RNTuple performance: Status and Outlook

Javier Lopez-Gomez[1], **Jakob Blomer**[1]

[1] EP-SFT, CERN, Geneva, Switzerland

E-mail: {javier.lopez.gomez,jblomer}@cern.ch

Abstract. Upcoming HEP experiments, e.g. at the HL-LHC, are expected to increase the volume of generated data by at least one order of magnitude. In order to retain the ability to analyze the influx of data, full exploitation of modern storage hardware and systems, such as low-latency high-bandwidth NVMe devices and distributed object stores, becomes critical.

To this end, the ROOT RNTuple I/O subsystem has been designed to address performance bottlenecks and shortcomings of ROOT's current state of the art TTree I/O subsystem. RNTuple provides a backwards-incompatible redesign of the TTree binary format and access API that evolves the ROOT event data I/O for the challenges of the upcoming decades. It focuses on a compact data format, on performance engineering for modern storage hardware, for instance through making parallel and asynchronous I/O calls by default, and on robust interfaces that are easy to use correctly.

In this contribution, we evaluate the RNTuple performance for typical HEP analysis tasks. We compare the throughput delivered by RNTuple to popular I/O libraries outside HEP, such as HDF5 and Apache Parquet. We demonstrate the advantages of RNTuple for HEP analysis workflows and provide an outlook on the road to its use in production.

1. Introduction

HEP storage systems are generally tuned for write-once-read-many columnar access. Since its inception, the ROOT project[4] supports the columnar storage of arbitrary C++ types and collections through TTree. However, the expected increase in the amount of experiments data that needs to be processed and the fact that TTree was not designed to make optimized use of modern hardware and storage systems, called for a new, modernized re-engineering of TTree.

RNTuple is the new, experimental, backward-incompatible ROOT columnar I/O subsystem targeting high performance, reliability, and easy-to-use robust interfaces. Despite RNTuple still being under development, at this point it is feature-complete enough to carry out an evaluation.

In this paper we contribute with the following:

- A performance evaluation of TTree, RNTuple, and other well-known storage alternatives outside HEP: Apache Parquet and HDF5. Compared to a previous publication[1], we evaluate RNTuple focusing on different storage devices and a dataset with nested collections.
- A feature comparison and perspectives of using RNTuple in production.

2. ROOT's RNTuple overview

The design of RNTuple[3] comprises four layers: *(i)* event iteration layer, that offers a convenient interface for looping over events; *(ii)* logical layer, that maps complex C++ types onto columns; *(iii)* primitives layer, which groups ranges of elements of a fundamental type into pages; and

(iv) storage layer, that is responsible for the I/O of pages, clusters, and required metadata. This design makes it simple to support new data types or storage backends.

A page contains a certain range of values for a given column, whereas a cluster contains pages for a specific row range. Metadata includes a header that describes the data schema and a footer that contains the location of clusters and pages, among other information. Header/footer locations and their sizes are included in an anchor object. Figure 1 shows a simplified example of the on-disk layout.

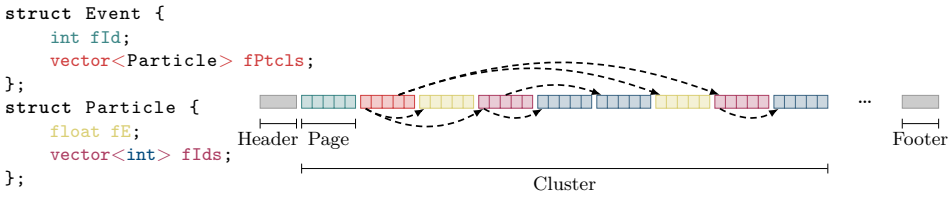

Figure 1. RNTuple on-disk format. Pages store values for a specific data member (note the color coding).

3. Evaluation

In this section, we evaluate RNTuple w.r.t. TTree and other well-known I/O libraries outside HENP: HDF5 and Apache Arrow/Parquet. Specifically, Section 3.1 compares the support level of important HENP I/O layer features. A quantitative experimental evaluation is provided in Section 3.2.

To make a fair comparison, all our test programs were written in C++ and parameters such as row group size, page size, and compression algorithm and level were set to match in all cases, where permitted. For Apache Parquet, we leveraged the Parquet-Arrow API permitting the convenient use of nested data structures and lists. For HDF5, however, the columnar storage of heterogeneous data types or nested collections thereof is not a trivial problem. Parts of our test code bridge this gap and allows switching between alternate data layouts simply by changing a C++ template parameter. Specifically, this layer provides the following layouts:

- **Row-wise:** uses HDF5 compound types and variable-length types to represent nested structures and collections, respectively. This layout creates a single dataset whose type is the outer-most data structure and its dataspace dimension is $1 \times N$.
- **Column-wise:** emulated columnar layout that uses one HDF5 dataset per column of a fundamental type, as described in [7]. Collections are translated to a HDF5 group and one additional index column.

In any case, HDF5 datasets are chunked. Given that chunks are individually accessed and (un)compressed, their size will be equal to the RNTuple or Parquet page size in all of our tests. The chunk cache size is set to the default size of a RNTuple cluster.

3.1. Qualitative evaluation

In Table 1, we compare the level of support of different must-have features in a HENP I/O layer. Compression is, in general, supported by all the analyzed storage formats; however, the native support for different algorithms greatly varies, e.g. HDF5 only supports zlib and szip. Vertical and horizontal data combinations refers to extending the dataset with new entries or columns, respectively. To the best of our knowledge, Apache Arrow allows reading rows from many files, but it is unclear whether new columns can be made available. Similarly, columnar access to multilevel nested structures and collections in HDF5 is unclear. Schema evolution, i.e. handling changes in the data schema such as adding, removing or changing the type of a column, is only

	HDF5	Parquet	TTree	RNTuple
Transparent compression	(1)	●	●	●
Columnar access	(2)	●	●	●
Merging without uncompressing data			●	○
Vertical/horizontal data combinations		/	●	○
C++ and Python support	/	●	●	●
Support for structs/nested collections	?	●	●	●
Architecture-independent encoding	●	●	●	●
Schema evolution			●	○
Support for application-defined metadata	●			○
Fully checksummed	●	●		○
Multi-threading friendly	○	●	●	●
Native object-store support	●	●		●
XRootD support			●	●
Automatic schema creation from C++ classes			●	●
On-demand schema extension (backfilling)			●	○
Split encoding / delta encoding		●		○
Variable-length floats (min, max, bit size)			●	○

● Supported ○ Planned / Under development / Partial / Incomplete ? Unclear
(1) Only for chunked datasets (2) Via emulated columnar

Table 1. Comparison of features available in TTree, RNTuple, HDF5, and Apache Arrow/Parquet.

supported in TTree; preliminary support for this feature in RNTuple is foreseen for Q2 2022. Native support for object stores is available in HDF5, RNTuple (DAOS), and Apache Arrow (S3). Finally, split encoding typically improves the compression ratio by reordering bytes in integer/floating point numbers, so that the $n-th$ byte of each value is contiguous in memory.

3.2. Experimental evaluation

In this section, we provide experimental measurements of the analysis throughput, total amount of bytes read, and file size for TTree, RNTuple, HDF5 (both row-wise and column-wise) and Apache Parquet in a variety of situations. The hardware and software environment, datasets used, and test cases are described in the following.

Hardware and software environment. Our benchmarks ran on a single node based on 1×AMD EPYC 7702P 64-Core processor running at 2 GHz, and 128 GB DDR4 RAM. SMT was enabled, although disabling it yielded similar results for the workload in our tests. This machine is also equipped with a Samsung PM173X NVMe SSD, and a TOSHIBA MG07ACA1 SATA hard disk drive. A ext4 filesystem resides on each drive using a 4 KB block size and default mount options. CephFS was used as the network filesystem for tests that operate over a network share.

The software environment is based on CentOS Linux 8.3 (kernel 5.15.1), Apache Arrow 5.0.0, HDF5 1.10.5, and ROOT git revision `5001281762` built with g++ 8.4.1.

Test cases. The experiments ran in the evaluation have used the following datasets as input:

- **LHCb Run 1 Open Data B2HHH (B meson decays to three hadrons).** No nested collections, containing 8.5 M events, 26 branches[1]. The compressed file size is 1.1 GB.
- **CMS Open Data Higgs → 4 leptons MC.** NanoAOD-like[6] format, 300 k events, 84 branches. This dataset was concatenated 16 times so as to make a larger file of ∼ 2.1 GB.

We carried out two different experiments[2]: *(i)* running a simple analysis program over the LHCb dataset that generates the B mass spectrum histogram and measures the end-to-end, i.e. from storage to histogram, analysis throughput in uncompressed MB/s; and *(ii)* measurement

[1] In TTree, the term "branch" refers to a column; usually, both terms can be used interchangeably.

of the read rate in uncompressed MB/s for both the LHCb and the CMS datasets, retrieving all entries in 10 selected branches.

The original files were in TTree format. In a first stage, a third program is used to generate the equivalent files for each of the other formats (HDF5 row-wise, HDF5 column-wise, and Apache Parquet). All files taken as input by the benchmark programs use zstd with compression level 5, except for HDF5 that uses zlib with compression level 3 given that HDF5 lacks native support for zstd. RNTuple and Apache Parquet multi-threaded I/O and (un)compression was enabled in all cases. RNTuple benchmarks use a cluster prefetch value of 5.

Each experiment and storage format combination is run four times, differing on the location of the input file: CephFS, SSD, HDD. To ensure that data is forcibly read from the underlying storage, the Linux page cache is cleared prior to running each test. A fourth iteration uses the same input file as in the HDD case but does not clear the page cache before the test is executed. Additionally, we measured the total file size after conversion and the raw bytes read by each experiment–format pair, as reported by the vmtouch[5] tool. In all the plots shown below, the data point and error bars (where applicable) refer to the average and minimum/maximum values measured over 10 executions, respectively.

In a first stage, we measured the analysis throughput for all combinations of format and storage device for the LHCb dataset. The analysis program requires reading 18 out of 26 columns. As can be seen in Figure 2, RNTuple speedup over Apache Parquet is between ×1.4 and ×2.2 depending on the scenario. Also note that the performance of HDF5 is severely affected by the lack of multi-threaded I/O and decompression.

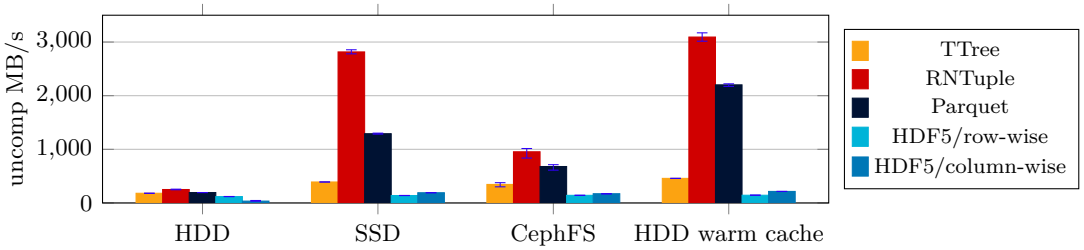

Figure 2. LHCb B2HHH analysis throughput (18/26 branches; compressed).

Whereas the previous plots give an idea of the performance of each candidate, these results do not represent the behavior in case of accessing collections. In a second experiment, we measured the read rate in uncompressed MB/s for 10 columns of both the LHCb and CMS dataset. This test allows us to compare the performance differences for collections. As can be seen in Figure 3, for the LHCb dataset, RNTuple's worst result is comparable to Apache Parquet. In all the other tests, RNTuple outperforms other alternatives. For the CMS dataset, RNTuple worst case achieves at least the same result as Parquet. Differences between both plots is explained by the use of different column types (`double` w.r.t. `float`) and the presence of collections.

Finally, we measured the compressed file size and total bytes transferred during the last test for the CMS dataset (see Figure 4). As can be seen, RNTuple provides the smallest file while the amount of bytes read is about the same as Parquet. It is worth noting that HDF5 row-wise does not read the whole file if the compound type definition provided at run-time misses some members w.r.t. the committed (on-file) type; however, the throughput in this case is extremely low (< 1 MB/s; see Figure 3.b).

4. Conclusion and future work

RNTuple is able to deliver the highest throughput among the analyzed alternatives in all of our tests thanks to its performance-tuned implementation and parallel I/O scheduling and decompression. The latest developments are available in the `ROOT::Experimental` namespace. The feature roadmap aims at complete support for the HENP event data storage use case.

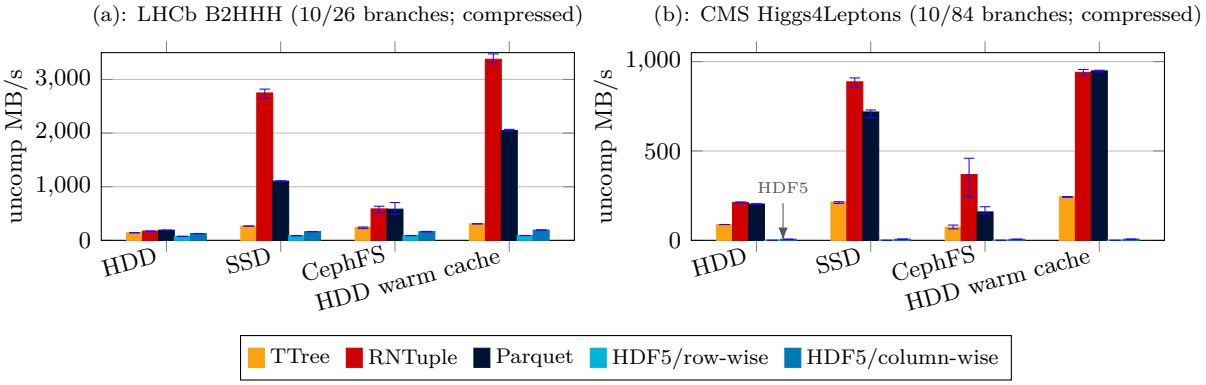

Figure 3. Read rate (uncompressed MB/s; 10 selected branches). Note the low (< 10 MB/s) transfer rate for HDF5 in the CMS case.

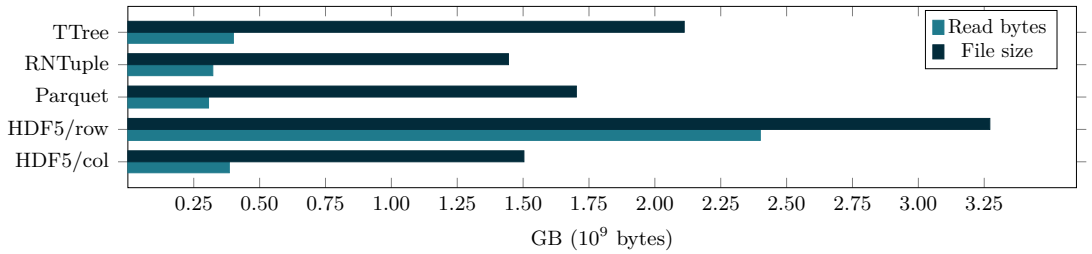

Figure 4. File size and raw bytes read for the CMS dataset (10/84 branches).

RNTuple is expected to become production grade in 2024. A number of important features are scheduled for 2022: schema evolution, on-demand addition of new columns to the model, complete support for vertical/horizontal data combinations, and merging without uncompressing pages, only to name a few. As a future work, we also plan to compare the performance achieved by the RNTuple and HDF5 DAOS connectors.

Acknowledgments

This work benefited from support by the CERN Strategic R&D programme on Technologies for Future Experiments (CERN-OPEN-2018-06).

References

[1] J. Blomer. "A quantitative review of data formats for HEP analyses". In: *J. Phys. Conf. Ser.* 1085.3 (2018), p. 032020. DOI: 10.1088/1742-6596/1085/3/032020.

[2] J. Blomer et al. *ROOT RNTuple Virtual Probe Station*. Accessed: 2022-02-10. URL: https://github.com/jblomer/iotools/tree/master/compare.

[3] Blomer, Jakob et al. "Evolution of the ROOT Tree I/O". In: *EPJ Web Conf.* 245 (2020), p. 02030. DOI: 10.1051/epjconf/202024502030.

[4] Rene Brun and Fons Rademakers. "ROOT—An object oriented data analysis framework". In: *Nuclear instruments and methods in physics research section A: accelerators, spectrometers, detectors and associated equipment* 389.1-2 (1997), pp. 81–86.

[5] Hoyte Doug. *vmtouch: the Virtual Memory Toucher*. 2012. URL: https://hoytech.com/vmtouch/.

[6] Rizzi, Andrea, Petrucciani, Giovanni, and Peruzzi, Marco. "A further reduction in CMS event data for analysis: the NANOAOD format". In: *EPJ Web Conf.* 214 (2019), p. 06021.

[7] S. Sehrish et al. "Python and HPC for High Energy Physics Data Analyses". In: *7th Workshop on Python for High-Performance and Scientific Computing*. 2017.