Storing LHC Data in DAOS and S3 through RNTuple

26^{th} International Conference on Computing in High Energy and Nuclear Physics

Giovanna Lazzari Miotto – Federal University of Rio Grande do Sul (BR)
Javier López-Gómez – CERN (CH)

CHEP 2023, 9 May 2023

Contents

Introduction

ROOT RNTuple

HL-LHC is projected to increase ROOT data at least tenfold

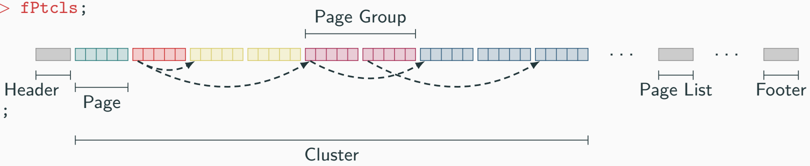
- **HW landscape changing:** NVMe devices, architectural heterogeneity, parallelism, distributed facilities...
- TTree is not designed to accomodate this leap $\longrightarrow$ enter **RNTuple**

RNTuple goals:

- $\times$2-5 better single-core performance
- 1 GB/s/core sustained end-to-end throughput
- 25% smaller files
- Robust interfaces and systematic use of exceptions
- ...and more: check yesterday's RNTuple talks by Jakob [1], Florine [2]

RNTuple Organization & On-Disk Layout

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



- **Page:** array of values of a fundamental type. Size $O(tens\ of\ KiB)$
- **Cluster:** 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

- **Metadata:** header, page lists, footer

Why Object Stores

In a highly-parallel WORM[1] setting, object stores align well with our requirements:

- Extremely scalable
- Widely deployed in cloud service providers

Contexts: **HPC** (Intel DAOS) $\neq$ **Cloud** (Amazon S3, Microsoft Azure, Google Cloud)

Downsides

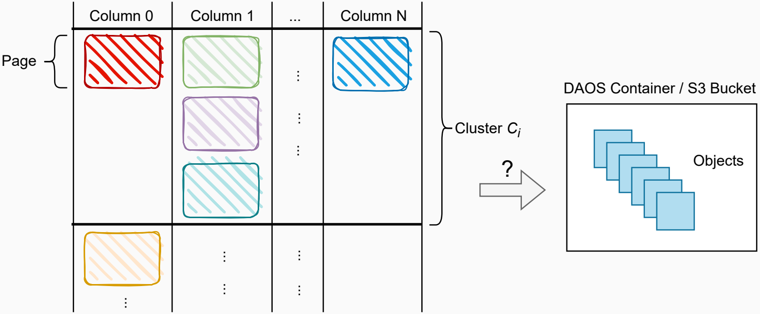
- No hierarchical namespaces
- Not based on POSIX I/O API
- No storage of other ROOT classes: histograms, etc.

[1] "Write Once, Read Many"

Challenge: Mapping RNTuple → Object Stores

Blow up RNTuple the right way

- Data granularity: *page*, *page group*, *cluster*[a]



Factors to consider

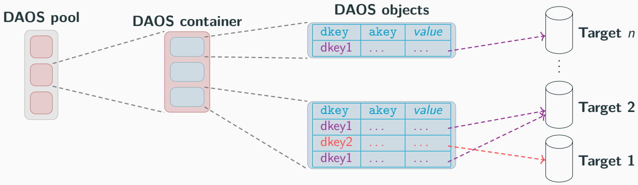
- Analysis pattern
- Throughput, latency
- Cost per request
- Memory consumption

[a] dependent on byte ranges support

Storing RNTuple data in DAOS

DAOS

- Fault-tolerant object store for HPC
  - Low-latency, high-bandwidth, high IOPS
- Foundation of the Intel exascale storage stack
- Used in 44% of the top 25 systems in IO500[2] (e.g., ANL Aurora)



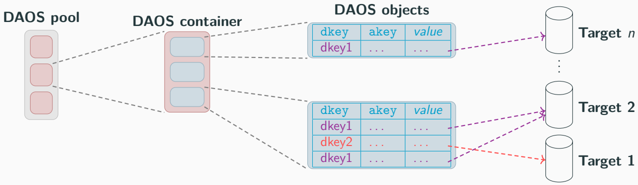
- Locality feature: $\langle object, dkey \rangle \rightarrow$ target

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

[2] https://io500.org/

DAOS

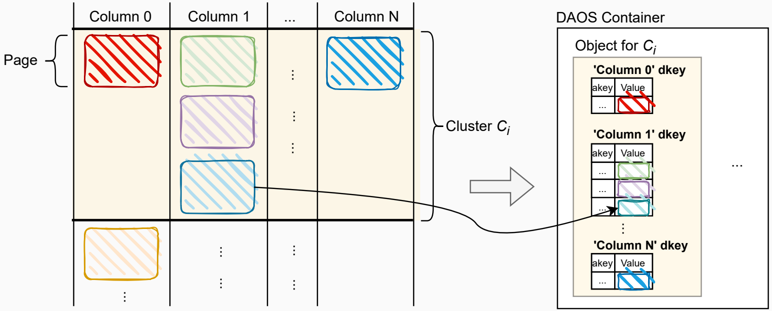
- Fault-tolerant object store for HPC
  - Low-latency, high-bandwidth, high IOPS
- Foundation of the Intel exascale storage stack
- Used in 44% of the top 25 systems in IO500[2] (e.g., ANL Aurora)



- Locality feature: $\langle \mathit{object}, \mathit{dkey} \rangle \to$ target

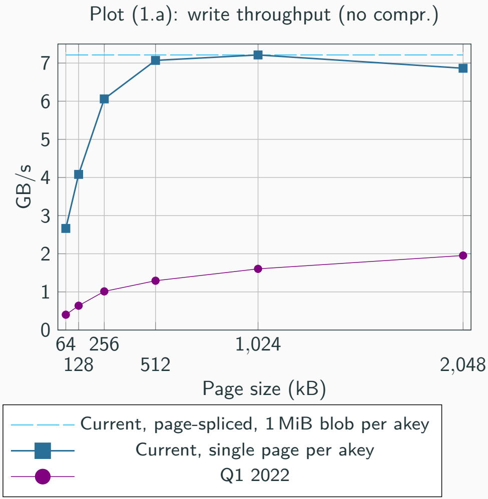
```
auto ntuple = RNTupleReader::Open("DecayTree",
    "daos://my-pool/my-container");
```

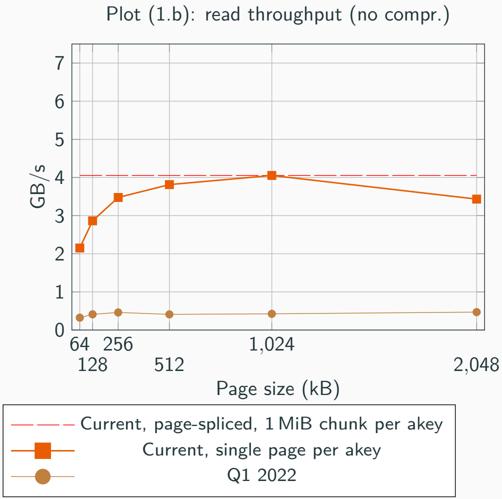
[2] https://io500.org/



Page splicing: empirically optimal at 1 MiB blobs (throughput × granularity)

DAOS backend evaluation: LHCb





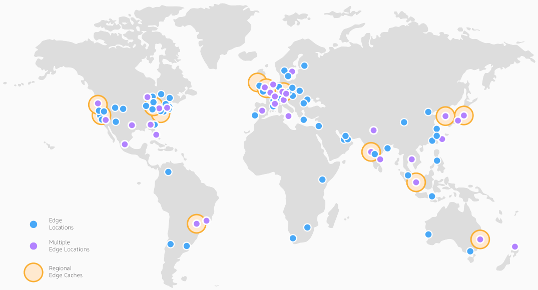
Storing RNTuple data in S3

Amazon S3

Industry's de facto standard

Key differences to DAOS:

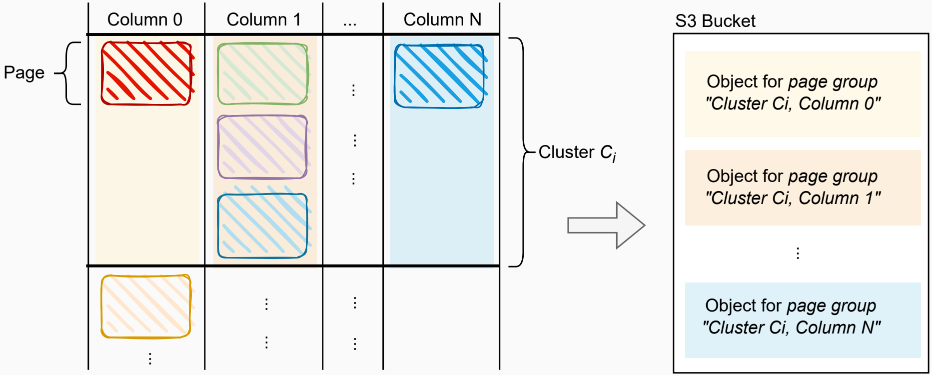
- Flat namespace: just buckets with objects
- Worldwide server distribution across "regions" and "edges"
- Potential interop with research computing infrastructures, e.g., WLCG



AWS edge locations and regional caches. Amazon.

```
auto ntuple = RNTupleReader::Open("DecayTree",
    "s3://server-region/my-bucket");
```

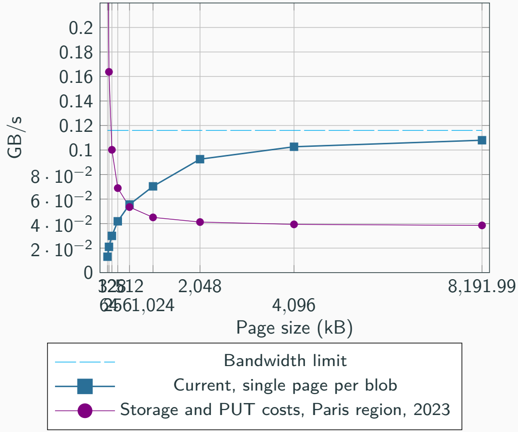
RNTuple ↔ S3



HTTP client (Davix): bigger blobs to counteract latency over network

S3 backend evaluation: LHCb

Plot (1.a): write throughput (no compr.)



Plot (1.b): read throughput (no compr.)

On data migration

RNTuple Migrator

RNTuple performance is very sensitive to its I/O parameters → see Dante's poster

- As seen, even between two object stores!
- Moving data between storage systems requires reshaping for efficient access

Implemented

Fast cloning (like `TTree`'s `CloneTree`)

- Avoids page decompression unless necessary

In development

Reshaping ntuples

- Compression algorithm, page size, cluster size, field selection

Summary

In conclusion

- RNTuple is set to be production-ready in 2024, leveraging **HPC** and **cloud object stores** for Run 3 / 4 analyses
    - **Native**, **mature DAOS backend** with 8+ GB/s writes, 4+ GB/s reads single-node
    - **Native**, **experimental S3 backend**
- Efficient data migration across storage systems, ntuple reshaping (WIP)

Next steps

- Optimize S3 backend based on first results
    - Leverage AWS C++ SDK with Davix as compatibility fallback
- Expand and improve RNTuple Migrator

Thanks!

CHEP 2023

Thanks!