Mapping ROOT RNTuple I/O data structures to DAOS objects

Javier López-Gómez *
Giovanna Lazzari Miotto **
Vincenzo Eduardo Padulano ‡

CERN openlab Technical Workshop, 2023-03-16

* CERN (CH)
** Federal University of Rio Grande do Sul (BR)
‡ 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)

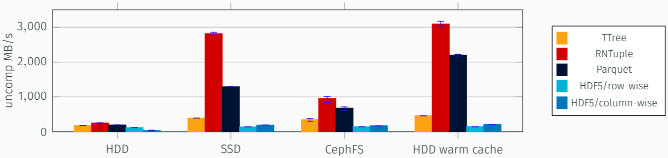
- Capable of storing the HENP event data model: nested, inter-dependent collections of data points
- Performance-tuned for HENP analysis workflow (columnar binary layout, custom compression, etc.)
- Automatic schema generation and evolution for C++ (via cling) and Python (via cling + PyROOT)
- Integration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

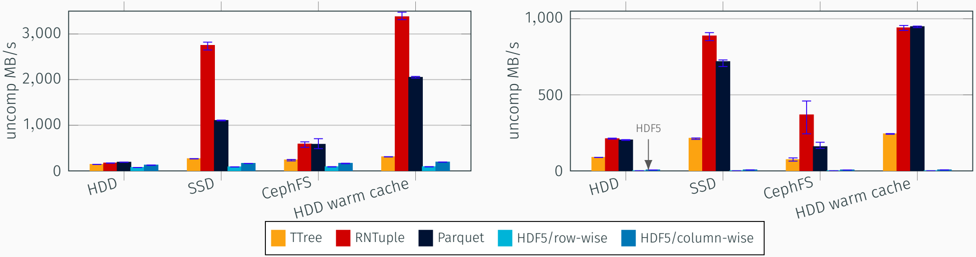
- Getting ready for a new hardware landscape: architectural heterogeneity, parallelism on all levels, blurring between device classes (e.g. active storage, NV-DIMMs)

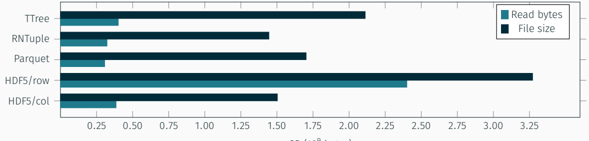
RNTuple State of Affairs: Throughput and Size



(a): LHCb B2HHH (10/26 branches; compressed)

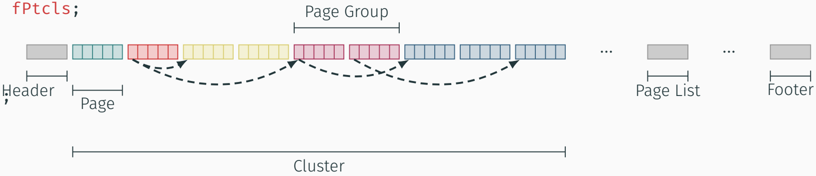
(b): CMS Higgs4Leptons (10/84 branches; compressed)





RNTuple On-disk File Format

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



- **Page:** Array of values of a fundamental type (typically compressed). Size in the ≈ tens of KiB
- **Cluster:** Comprises all pages containing data for a specific row range, e.g. 1–1000
- **Page group:** All pages that contain data for the same column in a given cluster

- **Header / Page List / Footer:** Information about the schema, cluster summaries, and location of pages

Storing RNTuple data in DAOS

Why DAOS?

Issues with traditional storage stack

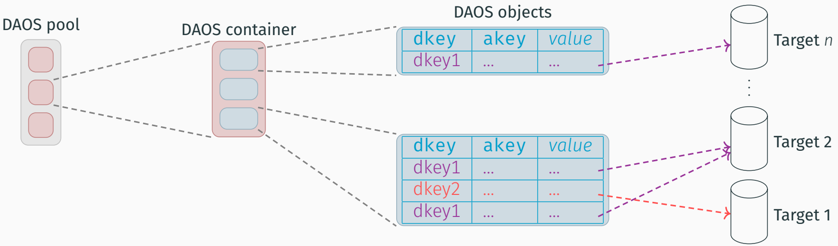
- Designed for spinning disks (few IOPS) and not ideal for NVMe devices
- POSIX I/O (strong consistency) limits parallel filesystem scalability

- Fault-tolerant object store optimized for high bandwidth, low latency, and high IOPS. Foundation of the Intel exascale storage stack
- 44% of the top 25 systems in IO500[1] based on DAOS, including ANL Aurora
- Acquired experience can be reused in implementing support for other object stores, e.g. S3.

- DAOS provides a compatibility layer, incl. POSIX filesystem (via `libioil` or `dfuse`), however... NOT ideal!

[1] `https://io500.org/`

DAOS 101



- **Object:** essentially a Key–Value store with locality, as in
  - The key is split into **dkey** (distribution key) and **akey** (attribute key), and...
  - the **dkey** impacts data co-locality: same distribution key maps to same target.
- **Object class:** determines redundancy type, i.e. replication / erasure code

What is new in 2023?

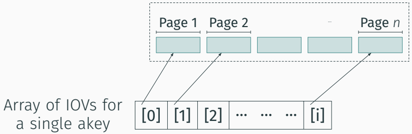
1. **Improved RNTuple $\leftrightarrow$ DAOS mapping** preserving *page* co-locality, tuned for typical HENP analysis patterns:

$$\textit{cluster} \mapsto \texttt{OID}, \quad \textit{column} \mapsto \texttt{dkey}, \quad \textit{page} \mapsto \texttt{akey}$$

2. **Coalesced R/W requests** by **{OID, dkey}** to minimize I/O calls and exploit target parallelization

3. **Vector writes**: per-cluster data buffering; issue coalesced, parallel writes

4. Multiple IOVs per *akey*: allows for transferring a page range in a single operation, targeting **high throughput independently of native page size**



5. And more: better queue management, multiple ntuples per container...

RNTupleReader: file vs. DAOS

Compromise: only change consists in replacing the file path

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

to a `daos://` URI

```
auto ntuple = RNTupleReader::Open("DecayTree",
      "daos://my-pool/my-container");
```

Evaluation

Evaluation

Test environment

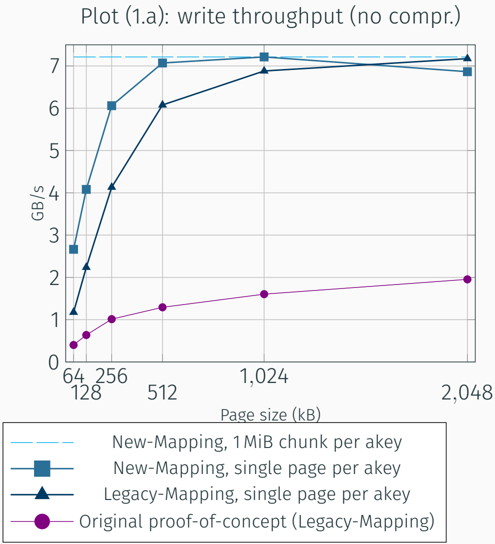
- **HPE[2] Delphi:** 2 servers, 6 client nodes. Mellanox InfiniBand.

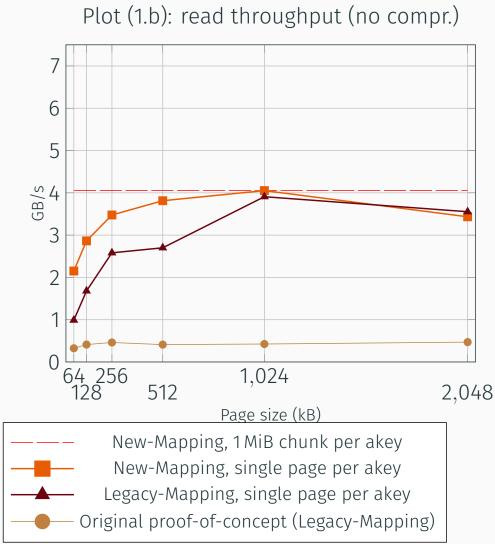
Test case

- **Steps:** (a) move data into DAOS, and
  (b) run analysis using imported data (single-process, single-node).
- **Dataset:** LHCb OpenData B2HHH: 8.5 M events, 26 branches replicated $\times$10 (total size of 15 GB).
- with/out compression (zstd) and leveraging different RNTuple-to-DAOS mappings

[2] Access to the hardware for the experimental evaluation was kindly provided by Hewlett-Packard Enterprise.

Read/Write Throughput vs. Page Size





State of Affairs: Beginning vs. End of 2022

Conclusion

Conclusion

- Many new features made it into RNTuple last year: support for new C++ types, custom collections, custom I/O rules, etc.
- Matured DAOS backend with major performance improvements, becoming ready for real-world analyses
- RNTuple is scheduled to become production grade in 2024[3]

Next steps

- Leverage single-node DAOS improvements in distributed analysis with ROOT's DistRDF
- Roll out Amazon S3 backend (coming soon)

[3] We appreciate the first experiments implementing RNTuple writers in their workflows, providing feedback on features and performance.

Thanks!

Backup

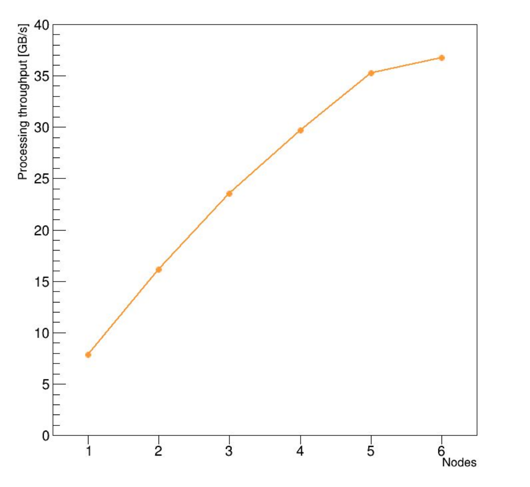
DistRDF + RNTuple/DAOS Caching: HPE benchmark (1)

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

- NOTE: the benchmark dates back to **Q4 2021**; re-running this again is still **WIP**!

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