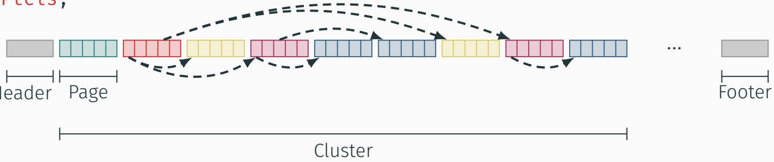
RNTuple Goals

- Less disk and CPU usage for same data content
  - **25% smaller files, $\times$2-5 better single-core performance**
  - 10GB/s per box and 1GB/s per core sustained end-to-end throughput (compressed data to histograms)
- Native support for object stores (targeting HPC)
  - **DAOS: collaboration between CERN, Intel, and HPE**
  - Experimental support for S3, ...
- Lossy compression
- Systematic use of exceptions to prevent silent I/O errors

- Getting ready for a new hardware landscape: architectural heterogeneity, parallelism on all levels, blurring between device classes (e.g. active storage, NV-DIMMs)

RNTuple On-disk File Format

```
struct Event {
    int fId;
    vector<Particle> fPtcls;
};
struct Particle {
    float fE;
    vector<int> fIds;
};
```



Page: Array of fundamental types (maybe compressed). Size defaults to 64 KiB, but can be adjusted

Cluster: Set of all the pages containing data for a specific row range, e.g. 1–1000

Header and Footer: Information about the schema and location of pages/clusters, respectively

Storing RNTuple data in DAOS

Why Intel DAOS?

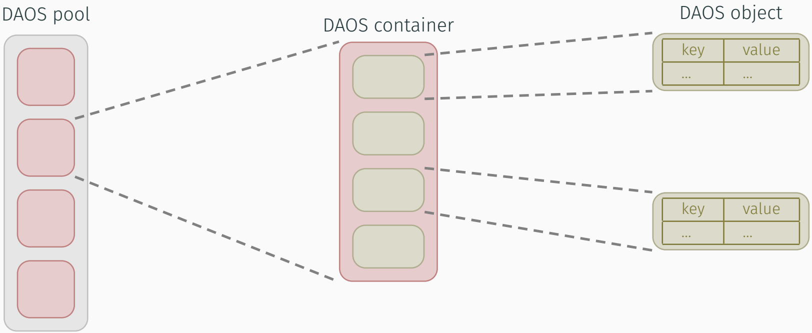
Issues with traditional storage stack

- Designed for spinning disks (few IOPS): I/O coalescing, buffering, etc., became less relevant for modern devices.
- POSIX I/O (strong consistency), has been acknowledged as a major problem for parallel filesystem scalability.

- Modern fault-tolerant object store optimized for high bandwidth, low latency, and high IOPS. Foundation of the Intel exascale storage stack.
- Optimal use of Intel Optane DC persistent memory and NVMe SSDs; access time $O(\mu s)$.
- Argonne's Aurora[1] I/O system will be based on DAOS.
- Experience acquired supporting this in RNTuple can be reused for other object stores (e. g. Amazon S3).

[1]https://alcf.anl.gov/aurora

DAOS Overview



- **Object:** a Key–Value store with locality.
  - The key is split into **dkey** (distribution key) and **akey** (attribute key).
  - dkey affects data locality: DAOS guarantees that same dkey maps to same target.
- **Object class:** determines redundancy (replication/erasure code).

Legacy software can still use DAOS through its compatibility layer, i.e.

- **POSIX filesystem (libdfs).** Can be used either through `libioil` (I/O call iterception) or `dfuse` (FUSE filesystem).
- **MPI–IO.** Provides DAOS support through a ROMIO driver (MPICH and Intel MPI).
- **HDF5, Apache Spark, ...**

...although throughput may not be on par to direct use of libdaos.

Alternatives for mapping clusters/pages to DAOS

- One page per OID in a single *akey*. Constant *dkey*.
- One cluster per OID and one *akey* per page in the cluster. Constant *dkey*.
- One cluster per OID ★. Same as above, but varying *dkey* (e.g. one *dkey* per page group, where a "page group" is all the pages of a certain column in a certain cluster).

RNTupleReader: file vs. DAOS

Only requires the replacement of the file path

```
auto ntuple = RNTupleReader::Open("DecayTree",
      "./B2HHH~zstd.ntuple");
```

to a **daos://** URI

```
auto ntuple = RNTupleReader::Open("DecayTree",
      "daos://e6f8e503-e409-4b08-8eeb-7e4d77cce6bb/b4f6d9fc-e081-41d4-91ae-41adf800b537");
```

Evaluation

Test environments

- **CERN openlab:** 3 servers, 2 clients. Intel Omni-Path.
- **HPE Delphi:** 2 servers, 9 clients. Mellanox InfiniBand.

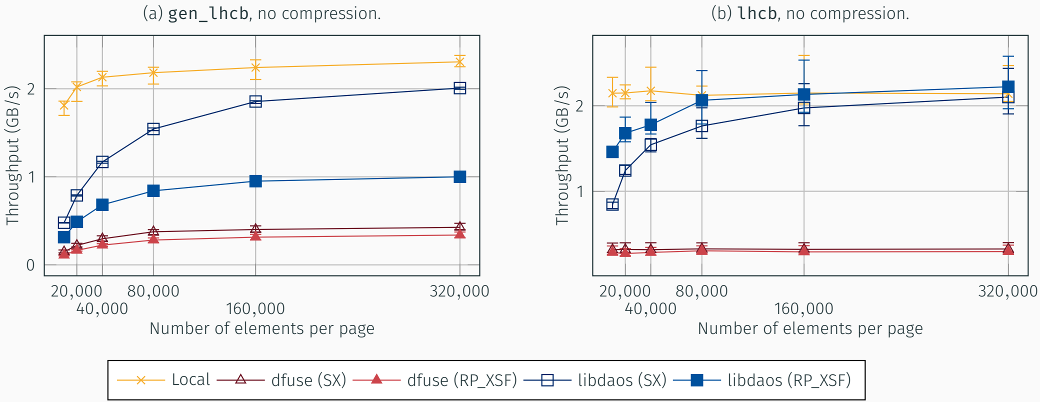
Test cases

Steps: (a) move data into DAOS, and
(b) run analysis using imported data.

Conditions:

1. **Constant page size, increasing cluster size.** Observe the effect of queuing many small read operations.
2. **Increasing page size, constant cluster size.** Impact of the I/O request size on the throughput.

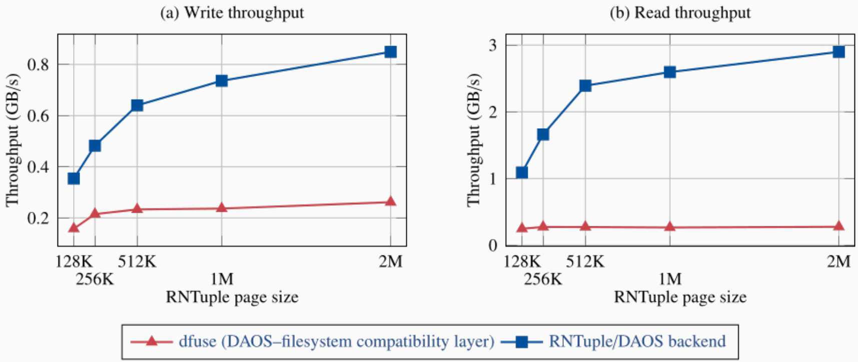
CERN openlab: one page per OID, increasing page size, constant cluster size



HPE Delphi: one page per OID, increasing page size, constant cluster size

- Preliminary tests: poor performance of ∼550MB/s.
- Wrong use of event queues: EQ created (destroyed) before (after) each bulk read.

Single-process, single-thread results after patching:



Conclusion

Conclusion

- Higher read throughput with large pages (larger transfer size).
- RNTuple libdaos-based backend outperforms `dfuse` in our tests.

- Room for improvement, e.g.
  - issue a single `daos_obj_fetch()` call to retrieve data for multiple pages (implies using “One cluster per OID” mapping, at least)
  - improve use of event queues

Next steps

- General refactoring of the DAOS backend.
- Optimize ingestion of existing HEP data into a DAOS-based data center.

Thanks!

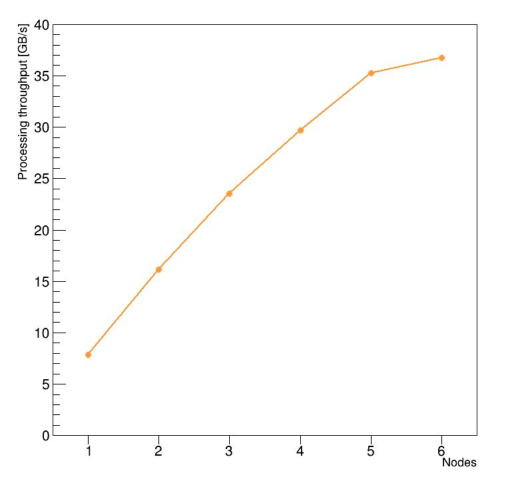
Backup

DistRDF + RNTuple/DAOS Caching: HPE benchmark (1)

- Benchmark based on LHCB opendata B2HHH
- 800 GB dataset cache on DAOS
- Read and process with distributed RDataFrame + RNTuple DAOS backend

DistRDF + RNTuple/DAOS Caching: HPE benchmark (2)

processing throughput



- First working example of distributed RDataFrame reading RNTuple data!
- DAOS backend just works, even when issuing read requests from multiple nodes
- 70% of the nominal bandwidth (48 GB/s) of the cluster achieved