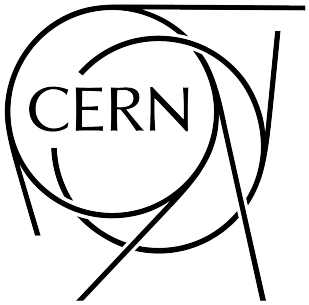
Exploring RNTuple Huge-File Support

Anna Helena Harms

Summer Student Report submitted
to the CERN Summer Student Programme



Supervisors: Jakob Blomer, Jonas Hahnfeld

August 21, 2026

Keywords: ROOT; RNTuple; C++

Contents

1 Introduction

1.1 ROOT and RNTuple

ROOT is an open source framework for processing and analyzing scientific data. It is prolific in both high energy and nuclear physics and is primarily developed at CERN [2]. Within ROOT, RNTuple is the new columnar data format and I/O subsystem [1]. It is the successor to TTree, with improvments in the areas of compression, throughput, compatability with modern storage devices and huge-file support.

As a columnar data format, an RNTuple data set is made up of columns of event properties (e.g. "energy" or "transverse momentum"), each of which will have a number of elements which are grouped into pages. Pages are the unit of addressing and compressing data blocks on disk and are typically in the range of tens of kilobytes to megabytes. All of the columnar data that belongs to a certain event range is grouped into a cluster. A visual reference for this stucture can be seen in Figure 1.

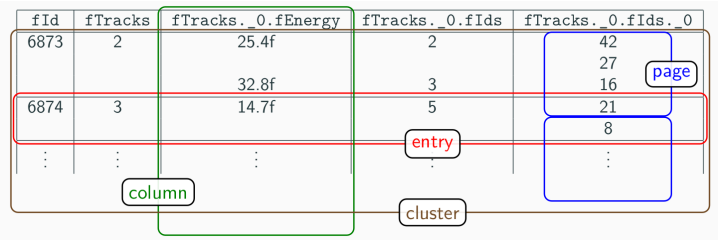


Figure 1: Diagram showing structure of the RNTuple data format. Created by Jonas Hahnfeld [3]

Though not shown in Figure 1, another component is the cluster group which is a list of consecutive clsuters for which a single block of metadata is written out.

1.2 Reading and Writing RNTuple Data

During writing, RNTuple decomposes event data into column elements and fills up pages and clusters. Before pages and clusters start to take up too much memory, they are flushed to disk. The flushed data are released from memory, but the metadata associated with the clusters remains in memory and this process repeats until all of the entries have been written to disk. This results in metadata accumulating in memory over the course of writing the data to file.

When reading RNTuple data from file, all of the metadata is loaded into memory at once. This metadata is then used to locate and load the data pages as needed.

1.3 Memory Usage and Huge-File Support

Recent experiments are increasingly using larger and larger data files. For the upcoming High Luminosity Large Hadron Collider (HL-LHC) run, these files are expected to exceed 100 GB. Within this range of huge files, the metadata of the file becomes too large to fit into memory all at once. Since RNTuple will be adopted as the standard columnar data format for all of the HL-LHC experiments, one of its design goals is to handle file sizes this large. In order to properly support huge files, piecewise streaming of metadata is proposed.

2 Benchmarking Memory Usage

2.1 Overview

Since huge-file support introduces memory constraints as a design concern during data streaming of RNTuple data, benchmarks were needed to confirm both current and improved reading and writing behaviour. As part of this project, code was custom developed for benchmarking memory usage while streaming RNTuple data between memory and disk [4]. This project focused on benchmarking the data writing process while designing and producing code that allowed for easy implementation of data reading benchmarks. This will be implemented and used by other members of the ROOT development team after the end of this project.

2.2 Code Structure

The main bash script manages and executes two ROOT macros. Once executed, it will repeatedly call read_write_record.C which writes trivial RNTuple data to file and records the memory usage to csv files during this process. Afterwards, analyze.C is called to extract and analyze data from the csv files and produce benchmarking plots.

2.3 Code Restrictions and Simplifications

In order to investigate the memory usage during data streaming, certain restrictions and simplifications were implemented during the data writing process. For example, when constructing the RNTuple data models, all of the columns were set to the basic integer type and each of the entries were filled with zeros. Then, in order to easily derive the number of pages, the page size was set to the same size as a single integer (4 bytes). This means that the number of pages is equivalent to the number of entries times the number of columns.

Another restriction was that the cluster size was fixed within the benchmarking code. In this case, it was simply set high enough (64 MB) to prevent the RNTuple writer from automatically commiting a cluster before the call to commit was manually called.

In line with this simplified scheme for cluster commits, cluster group commits were also simply called after a fixed number of entries.

2.4 Parameter Selection and Iteration Scheme

Since the number of pages is determined by the number of columns and entries, these are both given as parameters for the read_write_record.C macro. These parameter sets are iterated over in the main bash script mem_usage_vs_page_num.sh. The iteration scheme was such that the number of fields was fixed at 500 whereas the number of entries was the iteration number multiplied by a factor of 10.

The two main iteration parameters located in the mem_usage_vs_page_num.sh script. The first is upperIterLimit which determines how many different parameter sets read_write_record.C will run with. Consequently, this will also be the number of data points in the output plots. This was set to 500 in order to cover a reasonable number of points. The second is iterRepeatFactor which is the number of times read_write_record.C will be run for a given parameter set. The average results from these repetitions will produce a single data point. Hence, this parameter affects the statistical uncertainty in the output plots.

3 Current Memory Usage Benchmarks

In order to confirm the suspected pattern of memory usage during writing of RNTuple data, the benchmarking code described in Section 2 was run using version 6.41.01 of ROOT. The resulting plots are shown in Figures 2 and 3 which highlight the resident set size and virtual set size respectively. In both cases, the horizontal axis represents the number of pages which are written to file. The vertical axis represents the maximum memory usage during the process of writing RNTuple data to file, averaged over the number of runs with the same number of pages. This follows the iteration scheme discussed in Section 2.4.

Two distinct regions can be seen in Figure 2, seperated by a sharp increase around 100 pages. The first part of the plot can be explained as representing the data which is stored in memory before being written. The large increase occurs when the first cluster is written to file and each subsequent smaller increase can be linked to another cluster being written to file. The memory usage is seen to increase as the metadata associated with the clusters accumulates in memory as predicted.

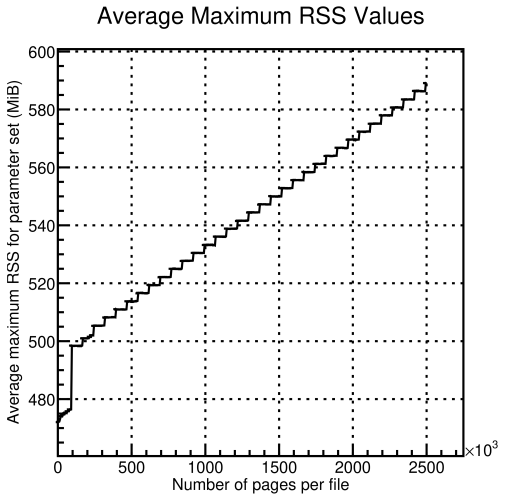


Figure 2: Shows the average maximum resident set size (RSS) versus the number of pages written to file. Output from the benchmarking code described in Section 2

A similar output plot can be can be seen in Figure 3 which outlines memory usage in terms of the virtual set size. The main observable differences are that the plot has less noise and the memory usage is generally higher overall which is again in line with expectations.

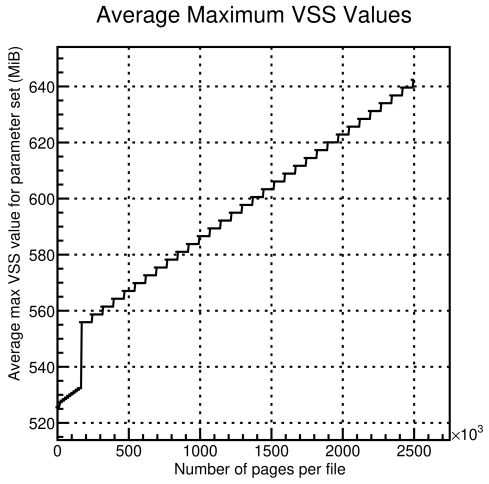


Figure 3: Shows the average maximum virtual set size (VSS) versus the number of pages written to file. Output from the benchmarking code described in Section 2

Without any mechanism for curbing the increase seen in Figures 2 and 3, it is clear that this will continue indefinitely and that when dealing with large enough data files this will exceed the memory capacity.

4 Prototyping Source Code Improvements

4.1 Changes Implemented

In order to reduce the memory usage, piecewise streaming of metadata is to be implemented. For writing data to file, this means grouping the metadata resulting from clusters into cluster groups and write them to file. This mirrors the way that data is written to file and kept track of with clusters.

Cluster groups as a concept are already implemented in the ROOT source code and the benchmarking code manually calls the function to commit these cluster groups to memory. This doesn't prevent metadata from accumulating in memory though. In order to reduce memory usage, this function had to be altered to not only commit the cluster groups but also to flush the clusters within that group from memory once they have been written to file.

4.2 Benchmarks

After implementing the changes to the source code, the benchmarking code was rerun using the altered version of ROOT 6.41.01. The output plots can be seen in Figures 4 and 5.

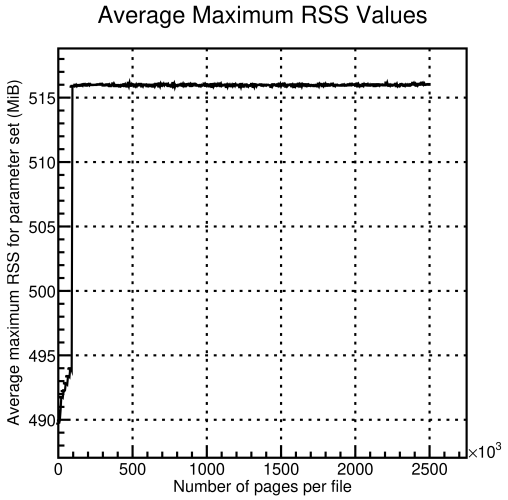


Figure 4: Shows the average maximum resident set size (RSS) versus the number of pages written to file. Output from the benchmarking code described in Section 2

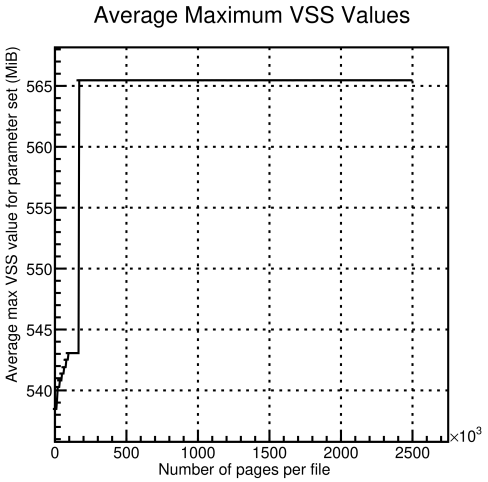


Figure 5: Shows the average maximum virtual set size (VSS) versus the number of pages written to file. Output from the benchmarking code described in Section 2.

In the beginning, Figures 4 and 5 are the same as Figures 2 and 3 in Section 3. The difference arises once there are enough pages being written to commit a cluster. After clusters start being written, instead of steadily increasing, memory usage stays roughly constant. Since the cluster group data is still accumulating over time, this can not be called fully constant. That being said, these plots show significant improvements in the overall memory usage.

5 Future Steps

5.1 Benchmarking Code

The benchmarking code developed for this project will be taken on by the ROOT team and used as a base for further benchmarking of RNTuple data streaming. In particular, it will also be used to confirm improvements in memory usage efficiency after the changes prototyped in Section 4 have been fully implemented. Additionally, it will be modified to benchmark reading of RNTuple data from file and confirm improvements on this front as well.

5.2 ROOT Source Code Changes

Given the potential improvements in efficiency of memory usage confirmed in Section 4, these changes to the source code are to be finalized by updating the ROOT source code.

The next major step after the simple modifications to the source code is to develop a heuristic for automatically writing cluster groups. During the

prototyping phase, cluster groups were simply written after an arbitrary number of clusters. In production a more sophisticated method will be implemented, likely based on the number of produced pages.

6 Acknowledgments

I would like to thank all members of the ROOT team in the EP-SFT group at CERN for laying the foundations of this project and enabling my progress. In particular, I would like to thank to my supervisors Jakob Blomer and Jonas Hahnfeld for their support and guidance throughout this project.

I would also like to acknowledge the Institute of Particle Physics as well as the National Sciences and Engineering Research Council of Canada for supporting this project.

References

[1] Jakob Blomer, Philippe Canal, Axel Naumann, and Danilo Piparo. Evolution of the root tree i/o. *EPJ Web of Conferences*, 245:02030, 2020.

[2] Rene Brun and Fons Rademakers. ROOT - an object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment A*, 389:81–86, 1997.

[3] Jonas Hahnfeld, Jakob Blomer, and Thorsten Kollegger. Parallel Writing of Nested Data in Columnar Formats, 2024. Euro-Par 2024.

[4] Anna Harms. RNTuple Memory Usage Benchmarking Code. `https://github.com/a-harms/mem_usage_vs_page_num`, 2026.