I/O performance studies of analysis workloads on production and dedicated resources at CERN

Andrea Sciabà[1], *Jakob* Blomer[1], *Philippe* Canal[2], *Dirk* Duellmann[1], *Enrico* Guiraud[1], *Axel* Naumann[1], *Vincenzo Eduardo* Padulano[1], *Bernd* Panzer-Steindel[1], *Andreas* Peters[1], *Markus* Schulz[1], and *David* Smith[1]

[1]CERN, European Organization for Nuclear Research, Geneva, Switzerland
[2]Fermi National Accelerator Laboratory, Batavia, Illinois, USA

Abstract. The recent evolutions of the analysis frameworks and physics data formats of the LHC experiments provide the opportunity of using central analysis facilities with a strong focus on interactivity and short turnaround times, to complement the more common distributed analysis on the Grid. In order to plan for such facilities, it is essential to know in detail the performance of the combination of a given analysis framework, of a specific analysis and of the installed computing and storage resources. This contribution describes performance studies performed at CERN, using the EOS disk-based storage, either directly or through an XCache instance, from both batch resources and high-performance compute nodes which could be used to build an analysis facility. A variety of benchmarks, both synthetic and based on real-world physics analyses and their corresponding input datasets, are utilized. In particular, the RNTuple format from the ROOT project is put to the test and compared to the latest version of the TTree format, and the impact of caches is assessed. In addition, we assessed the difference in performance between the use of storage system specific protocols, like XRootd, and FUSE. The results of this study are intended to be a valuable input in the design of analysis facilities, at CERN and elsewhere.

1 Introduction

The evolution of the LHC physics program and the accelerator and detector capabilities towards HL-LHC are driving the ever expanding requirements for data processing, which are the main focus of software development in the experiments. Not only processing time per event must be reduced without any loss in physics accuracy, but smaller event data formats become essential to reduce storage needs. These changes, together with the emergence of new data analysis ecosystems based on Python and distributed computing tools, are shifting analysis techniques towards interactivity, where physicists can realistically run their analyses on large datasets in a matter of minutes. One, simple, way to achieve it is to use high performance nodes with a high core count and large amounts of memory and storage, to be shared by a few users. A much more flexible alternative is to establish analysis facilities, allowing many users to run interactively on a large amount of distributed and dedicated resources. Both approaches need to be extensively tested and prototyped in order to understand the hardware requirements and the behavior of new analysis frameworks at scale.

An investigation on LHC analysis at CERN and its impact on the computing infrastructure was conducted in 2022, showing that the CERN infrastructure is able to sustain the analysis load (largely consisting of batch jobs) with ample margin and no dedicated resources for analysis were needed [1]. However, it is expected that interactive analysis will considerably increase in scale, which is the motivation behind the work presented in this contribution.

2 Testing configuration

The set of measurements conducted have as a goal to study the behavior of different, realistic or actual analysis workloads based on different analysis ecosystems over different data access protocols and storage configurations, ultimately to help optimizing resource allocation as a function of the prevailing use cases.

As capturing the full spectrum of physics analyses is impractical, our approach consists in collecting a variety of workloads from different sources and with different characteristics. The workloads we used for testing can be grouped as follows:

a) synthetic benchmarks: *rootreadspeed*, *xrdcp*;

b) simplified analyses: RNTuple Virtual Probe Station, Analysis Grand Challenge (AGC) CMS $t\bar{t}$ analysis [2];

c) real analyses: CMS boosted Higgs boson search, and an undisclosed CMS analysis[1].

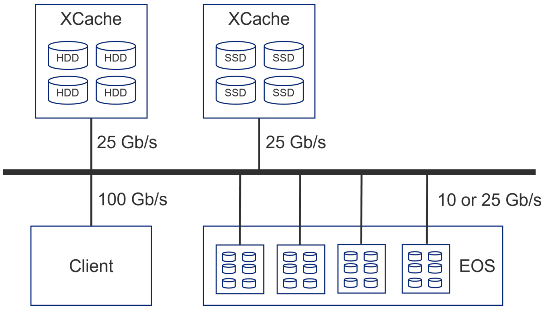


Figure 1: Testbed architecture, consisting of both dedicated and production services.

The hardware setup, shown in Fig. 1, consisted of:

- a client node with two AMD EPYC 7702 CPUs for a total of 128 cores, 1 TB of RAM and 40 TB of SSD storage;
- two XCache instances with two Intel Xeon Silver 4216 for a total of 32 cores and 192 GB of RAM each, one instance with 1 PB in hard drives and the other with 32 TB in SSDs;
- the EOS storage system at CERN [3].

An XCache server is a special XRootD [4] server configured to work as a caching layer for remote data access using the XRootD protocol and is being widely used by HEP computing centres to optimize network utilization [5].

The PrMon tool [6] was used to extract several performance metrics. In particular, the wallclock time, the "pseudo" CPU efficiency (total CPU time divided by the wallclock time times the number of workers or threads) and the read rate (from the local file system or from the network, as relevant) were measured to quantify performance and scalability.

[1] The nature of the analysis cannot be described at this time, as it is still unpublished.

3 Synthetic benchmarks

In order to establish a reference for the read performance of local access and EOS access, we measured the maximum achievable data rates using different methods: the *xrdcp* command and the the *rootreadspeed* tool, from the XRootD client and the ROOT software distribution [7] respectively. They both allow to parallelize the file transfers by setting the number of streams/threads, which we chose to match the number of cores in the client node. The average read rates are summarised in Table 1.

Table 1: Average read rates when reading the input dataset of the AGC analysis from the local filesystem of from EOS.

Tool	local filesystem (GB/s)	EOS (GB/s)
xrdcp	16.1	4.6
rootreadspeed	9.0	4.6

The EOS rates are capped to 40 Gb/s due to the local network architecture. The difference between *xrdcp* and *rootreadspeed* can be attributed to the fact that the former is a purely sequential read without any regard for the file content, while the latter also decompresses the data using the ROOT I/O, and as such it can be thought of measuring the best possible data rate for a RDataFrame-based analysis [8].

4 RNTuple Virtual Probe Station

The ROOT team developed a collection of simplified analysis code for ATLAS, CMS, LHCb and H1 data [9]. The code has been constructed so that it can perform the same analysis using data in TTree and RNTuple format [10] and can be run either single threaded or using a predefined number of threads. Furthermore, it can make use of RDataFrame.

The analyses cover both sparse and dense reading, from a few percent to 80% of the data. The data is organised following different data models. The LHCb based analysis is based on flat ntuples, H1 and CMS use smart collections optimized for ROOT TTree I/O and ATLAS has been implemented with *std::vector* collections.

These test codes collect their own metrics of performance parameters. In addition the probes have been run within the PrMon tool to complement and cross check the internal measurements.

File sizes used for the different experiments vary from 5-20 GB. Depending on the data format and the mode of running the probes collect on the order of 20 different metrics. Several of these metrics are specific to the data formats used and allow a detailed understanding of the contribution to the overall performance of the intermediary I/O and processing steps.

All tests have been run on the client node described in Section 2. Data has been stored on the local SSDs and on the EOS storage system at CERN. The data has been accessed directly and through disk and SSD based XCache instances, with populated and cold caches. To understand the performance impact of accessing EOS through FUSE, tests have been run using XRootD and FUSE for all workloads and framework combinations. Run time of FUSE based workloads has been about 50% longer.

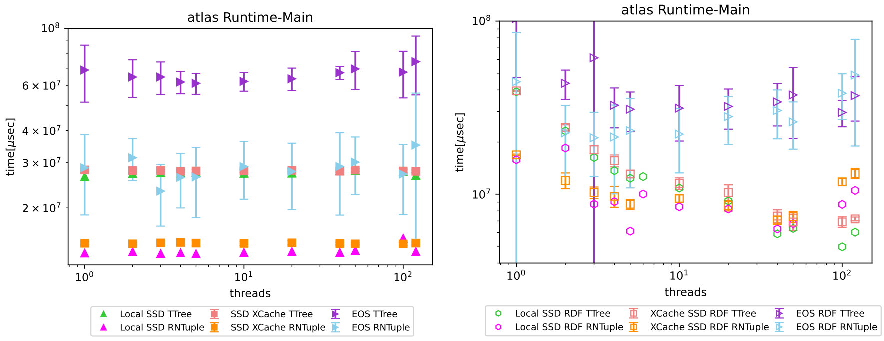
To see the impact of concurrent processing, the tasks have been run with 1-120 threads, progressively increasing the number of threads, covering the range with eleven data points.

Significant improvement of the run-time with the number of threads was been observed only when using RDataFrame. It was most pronounced when running on local data. However, improvements tapered off when more than O(10) threads where used. When accessing the data on the SSD based XCache some performance improvements could also bee seen, but less than when accessing local data.

When accessing data outside the node, especially when reading directly from EOS, significant variations in the throughput have been observed. In an attempt to measure performance that is closer to what a much larger user task would experience and to gain insight in the scale of the variations, each setup has been run ten times. This stabilized the average measurements significantly. However, it has been observed in a few per-mille of the tests that the maximum run time exceeded the average by a factor larger than five when accessing EOS directly.

When accessing data on unpopulated caches the performance was about 10% slower than when accessing EOS directly. The performance differences between disk and SSD based XCaches has been small.

As an example, some measurements for ATLAS are shown in Fig. 2. The access to EOS and the caches is via XRootD.



(a) TTree and RNTuple based analysis

(b) Same analysis using RDataFrame

Figure 2: Run time for 1-120 threads for different data location and formats, note the different scale on the time axis. When accessing data on the EOS service some measurements show a large variation, mostly created by a single measurements of the 10 leading to a data point.

5 Analysis Grand Challenge workloads

The workload used as a benchmark by the AGC was created as a technical demonstration and derived from an existing CMS $t\bar{t}$ production analysis. Its input dataset consists of 2269 files for a total of 3.6 TB, containing CMS Open Data and therefore freely redistributable. It is hosted in Nebraska, but it was replicated to EOS at CERN and to the client node.

The workload is based on the Scikit-HEP ecosystem [11] and the Coffea framework [12], but an equivalent version based on ROOT RDataFrame was created [13]. The amount of data actually read amounts to roughly 5% of the dataset size. Network access to the data was done via XRootD, and while HTTP(S) access was verified to work, it was not thoroughly tested. Access to EOS via a FUSE mount on the client node was tested only for the AGC version.

For each configuration, the workloads were run several times for different numbers of workers/threads, and errors on the recorded metrics were estimated using the standard deviations. In the case of EOS access, the errors embed the contribution from possible variations in EOS performance due to concurrent production activities.

5.1 Coffea version

This version of the workload does not depend in any way on ROOT, but it can read ROOT files via *uproot*. Coffea scales out locally using the Python Futures API, which by default creates subprocesses, but it can also use Dask to scale out to external nodes.

The workload was run against a local copy of the input dataset, a copy hosted on EOSCMS, one on EOSUSER and finally via the SSD-based and the HDD-based XCache instances with the input data pre-cached. The results are summarized in Fig. 3. These observations were made from the results:

- when running on local files, the CPU efficiency remains very high, which suggests that the workload is CPU-constrained and the I/O relatively lightweight. The scalability breaks for high numbers of workers due to a considerable increase in the CPU time;
- when reading directly from EOSCMS, the CPU efficiency stays fairly constant around a value of 60%, which points at the network latency as the bottleneck. The scalability shows a similar behavior as for the previous case. When reading from EOSUSER very similar results are obtained and are not shown;
- running via an XCache instance does not bring any performance improvements with respect to direct access to EOS, although it would still be useful in case the dataset is originally hosted at a remote site; moreover, SSDs do not perform noticeably better than HDDs (in our very basic, single-user scenario).

The performance when reading via FUSE was also measured, but the total read rate saturates around 100 MB/s; as expected, using FUSE is strongly discouraged to access input data.

Finally, the performance metrics were measured also for direct access to the Nebraska storage via XrootD and via both a cold and a warm HDD-based XCache using 64 workers; not surprisingly, access via a cold cache was slower than direct access ($\simeq$ 610 s and $\simeq$ 440 s respectively), while a warm cache performed exactly like EOS direct access.

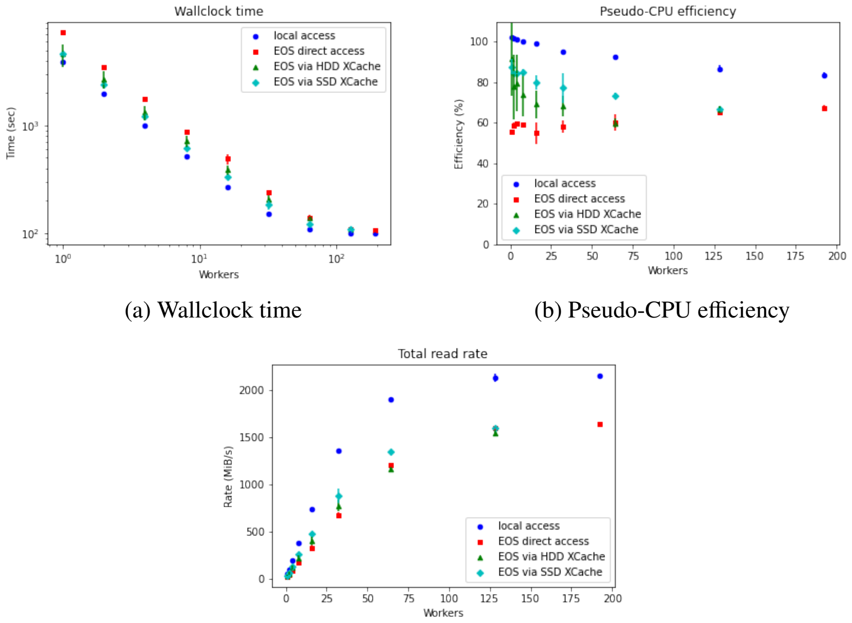
5.2 RDF version

The RDF implementation of the AGC workload was tested in the same conditions. A fundamental difference between the two implementations consists in the fact that RDataFrame uses multithreading to scale out within a client node. In the RDF case, some scalability problems were observed using ROOT 6.26 (the latest available at that time), and subsequently understood and fixed. Showing the results obtained before the fixes is still interesting, as an example of how a sub-optimal workload could impact performance and lead to different conclusions about how to best execute it.

The results are summarized in Fig. 4 and they lead to the following observations:

- when reading from the local filesystem, the workload shows excellent scalability, but with a marked increase in total CPU time for high numbers of threads, as in the Coffea case;
- direct access to EOS using ROOT 6.26 and local multithreading shows a clear bottleneck which will be discussed below;
- when using Dask to force the parallelization to happen via multiprocess, scalability is very good, confirming that the multithreading is the culprit;
- using a recent, and at the time of writing, yet unreleased version of ROOT from a nightly build, we observe a perfect scalability also for multithreading.

Another set of measurements was taken when accessing the input data via the XCache instances, having already cached the input data (“warm cache”) (Fig. 4). As it can be seen,



(a) Wallclock time

(b) Pseudo-CPU efficiency

(c) Average read rate

Figure 3: Performance metrics for AGC Coffea vs. number of worker processes for different locations of the input dataset.

the scalability “out of the box” using ROOT 6.26 is extremely poor, but this was understood as caused by three factors:

1. by default, the XRootD client uses only one event loop thread to hand the asynchronous I/O, which creates a considerable bottleneck when many concurrent file operations are performed. Setting the environment variable *XRD_PARALLELEVTLOOP* to 10 removes this bottleneck [1];

2. by default, XRootD will multiplex several network connections from the same client process to a remote XRootD server (such as an EOS disk server or an XCache instance) into one, which is in general a good idea, but will create a bottleneck when the client process uses many threads connecting to a single server. This becomes apparent in the case of XCache, as it is a single server, but it does not affect access to EOS, where files are almost guaranteed to be spread over a large number of disk servers. The multiplexing can be disabled by the user;

3. a significant lock contention affects RDF with large numbers of threads and files read via XRootD. This problem was fixed for in the ROOT library.

The values for the “optimized“ configuration in the right hand side of Fig. 4 show the impact of the first two fixes. In the case of direct EOS access, the improvement was negligible; on the other hand, the third fix has an extremely positive impact on EOAS direct access. The first two fixes should also be eventually integrated in future versions of XRootD.

https://doi.org/10.1051/epjconf/202429507025

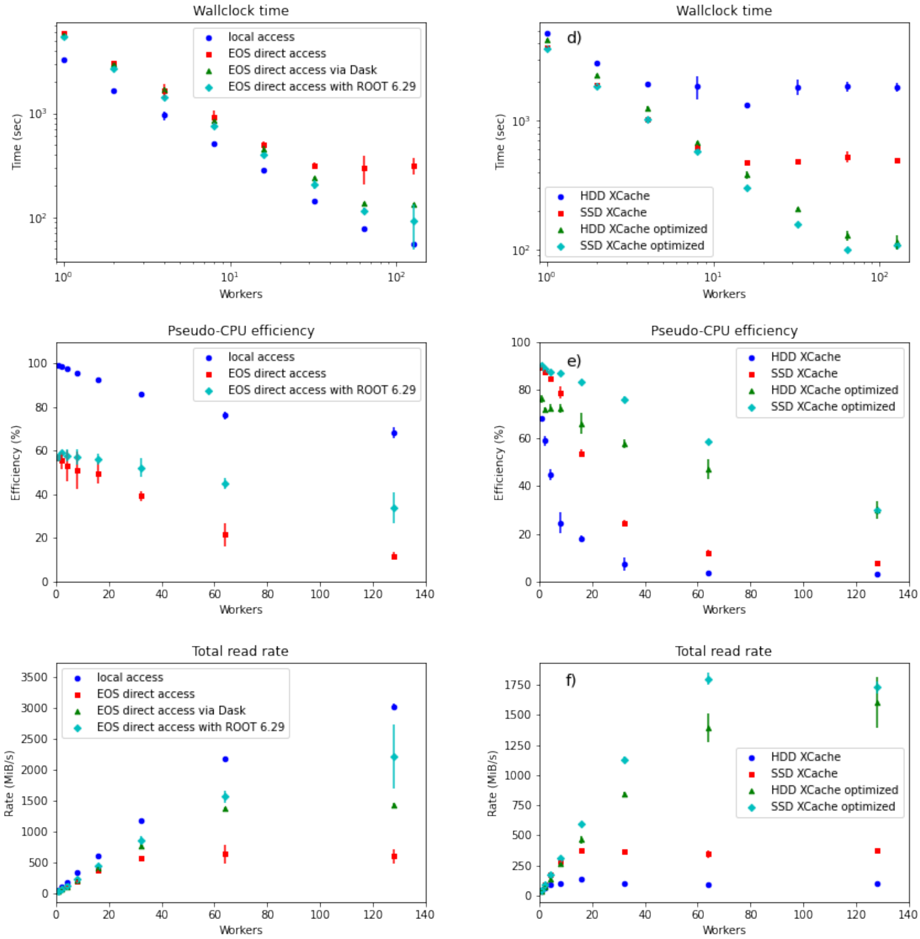


Figure 4: Performance metrics for AGC RDF: *a)* wallclock time; *b)* CPU efficiency, *c)* average read rate vs. number of workers for different locations of the input dataset; *d) e)* and *f)* show the metrics for access via XCache.

6 Boosted Higgs boson Coffea analysis

The first “real world” analysis used by us as a benchmark is a search for $ggH \rightarrow \bar{c}c$ events [14]. The input dataset used for the tests consists of data from 2017 in NanoAOD format, for a total of 6 TB and an average file size of 160 MB.

In this case, the goal was not to measure scalability, but to compare performance metrics in different data access scenarios. The results when using 64 workers are summarized in Table 2. This particular workload is clearly CPU-limited, with a rather low read rate in case of network access, much less than the AGC workload, as it can be expected from many “real” analyses. The long processing time hints at the fact that scaling out to a batch system, for example using Dask, would be absolutely necessary, if user desires his analysis to be “interactive”. Another interesting result is that access via an SSD-based XCache performs as well as local access.

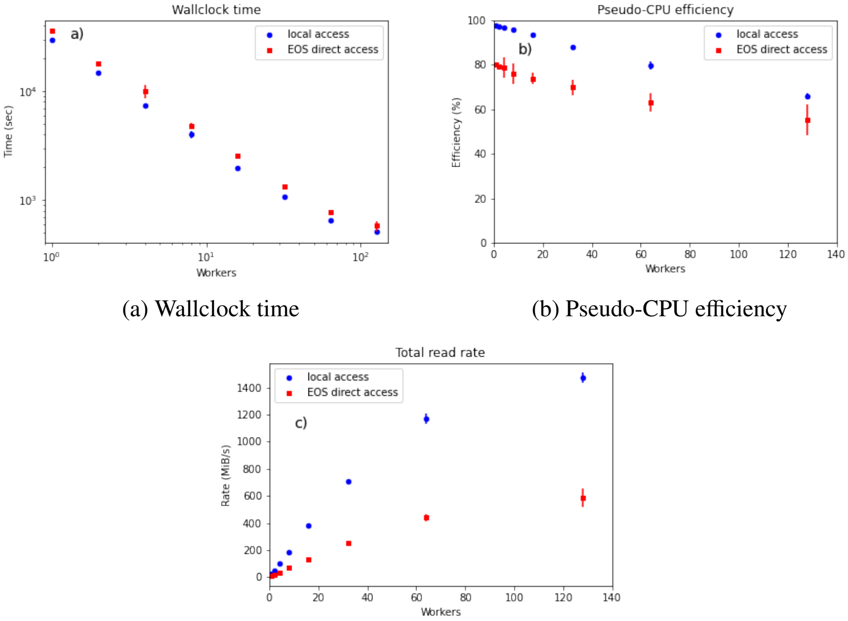
Table 2: Performance metrics for the boosted Higgs analysis for different data access modes.

	wallclock (hours)	pseudo-CPU eff (%)	read rate (MiB/s)
local access	4.11 ± 0.02	103.5 ± 0.2	268.1 ± 0.9
direct EOS access	4.62 ± 0.08	85.8 ± 4.8	34.8 ± 0.6
HDD XCache	4.82 ± 0.25	83.3 ± 5.2	32.9 ± 1.7
SSD XCache	4.09 ± 0.05	101.5 ± 0.5	39.3 ± 0.5

7 Real CMS RDF analysis

This is another example of an actual analysis, but based on RDataFrame and already used for benchmarking [15]. It originally consists of a preselection step, simulating the effect of triggers and applying a basic event selection, with some data augmentation, and a postselection step that applies the full event selection and produces histograms. The input dataset consists of 1 TB of NanoAOD events, of which 350 GB are actually read from EOS.

For the purpose of our work, only the preselection step has been considered and a preliminary version of ROOT 6.29 has been used, as it provides the best performance; the results are shown in Fig. 5.



(a) Wallclock time

(b) Pseudo-CPU efficiency

(c) Average read rate

Figure 5: Performance metrics for the real CMS RDF analysis vs. number of processing threads for different locations of the input dataset.

It can be seen that the scalability is excellent and the CPU efficiency relatively high. Compared to the AGC RDF workload, the difference in read rates between local access and EOS access is more pronounced, most probably due to the much higher fraction of data actually read.

8 Conclusions

The methodology applied to this study consisted of identifying a collection of workloads representing a variety of real or realistic analysis workloads and measure, for each one, performance and scalability under varying conditions in the context of a given computing infrastructure. The applications we tested showed excellent behavior (once all known issues were solved) and we concluded that a caching service would not necessarily bring a significant advantage with respect to direct access to EOS, which provides consistently good performance even in presence of the usual production load. Our testbed explores the case of a high performance client node with a single user, where we can expect the strongest scalability constraints; some preliminary tests where workloads were executed on the CERN batch infrastructure using Dask were perfomed (although not shown here) and represent the next step in our investigations, together with the simulation of a multi-user environment.

We would like to thank Josh Bendavid, Lindsey Gray, Alexander Held, Luca Mascetti, Andrzej Novak, Oksana Shadura and Tommaso Tedeschi for many useful discussions and suggestions and their contributions to this work.

References

[1] Dirk Duellmann, Bernd Panzer-Steindel, Markus Schulz, Andrea Sciabà, David Smith, https://doi.org/10.5281/zenodo.6535076

[2] Alexander Held, Oksana Shadura, https://doi.org/10.22323/1.414.0235

[3] Andreas J Peters and Lukasz Janyst 2011 J. Phys.: Conf. Ser. **331** 052015

[4] The XRootD project, https://xrootd.slac.stanford.edu/

[5] L A T Bauerdick et al, 2014 J. Phys.: Conf. Ser. **513** 042044

[6] Graeme Stewart and Alaettin Serhan Mete, https://doi.org/10.5281/zenodo.2554202

[7] The ROOT framework, https://root.cern.ch/

[8] Danilo Piparo, Philippe Canal, Enrico Guiraud, Xavier Valls Pla, Gerardo Ganis, Guilherme Amadio, Axel Neumann and Enric Tejedor, 2019 EPJ Web Conf., **214**, 06298

[9] RNTuple Virtual Probe Station, https://github.com/jblomer/iotools

[10] Jakob Blomer, Philippe Canal, Axel Naumann, and Danilo Piparo, 2020 EPJ Web Conf., **245**, 02030

[11] Eduardo Rodrigues *et al*, 2020 EPJ Web Conf., **245**, 06028

[12] Nicholas Smith, Lindsey Gray, Matteo Cremonesi, Bo Jayatilaka, Oliver Gutsche, Allison Hall, Kevin Pedro, Maria Acosta, Andrew Melo, Stefano Belforte and Jim Pivarski, 2020 EPJ Web Conf., **245**, 06012

[13] Vincenzo Eduardo Padulano, Enrico Guiraud, Andrii Falko, Elena Gazzarrini, Enrique García García and Domenic Gosein, First implementation and results of the Analysis Grand Challenge with a fully Pythonic RDataFrame, these proceedings

[14] A. Tumasyan *et al.* (CMS Collaboration) 2013 Phys. Lett. **131**, 041801

[15] Tommaso Tedeschi, Vincenzo Eduardo Padulano, Daniele Spiga, Diego Ciangottini, Mirco Tracolli, Enric Tejedor Saavedra, Enrico Guiraud and Massimo Biasotto, arXiv:2307.12579 (to be published)