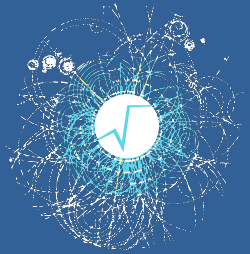


RNTupleInspector

A storage information utility for RNTuple

Florine de Geus[1,2] Jakob Blomer[1] Philippe Canal[3] Vincenzo Eduardo Padulano[1]

[1]CERN [2]University of Twente [3]Fermi National Accelerator Laboratory

https://root.cern

ACAT 2024

Stony Brook, Long Island NY, USA

March 11, 2024

Introduction

RNTuple is ROOT's next-generation columnar I/O subsystem, based on 25+ years of experience with **TTree**, aiming at:

1. Higher storage space efficiency and lower CPU usage
2. Robust and modern interfaces
3. Efficient use of modern hardware and object stores

Recently, it has become mature enough for integration and evaluation in experiment frameworks ▸ previous talk

Adoption of RNTuple

Successful adoption of RNTuple requires a solid understanding of its behaviour, both in a **runtime** and **static** context

Runtime behaviour

- → CPU and I/O performance with aptly chosen benchmarks (e.g. sample analyses, stress tests)
- → Benchmarks are typically experiment/analysis specific, but we can provide tools for measuring and reporting throughput

Static behaviour

- → Impact of different I/O parameters on storage space efficiency
- → Benchmarks can be largely general, with potentially some EDM-specific measurements

The RNTupleInspector

The **RNTupleInspector** is a utility interface for **static behaviour measurements**

Its primary goal is to help **understand** and **guide** the way data is stored with RNTuple

→ For us as RNTuple developers, but more importantly for experiment framework developers

This is achieved by providing methods for **elementary storage metrics** as well as a number of convenience methods for **combined storage information**

The provided information is **unambiguous** and **consistent** with the RNTuple specification

A quick reminder on the RNTuple format

Fields represent C++ PODs, classes or collections thereof

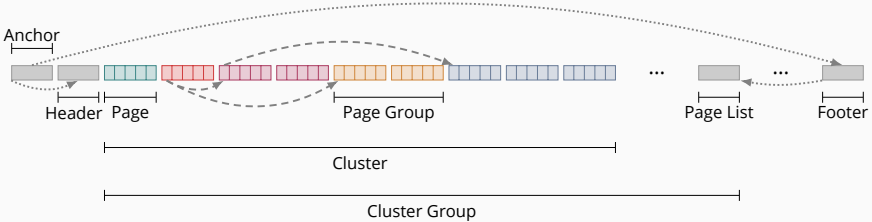
On disk, fields are stored as **columns** of fundamental types

Columns are compressed into **pages**

A set of pages covering a given entry range is a **cluster**

Clusters are bundled into **cluster groups**

```
struct Event {
  int id;
  vector<Particle> particles;
};
struct Particle {
  vector<int> trackerIds;
  float energy;
};
```



Key features of RNTupleInspector

Elementary metrics for getting the compression information and (un)compressed size of:

- Complete RNTuples
- Fields and subfields
- Columns

Similar to, e.g., `TTree::GetZipBytes()`, `TBranch::GetTotBytes()`

Combined storage information providing insights at a glance:

- Aggregated information per column type
- Distribution of page sizes for one or multiple columns
- Visualization of the RNTuple on-disk layout

The RNTupleInspector in action

For the following examples, we use one of the **W+jets** NanoAOD data samples provided for the Analysis Grand Challenge ▸ next talk

```
using ROOT::Experimental::RNTupleInspector;
auto inspector = RNTupleInspector::Create("Events", "cmsopendata2015_wjets_20547.root");
```

Elementary RNTuple metrics (1)

We can get information for the **whole RNTuple**

```
inspector->GetCompressionSettings();
```

→ (int) 505

```
inspector->GetUncompressedSize();
```

→ (unsigned long) 3972293720

```
inspector->GetFieldCountByType("(ROOT::VecOps::RVec|std::vector)<float>");
```

→ (unsigned long) 217

Elementary RNTuple metrics (2)

We can get information for a specific **field** (including its subfields)

```
auto fieldInspector = inspector->GetFieldTreeInspector("Muon_pt" /* fieldName */);
```

The RNTupleInspector provides access to the associated **descriptor** to verify its actual type:

```
fieldInspector.GetDescriptor().GetTypeName();
```

→ `(std::string) ROOT::VecOps::RVec<float>`

Elementary RNTuple metrics (3)

We can get information for a specific **column**

```
auto columnInspector =
    inspector->GetColumnInspector(fieldInspector.GetPhysicalColumnIds()[0]);
```

```
columnInspector.GetType();
```

→ (ROOT::Experimental::EColumnType) ROOT::Experimental::EColumnType::kSplitReal32

```
columnInspector.GetNPages();
```

→ (unsigned long) 20

Combined storage information: information per column type

We can get aggregated information for each **column type** present in the inspected RNTuple

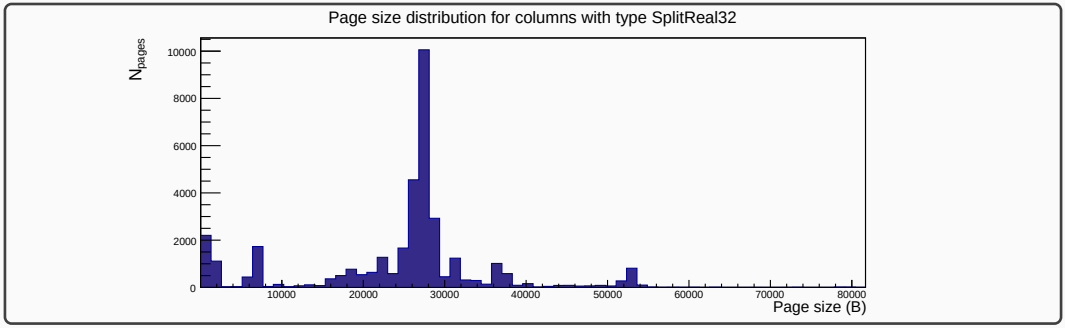
```
inspector->PrintColumnTypeInfo();
```

column type	count	# elements	compressed bytes	uncompressed bytes
Bit	496	592881545	10492765	592881545
UInt8	43	32241155	5513544	32241155
SplitIndex64	22	27479672	4615823	219837376
SplitReal32	300	566436211	873832424	2265744844
SplitUInt64	1	1249076	1107819	9992608
SplitInt32	83	210400896	62534108	841603584
SplitUInt32	2	2498152	31008	9992608

Combined storage info: page size distribution

We can get a histogram showing the **compressed** page size distribution for a **single column**, **column type** or **collection of columns**

```
auto pageSizes = inspector->GetPageSizeDistribution(EColumnType::kSplitReal32);
pageSizes->Draw("PFC");
```

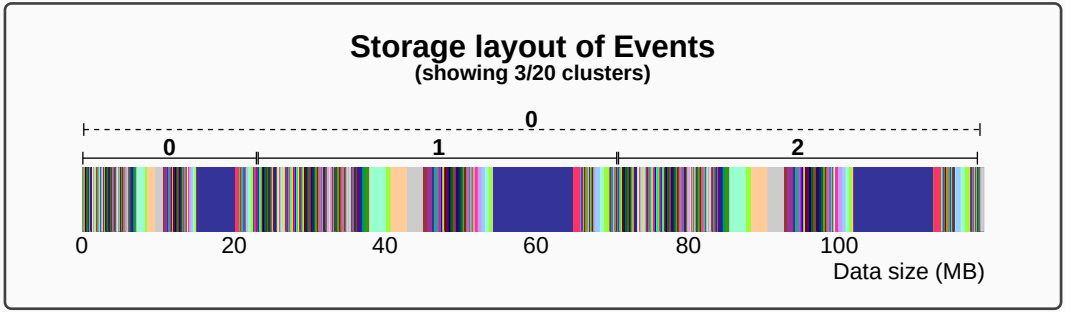


Combined storage information: on-disk layout visualization

We can use the RNTupleInspector to **visualize** the **on-disk layout** of an RNTuple

```
inspector->DrawStorageLayout("wjets.pdf" /* outputPath */, 3 /* nClusters */);
```

Let's first consider the dataset created with the default RNTuple write options:

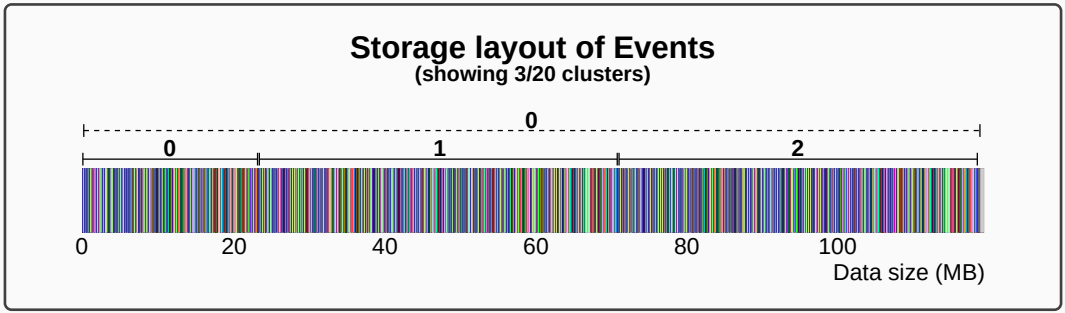


Combined storage information: on-disk layout visualization

We can use the RNTupleInspector to **visualize** the **on-disk layout** of an RNTuple

```
inspector->DrawStorageLayout("wjets.pdf" /* outputPath */, 3 /* nClusters */);
```

The same data set, but created with buffered writing disabled:



Conclusions and outlook

The RNTupleInspector is created to help make the **successful transition** to RNTuple

It provides features to get **direct insights**, in the form of elementary metrics and combined storage information

We are actively developing the RNTupleInspector, and **suggestions for new, useful features** to add are invaluable to us!

Find the full, up-to-date documentation of the RNTupleInspector here