Shared I/O Developments for Run 3 in the ATLAS Experiment

Alaettin Serhan Mete[a,*] **and Peter Van Gemmeren**[a]

[a]*Argonne National Laboratory*[†],
9700 S Cass Ave, Lemont, IL 60439, United States

E-mail: amete@anl.gov, gemmeren@anl.gov

The ATLAS experiment extensively uses multi-process (MP) parallelism to maximize data-throughput especially in I/O intensive workflows, such as the production of Derived Analysis Object Data (DAOD). In this mode, worker processes are spawned at the end of job initialization, thereby sharing memory allocated thus far. Each worker then loops over a unique set of events and produces its own output file, which in the original implementation needed to be merged at a subsequent step that would be executed serially. In Run 2, SharedWriter was introduced to perform this task on the fly, with an additional process merging data from the workers while the job was running, eliminating the need for the extra merging step. Although this approach was very successful, there was room for improvements, most notably in the event-throughput scaling as a function of the number of workers. This was limited by the fact that the Run 2 version does all data compression within the SharedWriter process. For Run 3, a new version of SharedWriter has been written to address the limitations of the original implementation by moving compression of data to the worker processes. This development also paves the way for using it in a hybrid mode of multi-thread (MT) and MP workflows to maximize the I/O efficiency. In this talk, we will discuss the latest developments in Shared I/O in the ATLAS experiment.

41st International Conference on High Energy physics - ICHEP2022
6-13 July, 2022
Bologna, Italy

*Speaker

†Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357

Contents

1. Introduction

Since the first days of data-taking, the ATLAS experiment [1] at CERN's LHC has been using Athena [2] as its main software framework. Athena is an open-source project that is based on Gaudi [3]. It consists of about 4 (1.5) million lines of C++ (Python) code. The core framework and the algorithms are written in C++, while Python is primarily used for job configuration/steering.

Originally designed to be executed serially on a single CPU core, Athena was extended to support multi-process (MP) parallelism for Run-2. This mode of operation, called AthenaMP, is still being used for a number of workflows, including the production of Derived Analysis Object Data (DAOD).

The DAOD, a ROOT [4] file, is the primary data format that is used in physics analyses. In Run-2, the physics analysis model was based on the idea of creating dedicated DAODs that target analyses with similar signatures. Skimming (selecting events) and slimming (keeping only relevant variables) were applied to minimize the overlap between different formats. In Run-3, ATLAS switched to a new physics analysis model [5] where analyses use two common/inclusive DAOD formats, namely DAOD_PHYS and DAOD_PHYSLITE. These formats include all reconstructed events and all common variables. Therefore, producing them is more Input/Output (I/O) intensive, compared to producing the derivation formats used in Run-2. This necessitated an upgrade to the I/O system of Athena that is used in the AthenaMP workflows.

2. Shared I/O at ATLAS

As mentioned in the previous section, ATLAS has been using AthenaMP for a number of workflows since the beginning of Run-2. This mode primarily takes advantage of Linux's *fork* and *copy-on-write* mechanisms. A number of so-called *worker* processes are forked from the main process just after the job initialization. This allows the workers to share memory allocations that happen until their creation, thanks to the *copy-on-write* technique. This significantly reduces the memory usage of the job compared to running serial Athena jobs in parallel (as many as the number of workers).

In the original AthenaMP implementation, each worker processes a set of unique events and writes out its own output files. These intermediate files are then merged in a subsequent step, called the file merging, that runs on a single CPU core. However, this not only increases the overall job execution time, hence reducing the event throughput, but also introduces inefficiencies in the use of available hardware resources. In order to improve the situation, SharedWriter [6, 7] was introduced during Run-2.

SharedWriter is a special worker that executes alongside the other workers. Instead of processing events, it retrieves the output data objects from the workers and merges them on the fly, eliminating the need to run the additional file merging step. Throughout Run-2, the original implementation of the SharedWriter, now referred to as the *legacy* mode, is used in the DAOD production jobs. It is estimated to save more than 30% of wall-time in these jobs. The legacy SharedWriter serializes the data it retrieves from the workers, compresses them, and summarizes the meta-data before saving them in the final product.

For Run-3, a number of improvements are made to the SharedWriter to increase its performance. Instead of a single instance of SharedWriter performing everything, most of the CPU intensive work is distributed across workers. This is done by adopting an approach that is similar to ROOT's ParallelFileMerger. This new approach takes advantage of socket programming. SharedWriter acts as a server and accepts connections from the clients (the workers). Workers, instead of sending their raw data to SharedWriter, serialize and compress their own data and commit clusters of events into in-memory files, called TMemFiles. When it is time to flush the data to disk, workers send their already serialized and compressed data to SharedWriter, and SharedWriter simply merges them. At the end of the job, SharedWriter collects the meta-data from the workers, summarizes and commits them into the output.

The downside of this approach compared to the legacy SharedWriter is the increased memory usage. This stems from the fact that now each worker needs to have its own buffers for the in-memory output files. However, it should be noted that this situation is very similar to the original AthenaMP implementation where each worker used to handle its own output. Having said that, compared to the legacy SharedWriter, the event-throughput gets a significant boost thanks to the parallelization of the most CPU intensive operation in the I/O chain, namely the compression. As far as the storage is concerned, the legacy SharedWriter has a slight edge over the new SharedWriter. In order to keep the job's memory usage at a reasonable level, the output is periodically flushed to disk once every certain number of events, $N_{\mathrm{AutoFlush}}$. The buffers are optimized when this disk-flush occurs for the first time, meaning the first cluster is always unoptimized. In the legacy SharedWriter, the output has a single unoptimized cluster, whereas in the new SharedWriter there is one unoptimized cluster per worker. Nonetheless, the practical impact this has on the performance diminishes as the job processes more events than the frequency at which the output is flushed to disk, i.e. $N_{\mathrm{total}} \gg N_{\mathrm{AutoFlush}}$, which is typical for production jobs.

3. Performance Benchmarks

In order to assess the performance of the new SharedWriter, a number of benchmark tests are performed. The tests consist of running a DAOD production job that produces one DAOD_PHYS and one DAOD_PHYSLITE file, a so-called derivation train, with varying numbers of worker processes.

In each iteration, the input is 25000 data reconstruction events. The tests are run on a dedicated bare-metal machine that has two AMD EPYC 7302 16-Core Processors 3GHz and a total of 252 GB of memory. In order to disentangle the hardware effects as much as possible, the Simultaneous Multi-Threading (SMT), as well as clock frequency boosting, are disabled.

Table 1 shows the scaling of the event-throughput and the memory usage. It should be noted that the standard ATLAS production setup is using 8 cores (workers) per job. In this configuration, the new `SharedWriter` improves the event-throughput by 70%, while increasing the memory usage by 13%. However, the memory usage remains below the ATLAS distributed computing limit of 2 GB/core. For a configuration with more workers, the gains are even more dramatic.

Figure 1 shows the CPU utilization as a function of processing wall-time for the benchmark `DAOD` production job in `AthenaMP` using 8 workers. The plot that corresponds to the legacy `SharedWriter` clearly shows that a single `SharedWriter` compressing all the data cannot keep up with the workers producing them. This results in over-committed CPU usage, i.e., 9 CPU cores are used instead of 8 since the `SharedWriter` is 100% busy, and the job gets throttled. However, in the new `SharedWriter` configuration, since the CPU-intensive compression is evenly distributed across workers, `SharedWriter` does not need to work as much and does not throttle the job. This can also be seen in the Figure 2, where events processed per worker (normalized to the first worker) are shown. In the legacy configuration, `SharedWriter` can only serve a handful of workers effectively, which lowers the efficiency of the additional workers, hence parallelization. This behavior is fixed in the new `SharedWriter`. However, it should be noted that the legacy `SharedWriter` performs just as well in a configuration where one produces a Run-2 `DAOD` format, which is less I/O-intensive. Therefore, it would be fair to conclude that although the new derivation formats require the extra boost they get from the new `SharedWriter`, the legacy `SharedWriter` can still perform adequately in some scenarios, especially in less I/O-intensive jobs that are memory limited. It is also worth mentioning that any `SharedWriter`, legacy or new, is better than plain `AthenaMP` followed by a serial merge job.

Number of Cores	**Events/Wall-time [1/s]**			**Memory/Core [GB]**		
	Legacy	**New**	**Δ [%]**	**Legacy**	**New**	**Δ [%]**
4	4.9	6.0	+21	1.94	2.10	+9
8	6.7	11.4	+70	1.67	1.88	+13
16	6.7	21.2	+216	1.45	1.81	+25

Table 1: The scaling of event-throughput and memory usage as a function of number of workers for the benchmark `DAOD` production job.

4. Outlook and Conclusions

The changes made to the physics analysis model for Run-3 necessitated a number of improvements to how I/O is handled in the `DAOD` production jobs in ATLAS. In order to improve the performance, a new `SharedWriter` mode is introduced, which solves a number of shortcomings of the legacy implementation. At 8 processes, a typical production setup, the new `SharedWriter` is

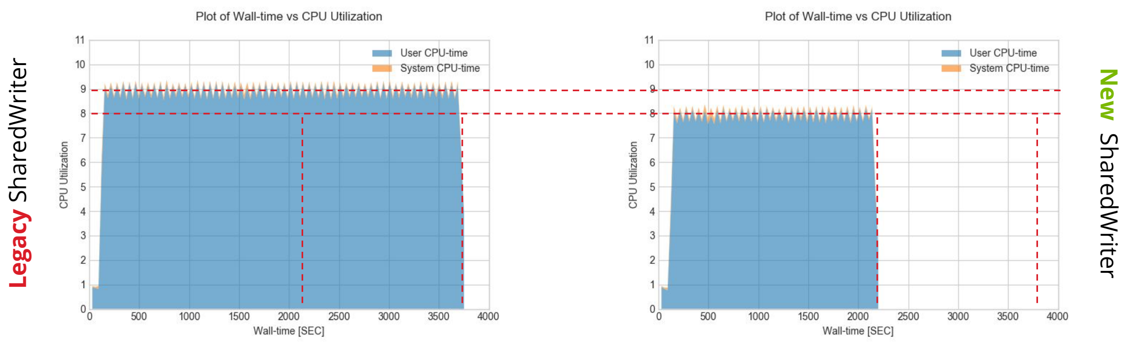

Figure 1: The CPU utilization as a function of processing wall-time for the benchmark `DAOD` production job in `AthenaMP` using 8 workers. The plot on the left corresponds to the configuration using the legacy `SharedWriter`, while the one on the right corresponds to the configuration using the new `SharedWriter`. The plots are obtained using `prmon` [8].

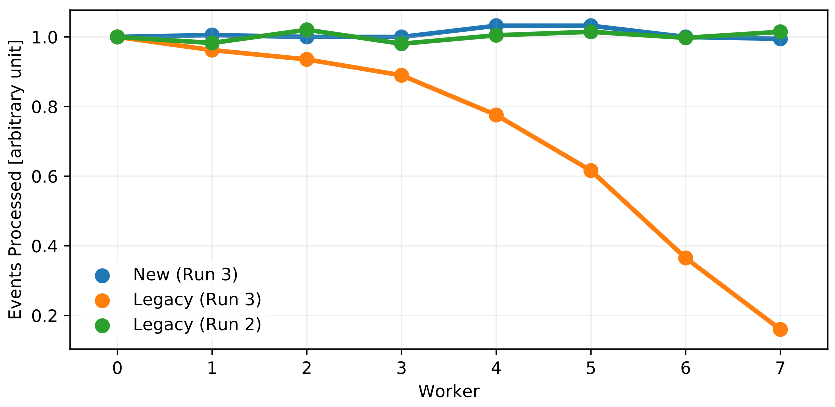

Figure 2: The number of events processed per worker (normalized to the first worker) for the benchmark `DAOD` production job in `AthenaMP` using 8 workers. The orange curve corresponds to the configuration using the legacy `SharedWriter`, while the blue curve to the configuration using the new `SharedWriter`. For comparison, the green curve shows the result for producing a Run-2 type `DAOD` file with the legacy `SharedWriter`.

found to improve the event-throughput by as much as 70% in a scenario where the job produces a so-called derivation train containing `DAOD_PHYS` and `DAOD_PHYSLITE` formats.

Beyond just `DAOD` production, `SharedWriter` can also help improve the performance of production jobs that are I/O-bound. For Run-3, ATLAS upgraded the `Athena` framework to support multi-threading, namely `AthenaMT`. ATLAS is in a unique position, where the benefits of both the MT and the MP workflows can be married in a hybrid mode, called `AthenaMP/MT`. This mode could be a viable option that both optimizes the memory usage as well as having optimal I/O handling thanks to `SharedWriter`. In that regard, the benefits of `SharedWriter` potentially go beyond just `AthenaMP` and these new modes will be investigated in the future for various other workflows.

References

[1] The ATLAS Collaboration. (2008). The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, **3**, S08003. https://doi.org/10.1088/1748-0221/3/08/S08003

[2] The ATLAS Collaboration. (2021). Athena. https://doi.org/10.5281/zenodo.4772550

[3] Barrand, G. et al. (2001). GAUDI – A software architecture and framework for building HEP data processing applications. *Comp. Phys. Comm.*, **140**, 45-55. https://doi.org/10.1016/S0010-4655(01)00254-5

[4] Brun, R. & Rademakers, F. (1997). ROOT - An Object Oriented Data Analysis Framework. *Nucl. Inst. & Meth. in Phys. Res.*, **A 389**, 81-86. https://doi.org/10.5281/zenodo.848818

[5] Elmsheuser, J. et al. (2020). Evolution of the ATLAS analysis model for Run-3 and prospects for HL-LHC. *EPJ Web Conf.* **245** 06014. https://doi.org/10.1051/epjconf/202024506014

[6] van Gemmeren, P. et al. (2012). I/O Strategies for Multicore Processing in ATLAS. *J. Phys.: Conf. Ser.* **396** 022054. https://doi.org/10.1088/1742-6596/396/2/022054

[7] van Gemmeren, P. et al. (2017). Shared I/O components for the ATLAS multi-processing framework. https://cds.cern.ch/record/2278398

[8] Stewart, G. A. & Mete, A. S. (2018). PrMon. https://doi.org/10.5281/zenodo.2554202