Exploring Intel DAOS for HEP Data with ROOT RNTuple

Javier López-Gómez – **CERN**
<javier.lopez.gomez@cern.ch>

Presenter: Jakob Blomer

CERN openlab technical workshop, 9 March 2021

ROOT project,
EP-SFT, CERN

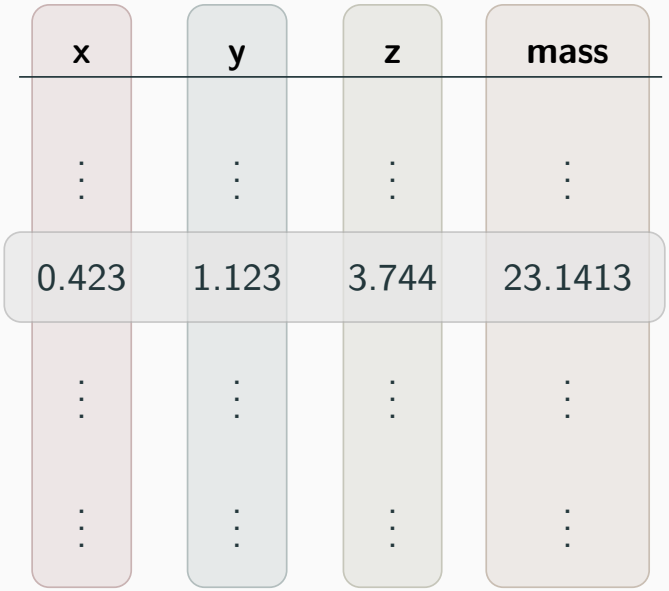
http://root.cern/

Contents

Introduction

TTree and RNTuple

- **1+ EB** of HEP data stored in TTree ROOT files.
- We gained 25+ years of experience with TTree. RNTuple is the R&D to evolve the HEP I/O layer aiming at optimal utilization of modern, fast storage hardware.
- We target a sustained throughput of 10 GB/s per node (compressed data to histograms)
- **Object stores are first-class citizens in RNTuple.**



Storage-backed, possibly nested arrays

RNTuple architecture

Data set iteration

Looping over events (rows) for reading/writing

`RNTupleView, RNTupleReader/Writer`

Logical layer / C++ objects

Mapping of C++ types onto columns, e.g.
`std::vector<float>` $\mapsto$ index column and a value column

`RField, RNTupleModel, REntry`

Primitives layer / simple types

"Columns" containing elements of fundamental types (`float`, `int`, ...) grouped into (compressed) pages and clusters

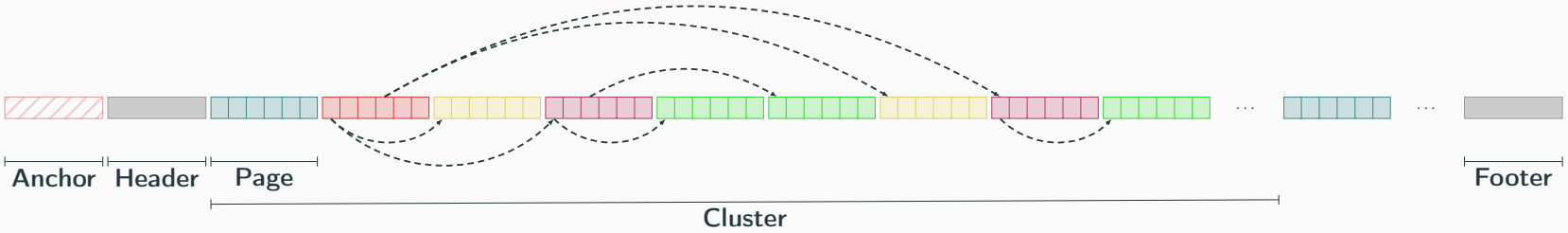
`RColumn, RPage, ...`

Storage layer / byte ranges

POSIX files, object stores, ...

`RPageStorage, RCluster, ...`

File backend: on-disk format




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

Anchor: specifies the offset and size of the header and footer sections.

Header: schema information.

Footer: location of pages and clusters.

Pages: memory-mappable fundamental types (possibly packed, compressed) — typically in the order of tens of KB.

Page group: pages in a given cluster that contain instances of the same data member.

Cluster: All data belonging to a certain event range — typically tens of MB.

RNTuple DAOS backend

libdaos C++ interface classes

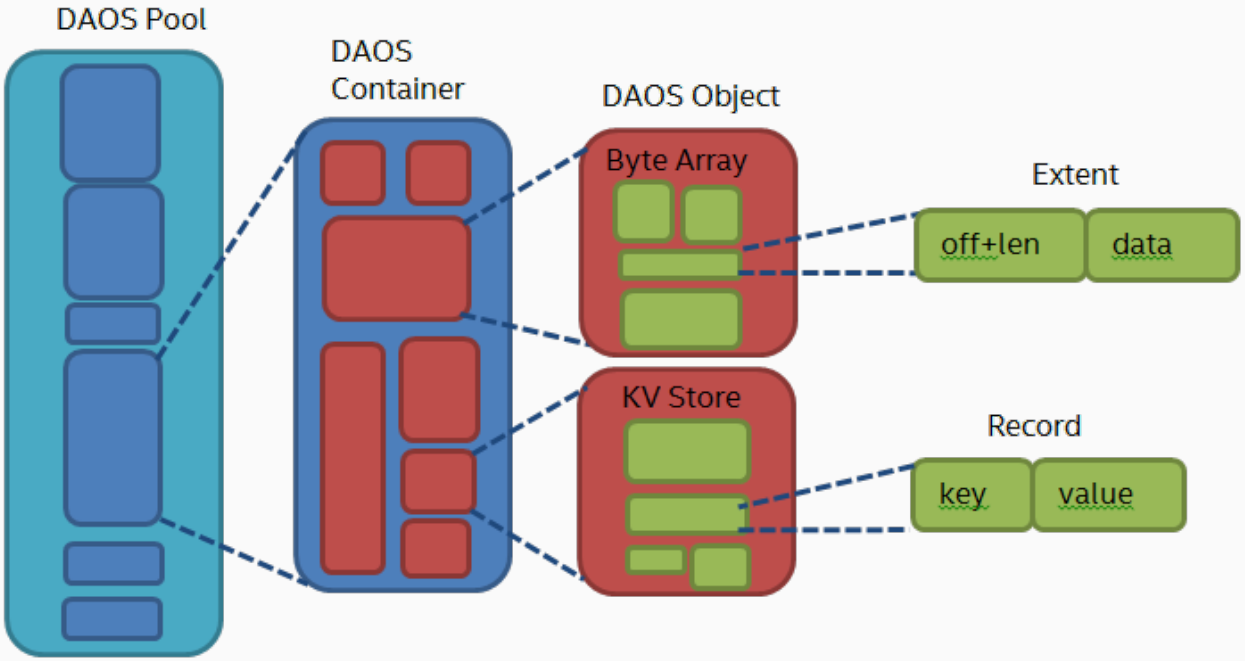
To simplify resource management, we wrote C++ wrappers for part of the libdaos functionality.

```
auto pool = std::make_shared<RDaosPool>(
    "e6f8e503-e409-4b08-8eeb-7e4d77cce6bb", "1");
RDaosContainer cont(pool, "b4f6d9fc-e081-41d4-91ae-41adf800b537");

std::string s("foo␣bar␣baz");
cont.WriteObject(daos_obj_id_t{0xcafe4a11deadbeef, 0}, s.data(), s.
    size(), /*dkey =*/ 0, /*akey =*/ 0);
```

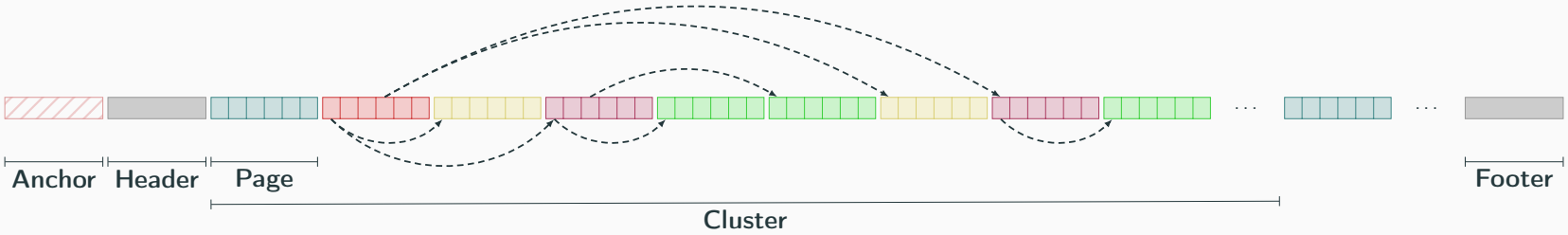
We also added a mock implementation of libdaos so that our code can be unit tested

Intel DAOS: pools, containers and objects



- **Object:** a Key–Value store with locality. Can be accessed through 3 different APIs: *multi-level key-array (native)*, *Key–Value*, and *Array*.
  - The key is split into **dkey** (distribution key) and **akey** (attribute key). $dkey_i \mapsto target_k$.
- **Object class:** determines redundancy (replication/erasure code).

DAOS backend: mapping things to objects




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

- Each cluster is stored in a separate object (OID assigned sequentially). There would be a natural mapping of clusters to compute nodes.
- Pages stored as part of the corresponding cluster object using different **dkey**s.
- **Header**, **Footer**, and **Anchor** are stored in three different objects with reserved OIDs.

Usage: RNTuple/file vs. RNTuple/DAOS

From the user's perspective

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

auto viewH1IsMuon = ntuple->GetView<int>("H1_isMuon");
auto viewH2IsMuon = ntuple->GetView<int>("H2_isMuon");
auto viewH3IsMuon = ntuple->GetView<int>("H3_isMuon");
```

- Only requires replacing the filesystem path by a DAOS URI.
- UUIDs are not meaninful to users (puts more responsibility to the data management system).

Usage: RNTuple/file vs. RNTuple/DAOS

From the user's perspective

```
auto ntuple = RNTupleReader::Open("DecayNTuple",
             "daos://e6f8e503-e409-4b08-8eeb-7e4d77cce6bb:1/b4f6d9fc...");

auto viewH1IsMuon = ntuple->GetView<int>("H1_isMuon");
auto viewH2IsMuon = ntuple->GetView<int>("H2_isMuon");
auto viewH3IsMuon = ntuple->GetView<int>("H3_isMuon");
```

- Only requires replacing the filesystem path by a DAOS URI.
- UUIDs are not meaninful to users (puts more responsibility to the data management system).

Preliminary evaluation

Test environment

Our evaluation ran on CERN OpenLab DAOS test machines:

- 3 DAOS servers, 1 DAOS head node.
- interconnected by an Omni-Path Edge Switch 100 Series | 24 ports.

System specifications	
CPU	Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz
CPU per node	24 cores/socket, 2 sockets, 2 threads/core (HT enabled)
Core frequency	Base: 1.0 GHz Range: 1.0GHz - 3.9GHz
Numa nodes	node0: 0-23,48-71 node1: 24-47,72-95
System Memory	12x 32GB DDR4 rank DIMMs
Optane DCPMM	12x 128GB DDR4 rank DIMMs
Optane FW version	01.02.00.5395
BIOS	version: SE5C620.86B.02.01.0011.032620200659 date: 03/26/2020
Storage	4x 1 TB NVMe INTEL SSDPE2KX010T8
HFI	1x Intel Corporation Omni-Path HFI Silicon 100 Series.
HFI Firmware	Termal Management Module: `10.9.0.0.208`; Driver: `1.9.2.0.0`

Figure 1: Server nodes HW (`olcsl-*`)

System specifications	
CPU	Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz
CPU per node	24 cores/socket, 2 sockets, 2 threads/core (HT enabled)
Core frequency	Base: 1.0 GHz Range: 1.0GHz - 3.9GHz
Numa nodes	node0: 0-23,48-71 node1: 24-47,72-95
System Memory	12x 16GB DDR4 rank DIMMs
BIOS	version: SE5C620.86B.02.01.0011.032620200659 date: 03/26/2020
HFI	1x Intel Corporation Omni-Path HFI Silicon 100 Series.
HFI Firmware	Termal Management Module: `10.9.0.0.208`; Driver: `1.9.2.0.0`

Figure 2: Client node HW (`olsky-03`)

RNTuple vs dfuse compatiblity layer (zstd compressed)

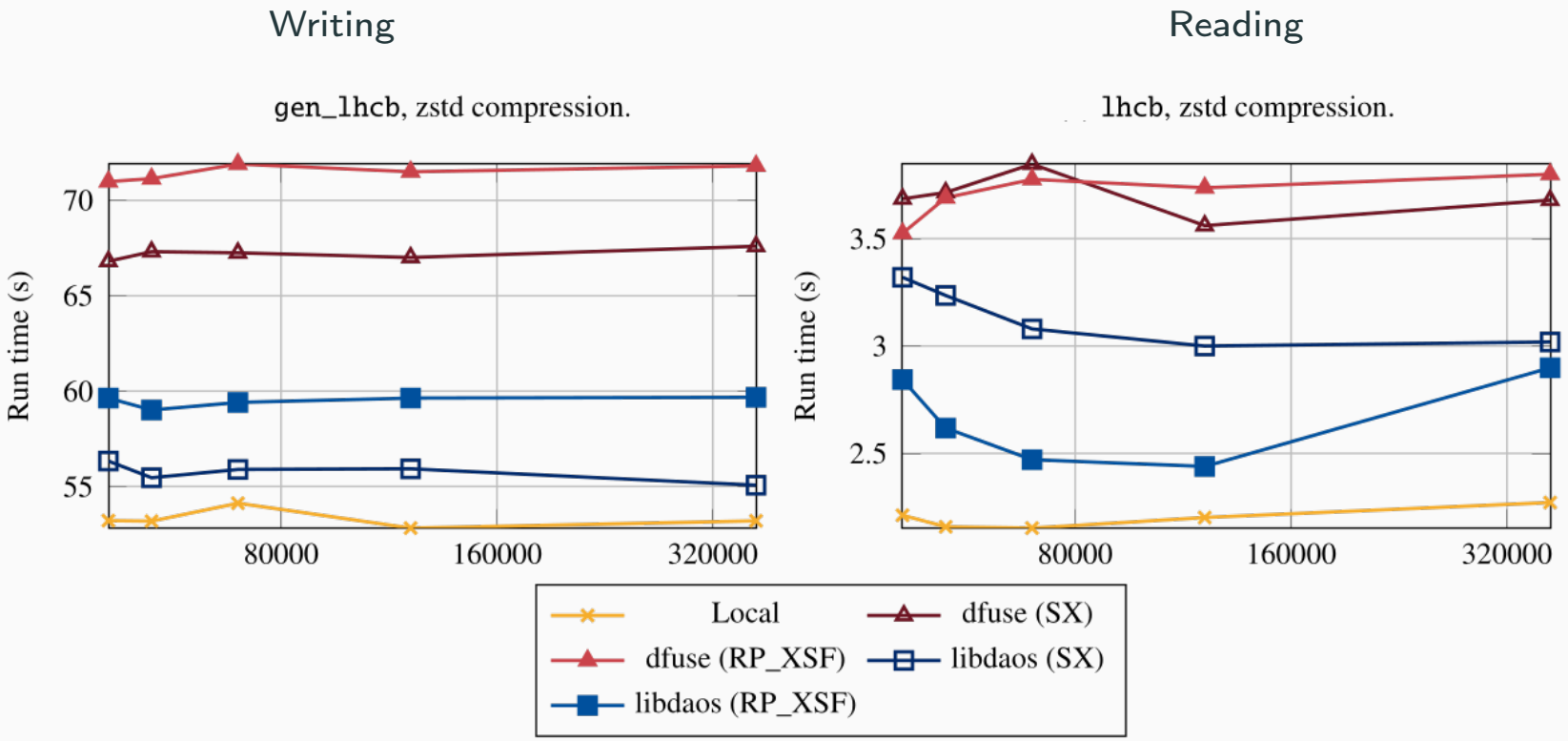


Figure 3: RNTuple benchmark from zstd compressed data to histograms on LHCb data (ofi+PSM2). Best raw read throughput: 2.3 GB/s.
Sources: `https://github.com/jblomer/iotools`

- Performance impact of RNTuple page size and DAOS object replication factor

Summary

Summary

- RNTuple architecture decouples storage from serialization/representation. Object stores are first-class.
- First prototype implementation of an Intel DAOS backend!
- The RNTUple native DAOS backend is substentially faster than the dfuse compatibility layer for reading and writing.
- We aim at fully utilizing the link layer – we think that a careful tuning of the RNTuple page size and the queue depth of RNTuple async I/O access are key to get there

Exploring Intel DAOS for HEP Data with ROOT RNTuple

Javier López-Gómez – **CERN**
<javier.lopez.gomez@cern.ch>

Presenter: Jakob Blomer

CERN openlab technical workshop, 9 March 2021

ROOT project,
EP-SFT, CERN

http://root.cern/