Optimizing ATLAS data storage: the impact of compression algorithms on ATLAS physics analysis data formats

Caterina Marcon[1,*], *Alaettin Serhan* Mete[2,**], *Peter* Van Gemmeren[2], and *Leonardo* Carminati[1]

[1]INFN Milano
[2]Argonne National Laboratory

Abstract. The increased footprint foreseen for Run-3 and HL-LHC data will soon expose the limits of currently available storage and CPU resources. Data formats are already optimized according to the processing chain for which they are designed. ATLAS events are stored in ROOT-based reconstruction output files called Analysis Object Data (`AOD`), which are then processed within the derivation framework to produce Derived `AOD` (`DAOD`) files. Numerous `DAOD` formats, tailored for specific physics and performance groups, have been in use throughout the ATLAS Run-2 phase. In view of Run-3, ATLAS has changed its analysis model, which entailed a significant reduction of the existing `DAOD` flavors. Two new formats, unfiltered, skimmable on read and designed to meet the requirements of the majority of the analysis workflows, have been proposed as replacements: `DAOD_PHYS` and `DAOD_PHYSLITE`, a smaller format containing already calibrated physics objects. As ROOT-based formats, they natively support four lossless compression algorithms: `lzma`, `lz4`, `zlib` and `zstd`. In this study, the effects of different compression settings on file size, compression time, compression factor and reading speed are investigated considering both `DAOD_PHYS` and `DAOD_PHYSLITE` formats. Moreover, the impact of the `AutoFlush` parameter controlling how in-memory data structures are serialized to ROOT files, has been evaluated. This study yields new quantitative results that can serve as a paradigm on how to make compression decisions for different ATLAS use cases. As an example, for both `DAOD_PHYS` and `DAOD_PHYSLITE`, the `lz4` library exhibits the fastest reading speed, but results in the largest files, whereas the `lzma` algorithm provides larger compression factors at the cost of significantly slower reading speeds. In addition, guidelines for setting appropriate `AutoFlush` values are outlined.

1 Introduction

Particle physics has an ambitious experimental program for the coming decades: during the High Luminosity Large Hadron Collider (HL-LHC) phase, scheduled to begin data taking in 2029 [1], events will be delivered and collected at unprecedented rates. Up to 200 interactions per proton-proton bunch crossing are expected and, by the end of Run-5, an integrated luminosity five times larger than the combination of all the previous runs will be delivered [1]. The

*e-mail: caterina.marcon@mi.infn.it
**e-mail: alaettin.serhan.mete@cern.ch

ATLAS experiment [2] expects to record data at 10 kHz, which is approximately ten times more than during previous runs, and, in addition, a comparable amount of Monte Carlo (MC) simulated data will be required to prevent simulation-dominated uncertainties. An extrapolation of the present computing model to HL-LHC conditions shows a significant shortfall in both computational capacity and disk space [1, 3]. In particular, storage will present a significant issue for HL-LHC computing: if CPU needs will flatten out once the design luminosity is reached, storage requirements will continue to increase over the lifetime of the HL-LHC.

ATLAS events are stored in ROOT-based [4] reconstruction output files called Analysis Object Data (`AOD`), which are then processed within the derivation framework to produce Derived `AOD` (`DAOD`) files. All file types are written in `xAOD` format with ROOT `TTrees` as the underlying data structure. Throughout Run-2 (2015-2018), several `DAOD` formats tailored for specific physics and performance groups, have been used. In view of Run-3, to limit the excessive content overlap between formats and reduce the storage needs, ATLAS is implementing a new analysis model, which entails a significant reduction of the existing `DAOD` flavors. Two new formats, unfiltered and skimmable, have been proposed as replacements [5]:

- `DAOD_PHYS` (∼50 kB/event): this format has been designed to meet the requirements of the majority of the analysis workflows and is primarily aimed at Run 3 analyses;
- `DAOD_PHYSLITE` (∼10 kB/event): this is a smaller format containing already calibrated physics objects and, as a consequence, the variables used to apply the calibrations do not need to be stored. This format will be used in Run 3 alongside its larger counterpart `DAOD_PHYS` and it will be the main format for Run 4.

The term *data compression* refers to the process of encoding information using fewer bits than the original representation. There are two different compressions: *lossless* and *lossy*. The first approach aims to reduce bits by identifying and eliminating statistical redundancy without causing information loss. In this case the process is reversible [6, 7]. The lossy compression, on the contrary, sacrifices the depth of the data to reduce the footprint for storing, handling and transmitting content. With this approach, an irreversible compression is applied.

Different lossless compression algorithms are already incorporated in ROOT and, as ROOT-based formats, both `DAOD_PHYS` and `DAOD_PHYSLITE` natively support lossless compression. This study aims to present an in-depth analysis about the impact of ROOT compression algorithms on the two formats. File size, compression time, compression factor and reading speed have been considered as the metrics for evaluating the effect of the compression settings. Moreover, the impact of the `AutoFlush` parameter, able to control how in-memory data structures are serialized, has been investigated.

2 Methods

ROOT is an open-source data analysis framework mainly written in C++ suitable for handling and visualizing large amount of data, including large columnar datasets[1], used by ATLAS as well as all LHC experiments.

In ROOT columnar datasets are represented by the `TTree` object (often simply referred to as tree) [8]. Data are logically structured in a list of independent columns called *branches* and *entries* (Figure 1). A branch can contain values of any fundamental type (or collections of those). As shown in Figure 1, data are serialized column-wise into buffers that are kept in memory until a certain size (that can be specified during branch creation) is reached and then automatically written to disk. Once the buffer is full, it gets compressed into a *basket*. To

[1] A columnar dataset is a database management system optimized to store data in columns. These databases can efficiently write and read data to and from disk in order to speed up the time required to return a query.

allow more efficient pre-fetching and better chunking of tree data, baskets are grouped into another structure referred to as *cluster*.

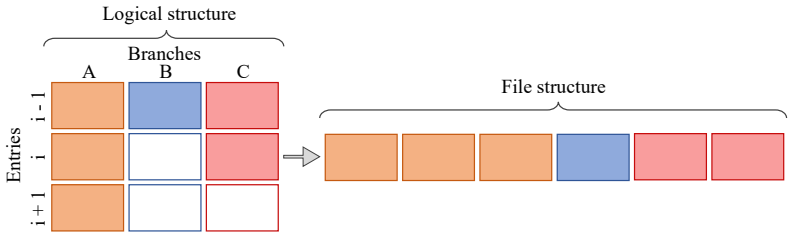

Figure 1: A simplified representation of the ROOT serialization mechanism [9].

ROOT provides also different mechanisms to control how data are written to ROOT files. The `AutoFlush` and `SplitLevel` mechanisms are particularly relevant to this study since they can have an impact on the performance of compression algorithms.

The tree can flush data (already structured in baskets) to file once a given cluster size (or a given number of events) is reached. By default this is done approximately every 30 MB of compressed data. The `AutoFlush` parameter specifies how large a single compression unit of a TTree is in terms of number of events/bytes. If required, TTrees can be structured in such a way that, for each branch, different sub-branches are created. The recursion level of nested splitting is called `SplitLevel` and it is configurable during the branch creation. Different values can be assigned to the `SplitLevel` parameter, ranging from 0 (no splitting, all data stored in the same branch, optimal when class dictionaries [10] are available) to 99, the default level, where all members at any recursion level are split into native types/attributes (optimal when dictionaries are not available).

Compression and decompression operations are a ROOT core functionality. Indeed, the framework provides four different algorithms for lossless compression[2]: `zlib`, `lzma`, `lz4`, `zstd`. These are reversible and already validated in ROOT from a physics standpoint: this lifts the requirement of a validation phase for this study.

All the algorithms can be tuned via the *CompressionLevel* option ranging from 1 to 9, where the latter offers the strongest compression.

The results presented in this paper have been obtained from a representative sample for the `DAOD_PHYS` format ($t\bar{t}$ sample, pre-compression size = 15.92 GB, original `AutoFlush` value = 500, original `SplitLevel` value = 99) and one for the `DAOD_PHYSLITE` format ($t\bar{t}$ sample, pre-compression size = 12.46 GB, original `AutoFlush` value = 1000, original SplitLevel value = 99). All calculations are carried out with ROOT 6.24 on a dedicated CERN standalone machine (Table 1). Each test has been repeated five times and the standard deviations are of the order of 3% for each test presented. For both formats, compression and reading tests have been performed on all branches of the CollectionTree, which accounts for ~90% of the total file size.

The investigations presented in this paper focus on the following aspects:

1. **File size vs Compression Level**: To estimate the impact of different compression levels on the file size, a ROOT standalone C++ code has been used. This tool is agnostic about the specific contents of a given input file: it creates a clone of the input TTree by mapping input memory addresses to the output object; the compression algorithm, compression level and `AutoFlush` of the output file are set according to the user's

[2]These compression algorithms refer to the ROOT version 6.24 used throughout this study.

Table 1: Computing resources

	CERN standalone machine
CPU	2× AMD EPYC 7302 3.0 GHz
Architecture	64 bit
N. of cores	16
Threads per core	2
Cache	L1: 1280 KB, L2: 10.0 MB, L3: 32 MB
RAM	256 GB
Filesystem	XFS
Operating system	CentOS 7

request and the input TTree is then copied to the output object one entry at a time. This study has been done keeping the default `AutoFlush` and `SplitLevel` parameters unchanged.

2. **Compression time vs Compression Factor**: The compression factor is defined as the ratio between uncompressed and compressed data. The compression time has been measured with the default `time` utility of the Linux shell as the total walltime of the compression process, comprising the initialization of the standalone compression tool and the whole read/write process of the TTree entries. The `time` utility does not allow to separate the actual compression time from the other steps, but it is reasonable to consider it dominant; indeed, the setup and read instructions are always performed on the same input file and the only variable left is therefore the compression/writing to the output file. The default `AutoFlush` and `SplitLevel` parameters for both formats have been left unchanged.

3. **Reading speed vs Compression Factor**: The reading tests of the compressed files have been performed within the ATLAS Athena framework, as the reading process should resemble as closely as possible the actual data analysis workflow. In this context, reading speed is defined as the ratio between bytes read and process time (where process time is the time spent processing the events). The reading code is already instrumented to monitor several parameters at runtime without affecting the code performance. This is achieved by sampling the system time before and after each atomic step and by calculating the resulting Δt. All the measurements are collected by PerfStats, a tool provided by ROOT, which gives access to a range of performance statistics from within the process. Metrics are then post-processed and averaged per-event figures are obtained. For each test presented, a subset of 20000 events has been randomly read and, for each event, 50% of the variables have been randomly considered. The study has been done keeping the default `AutoFlush` and `SplitLevel` parameters for both formats unchanged.

4. **Impact of AutoFlush on the compression algorithms' performance**: The `AutoFlush` parameter specifies how large a single compression unit of a TTree is in terms of number of events. By default, the `AutoFlush` for `PHYS` is set to 500, and to 1000 for `PHYSLITE`. For each algorithm, the performance (in terms of file size and reading speed) has been tested by setting different `AutoFlush` values. All tests have been carried out by setting the compression level to 5. The same hardware and software stack mentioned above has been used for all `AutoFlush` studies.

3 Results and discussion

3.1 File size vs Compression Level

In Figures 2 and 3 the file size of PHYS and PHYSLITE formats, respectively, has been monitored as a function of different compression levels. In both cases, `zstd` level 5 has been considered as a reference since it is the current ATLAS default. For both PHYS and PHYSLITE, `lzma` provides the best compression with reductions of $\sim$ 10% and $\sim$ 20%, respectively; on the contrary, `lz4` results in the largest files with increases up to 45% for PHYS and almost 50% for PHYSLITE. The file size metric depends primarily on the compression algorithm and not on the compression level.

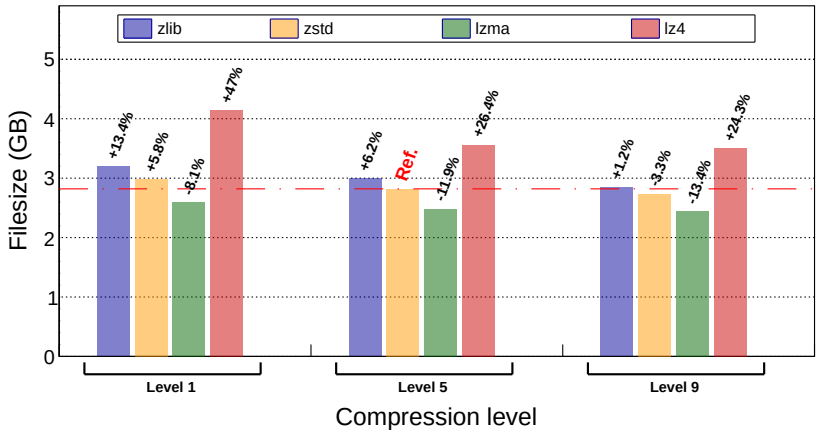

Figure 2: File size as a function of different compression levels (DAOD_PHYS format).

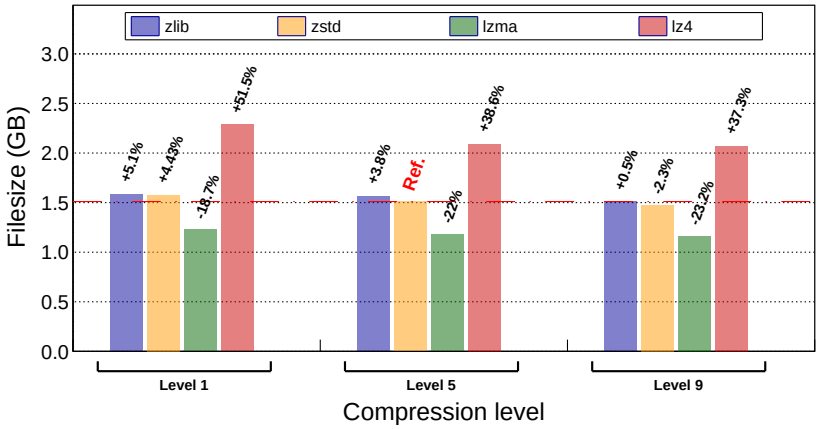

Figure 3: File size as a function of different compression levels (DAOD_PHYSLITE format).

3.2 Compression time vs Compression factor

In Figure 4 (PHYS on the left, PHYSLITE on the right), the overall compression time is presented as a function of the compression factor. A small compression time with a large com-

pression factor would be the ideal configuration. For both configurations, `lz4` provides fast compression times but suffers from low compression factors, whereas `lzma` achieves high compression factors but compression times are high. For `lzma`, `zlib`, `zstd`, compression level 9 exhibits a significant increase in compression time, without gains in compression factor; this level is only relevant when file size reduction is the most important metric.

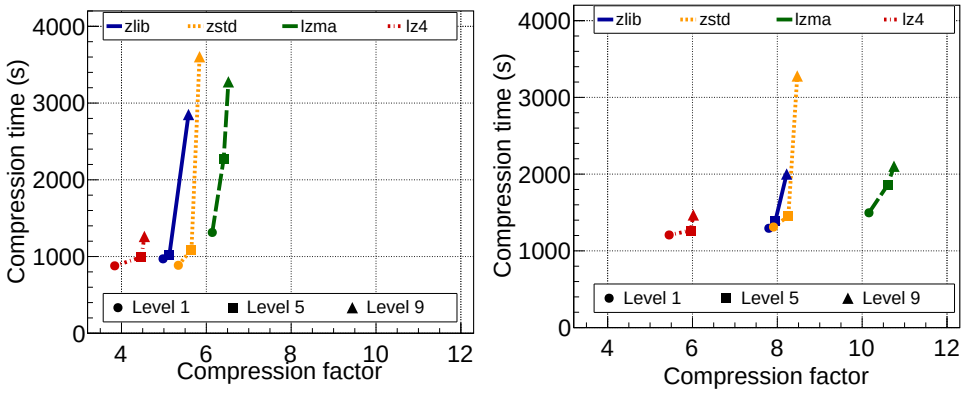

Figure 4: Compression time as a function of compression factor for `PHYS` (left) and `PHYSLITE` (right).

3.3 Reading speed vs Compression Factor

The reading speed as a function of the compression factor of the different algorithms has been also monitored for `PHYS` (Figure 5, left) and `PHYSLITE` (Figure 5, right). For these metrics the ideal configuration would be a large reading speed combined with a large compression factor. For both formats, `lzma` exhibits a low reading speed while `lz4` is the fastest algorithm in reading. For both `PHYS` and `PHYSLITE`, the reading speed depends primarily on the compression algorithm and not on the compression level. The only exception is for the `lz4` algorithm: with the `PHYSLITE` format, the impact of the compression level on the reading speed is not negligible.

3.4 Impact of AutoFlush on the compression algorithms performance

As mentioned in Section 2, ROOT also provides mechanisms to control how data is written to files. In Figure 6, the impact of the different algorithms in terms of file size has been evaluated as a function of different `AutoFlush` values ranging from 10 to 1000. On the left, results obtained for the `PHYS` format are presented, whereas on the right `PHYSLITE` results are shown. For both formats, in terms of file size reduction, compression algorithms are more efficient with more data to compress.

In Figure 7 (`PHYS` left, `PHYSLITE` right), the impact of the different algorithms on the reading speed has been evaluated as a function of different `AutoFlush` values. For `PHYS` files (left), the default `AutoFlush` value of 500 shows a good performance in terms of file size as well as reading speed. For `PHYSLITE` files (right), the original `AutoFlush` value of 1000 is reasonable, although `AutoFlush` = 500 shows a slightly better performance in terms of reading speed.

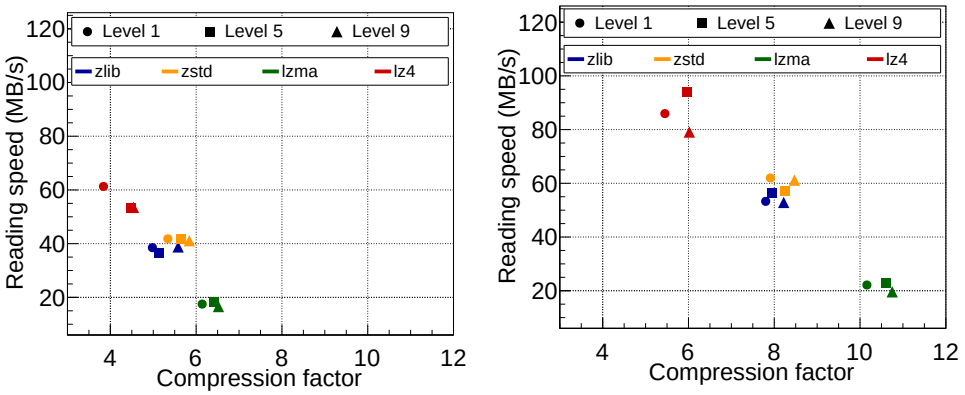

Figure 5: Reading speed as a function of compression factor for PHYS (left) and PHYSLITE (right).

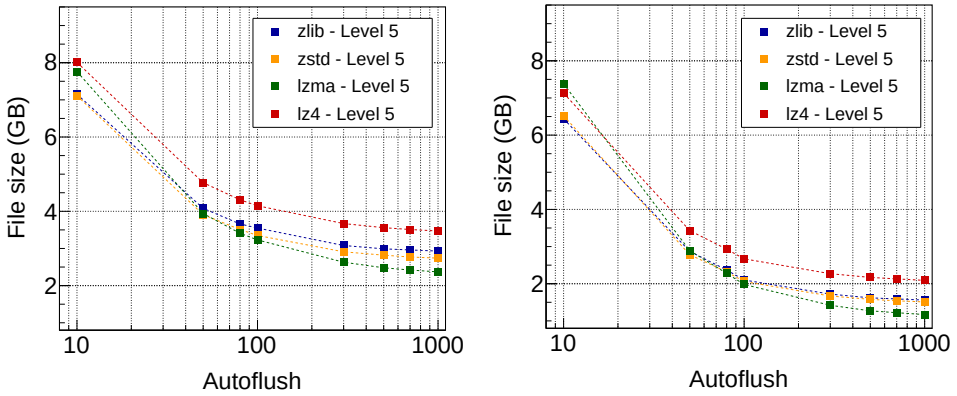

Figure 6: Impact of ROOT compression algorithms in terms of file size as a function of different AutoFlush values for PHYS (left) and PHYSLITE (right).

4 Conclusions and outlook

The LHC future runs will provide higher luminosity of particle collisions and storage will present a significant challenge for HL-LHC computing.

Since at the lowest level, ATLAS data is primarily managed using the ROOT analysis framework, there is an increasing interest in profiling the available lossless compression algorithms.

This study has shown that for both types of derived formats, lz4 is the fastest in reading but results in the largest files: this compression setting should be considered when fast reading is more important than file size reduction. For both formats, lzma provides higher compression at the cost of significantly slower reading speeds: this algorithm should be considered when file size reduction is the key metric.

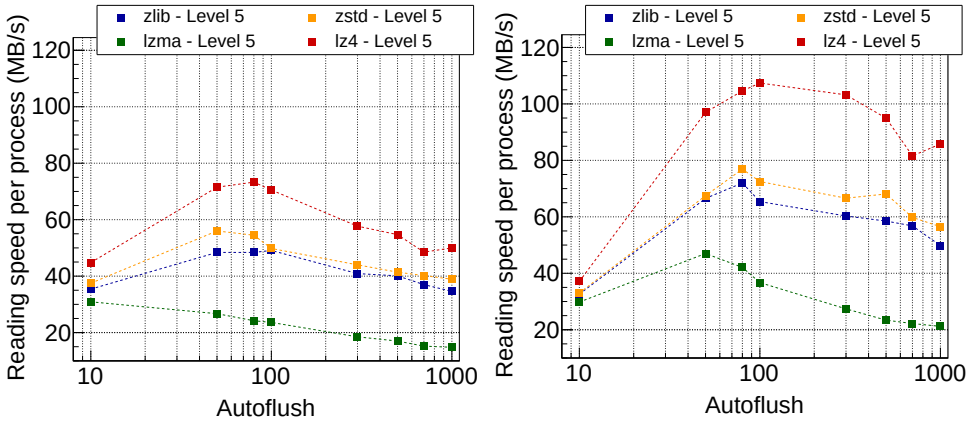

Figure 7: Impact of ROOT compression algorithms in terms of reading speed as a function of different `AutoFlush` values for `PHYS` (left) and `PHYSLITE` (right).

For both types of derived files, `AutoFlush` = 500 could be considered as a good compromise, taking into account both file size and reading speed. As a result of this study, the `AutoFlush` setting for `DAOD_PHYSLITE` files has been tuned to 500 events [11].

Studies on memory profiling and partial-event reading are ongoing.

References

[1] ATLAS Collaboration, Tech. rep., CERN (2022), `https://cds.cern.ch/record/2802918`

[2] ATLAS Collaboration, JINST **3**, S08003 (2008), doi:10.1088/1748-0221/3/08/s08003

[3] J. Albrecht et al., Computing and Software for Big Science **3** (2019), doi:10.1007/s41781-018-0018-8

[4] R. Brun, F. Rademakers, Nucl. Instrum. Methods Phys. Res. A: Accel. Spectrom. Detect. Assoc. Equip. **389**, 81 (1997), new Computing Techniques in Physics Research V, doi:https://doi.org/10.1016/S0168-9002(97)00048-X

[5] J. Schaarschmidt, J. Catmore, J. Elmsheuser, L.A. Heinrich, N.E. Krumnack, A.S. Mete, N. Ozturk (ATLAS), Tech. rep., CERN, Geneva (2023), `https://cds.cern.ch/record/2870350`

[6] I.M. Pu, in *Fundamental Data Compression*, edited by I.M. Pu (Butterworth-Heinemann, 2006), ISBN 978-0-7506-6310-6

[7] K. Sayood, in *Introduction to Data Compression (Fifth Edition)*, edited by K. Sayood (Morgan Kaufmann, 2018), ISBN 978-0-12-809474-7

[8] Blomer, Jakob, Canal, Philippe, Naumann, Axel, Piparo, Danilo, EPJ Web Conf. **245**, 02030 (2020), doi:10.1051/epjconf/202024502030

[9] O. Shadura, B. Bockelman, Journal of Physics: Conference Series **1525**, 012049 (2020), doi:10.1088/1742-6596/1525/1/012049

[10] *I/O of custom classes.* (2023), `https://root.cern/manual/io_custom_classes/`

[11] A. Mete, *Update PHYSLITE AutoFlush to 500 events* (2023), `https://gitlab.cern.ch/atlas/athena/-/merge_requests/63813`