Analysis Grand Challenge benchmarking tests on selected sites

David Koch[1,*], *Thomas* Kuhr[1], *Günter* Duckeck[1], and *Nikolai* Hartmann[1]

[1]Ludwig-Maximilians-Universität München

Abstract. A fast turn-around time and ease of use are important factors for systems supporting the analysis of large HEP data samples. We study and compare multiple technical approaches.
This article will be about setting up and benchmarking the Analysis Grand Challenge (AGC) [1] using CMS Open Data. The AGC is an effort to provide a realistic physics analysis with the intent of showcasing the functionality, scalability and feature-completeness of the Scikit-HEP Python ecosystem.
We will present the results of setting up the necessary software environment for the AGC and benchmarking the analysis' run time on various computing clusters: the institute SLURM cluster at LMU Munich, a SLURM cluster at LRZ (WLCG Tier-2 site) and the analysis facility Vispa [2], operated by RWTH Aachen.
Each site provides slightly different software environments and modes of operation which poses interesting challenges on the flexibility of a setup like that intended for the AGC.
Comparing these benchmarks to each other also provides insights about different storage and caching systems. At LRZ and LMU we have regular Grid storage (HDD) as well as an SSD-based XCache server and on Vispa a sophisticated per-node caching system is used.

1 Introduction

With the high-luminosity LHC coming up in 2029, more data than ever will be recorded. More recorded data naturally corresponds to more data that needs to be processed in order to perform physics analyses. This poses several new challenges for analysis software. On the one hand there is the obvious problem of pure computing power: what is the fastest and most efficient way to process these vast amounts of data? On the other hand, it is equally important to design the interfaces of analysis software in such a way that physicists can use them effectively with ease of use and fast turnaround times.

The Analysis Grand Challenge (AGC) aims to specify possible workflows for a physics analysis. The reference implementation [3] of the AGC is developed as part of the IRIS-HEP project and written in Python, using services and libraries from the Scikit-HEP [4] ecosystem with `coffea` [5] as the main framework. This effort allows to test and showcase the use of these tools at a large scale [1].

*e-mail: david.koch@physik.uni-muenchen.de

The analysis performed within the AGC is a semileptonic $t\bar{t}$-analysis. It includes a simple 1-lepton event selection, a top-quark reconstruction using combinatorics, a cross-section measurement and on-the-fly evaluation of some systematic uncertainties.

It runs on data that are part of the CMS Open Data [6] release, making it ideal for reproducible benchmarks, as there is no experiment-specific affiliation required to access the data. At the moment, the dataset has a total size of 3.44 TB which corresponds to 948×10^6 events that need to be processed.

The AGC supports a variety of workflows and pipelines, like streaming the data over the data transformation service ServiceX [7]. We however focus solely on a workflow that consists of reading the data in the form of ROOT files over the network and distributing the workload using Dask [8]. We implemented a minor change in the code that allows us to flexibly switch between job-scheduler backends using `dask-jobqueue`.

In this work, the AGC is used to perform integration tests and benchmarks on three analysis facilities located in Germany.

2 Benchmark tests

Analysis facilities come with quite diverse combinations of available hard- and software. To gain a more general picture on the usability and scalability of the techniques used in the AGC, we deployed it on different sites, ran the data-processing part of the pipeline and measured various metrics (see 2.2).

2.1 Sites

We deployed and ran the AGC on the following sites:

LMU The site at the Ludwig-Maximilians-Universität in Munich is an institute cluster consisting of one 20 core node and several desktop computers with 2-8 CPUs per machine available for distributed computing. The available job-scheduler is SLURM. The data are read via XRootD [9] from the nearby ATLAS `LOCALGROUPDISC` at LRZ. The network capacity to LRZ is 1 GBit/s per node.

LRZ LRZ is a WLCG ATLAS Tier-2 site in Munich. The job-scheduler here is also SLURM. Data are stored on regular HDD based grid storage as well as on a SSD based XRootD proxy cache (XCache) server. Here every node has a 10 GBit/s network connection.

Vispa Vispa [2] is an analysis facility operated by RWTH Aachen. It provides a web-based terminal, code editor and jupyter hub [10]. The job-scheduler used here is HTCondor [11]. Data are stored locally on site-owned SSDs and read via NFS.

2.2 Measurements

The version of AGC deployed at the time (`v0`) basically performs three tasks: metadata acquisition[1], data reading and subsequent processing and plotting. For the sake of these benchmarks we only consider the first two steps, as they are the steps in the pipeline that run distributed.

Representing the distribution of computing tasks across multiple workers on a vertical axis and time on a horizontal axis, the workload can be graphically represented like in fig. 1.

[1]By now `v2` has been released in which the metadata acquisition has been removed.

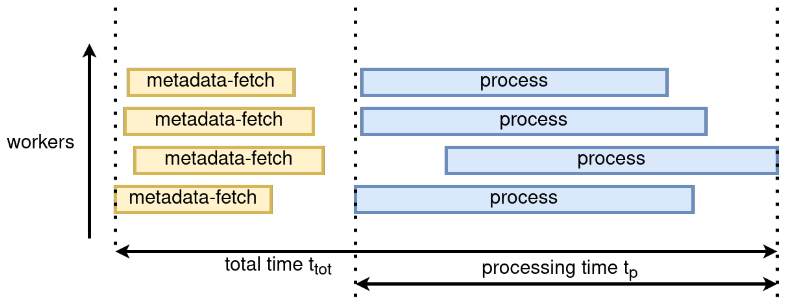


Figure 1. Graphical representation of the distribution of the workload across multiple workers and the performed tasks.

First, the task of fetching all required metadata gets performed. Only after it is done, the processing of the data can begin. Apart from the total run time t_{tot}, different partial run times can be directly measured: the total processing time t_p, which is the total time it takes from the first processing job to begin until the last processing job finishes, and the sum of all process times t_{p_i} across all workers. The first two run times can be measured directly by the process which spawns the jobs while the cumulative time can be retrieved via `coffea`'s tooling. The time it takes to fetch the metadata can then be calculated by subtracting the process time from the total run time. Note though that the result will also include time in which Dask accumulates results and prepares the start of the processing jobs, so it measures not exactly only the time it takes to acquire the metadata.

Before each measurement it was ensured that more than 90% of the requested workers were ready. This waiting time is *not* included in the measurements.

These measurements were repeated on all sites with an increasing number of workers to observe the scaling behavior. For each number of workers, the same benchmark was run at least twice on each site. The mean and standard deviation are taken as the resulting run times.

At LRZ, the benchmark was performed first with XCache disabled and then twice with XCache enabled. Running it multiple times is useful when benchmarking workloads that contain intensive caching as it is expected that subsequent runs with a full or partially full cache will run faster. It was explicitly ensured that the cache did not contain any data before the first run.

As an additional metric to measure the efficiency of the job-scheduler, we define the overhead factor f as the ratio of the total process time t_p and the average process time per worker $\bar{t}_{p,w}$:

$$f = \frac{t_p}{\bar{t}_{p,w}} = \frac{t_p}{\sum_i t_{p_i}/n} \tag{1}$$

where $\sum t_{p_i}$ is the sum of the run times of all processing tasks across all workers and n is the number of workers.

In cases where the workload is perfectly split up between the workers, each worker runs jobs for the same amount of time without in-between idle time, therefore the average run time for processing jobs per worker would be equal to the total processing time and thus $f = 1$. If however the same amount of work is distributed unevenly and therefore some workers are idling (see fig. 2), the total processing time increases while the average process time per worker stays the same, thus we get $f > 1$.

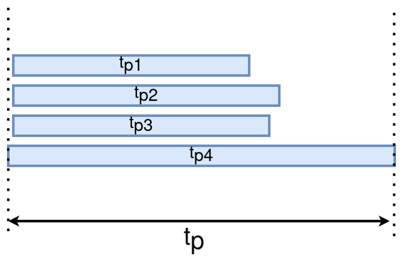


Figure 2. Illustration of inefficiently scheduled jobs: one job keeps running while all other workers are already done, idling.

2.3 Results

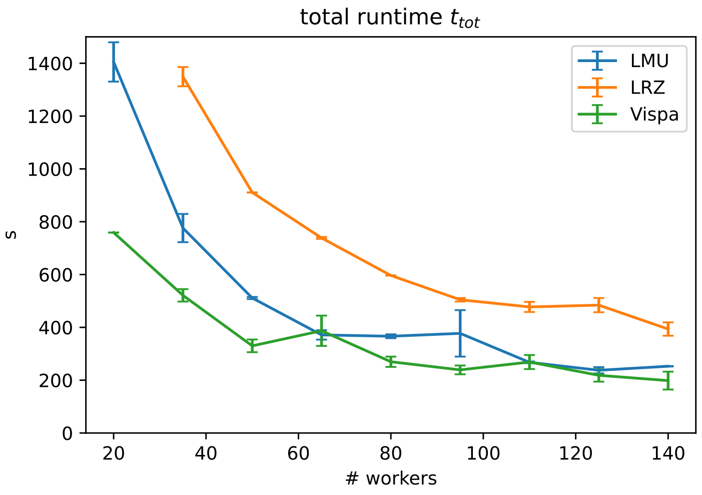
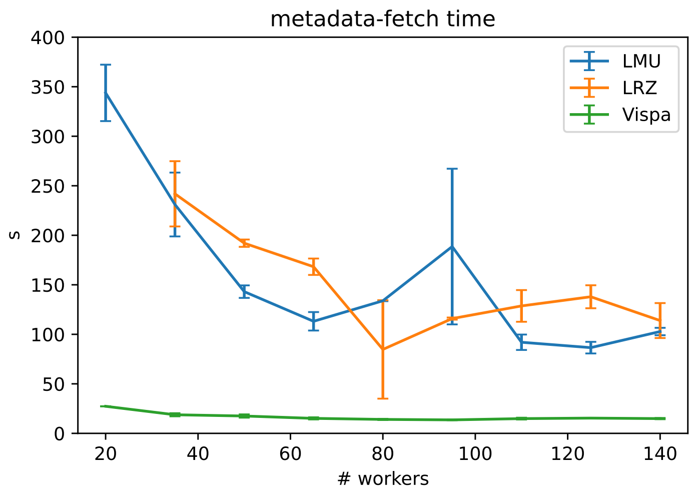


Figure 3. Total run time with varying number of workers on all sites

The scaling of the total run time with a growing number of workers (see fig. 3) behaves as expected with a $1/n$ dependency. However there is quite a large offset at $200 - 400$ s, meaning the analysis cannot be ran in a shorter amount of time on these sites, independent of the number of workers. The general scaling behavior is consistent on all sites, however it can be seen that on LRZ, the benchmarks run slower than on the other two sites. This could be due to the fact that some nodes on LRZ are already quite old.

Splitting up the total run time into the run time of the subtasks metadata fetching and processing is shown in fig. 4. Acquiring metadata does not require any computationally expensive task and is thus expected to be I/O intensive. It is evident that a fast network connection or even accessing the data locally benefits tasks like this greatly. This can be seen on the upper plot of fig. 4: on Vispa, data are available on local SSDs and need not be read via the XRootD protocol, therefore the time to fetch metadata is significantly faster here, saving up to several minutes. On LRZ and at LMU, the time to acquire metadata roughly scales



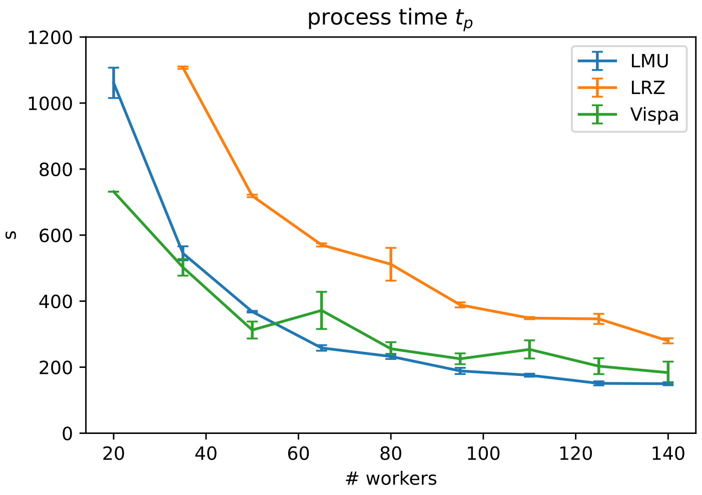


Figure 4. Metadata acquisition time and process time. The total run time consists of the sum of the two.

like $1/n$ although with large fluctuations. This can be attributed to varying occupancy of the available bandwidth by other users and is out of our control.

The time it takes to actually download and process the data is shown on the lower plot of fig. 4. The distribution closely resembles that of the total run time and seems to dominate its shape.

From these measurements, the overhead factor f (see eq. 1) was calculated and plotted in fig. 5. It can be seen that it is well above the desired value of $f = 1$, indicating a large inefficiency in the way the jobs are scheduled. Interestingly, the overhead is mostly constant for varying numbers of workers. These large values for f can be traced back to occasional jobs with exceptionally long execution times on a small number of workers while other workers were idling (see fig. 2 for an illustration of this effect). While the root cause of this problem

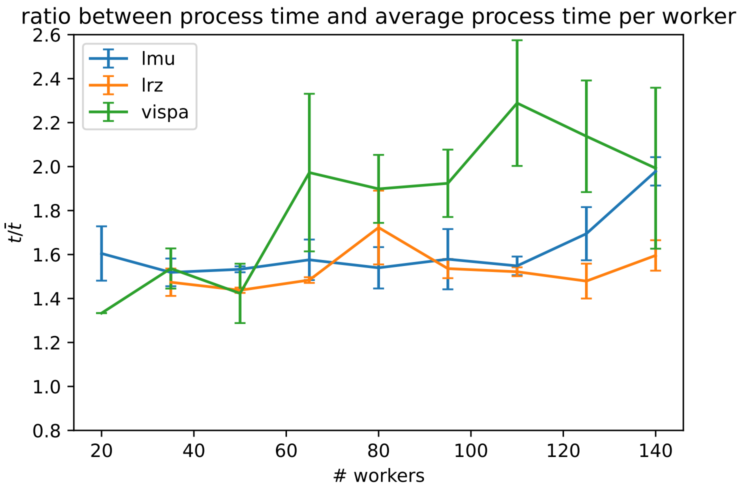


Figure 5. Overhead factor f. At $\sim$ 1.6, meaning an inefficiency of over 50%, it becomes clear that there is room for improvement.

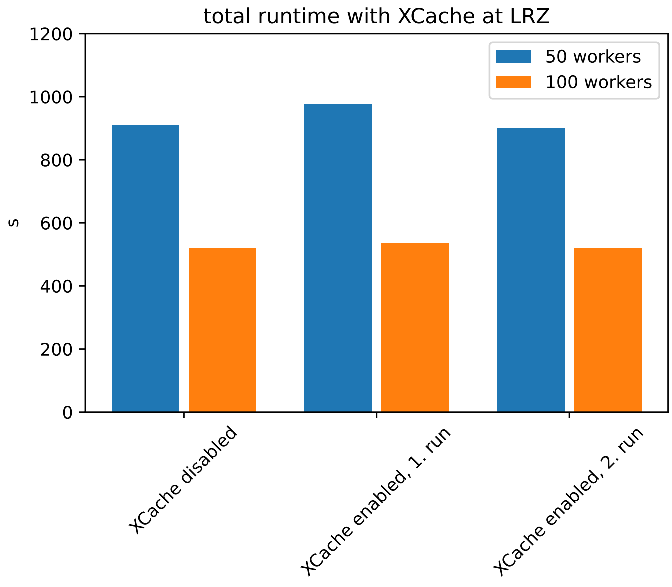


Figure 6. Total run time at LRZ with XCache enabled and disabled, running on 50 and 100 workers.

is yet to be investigated, we suspect the issue to be in Dask. Note that the overhead factor is a relative quantity and as such does not reflect the absolute performance of a site. Vispa for example has an overhead factor furthest away from 1 while still performing best in terms of runtime because it has faster processors and benefits greatly from a rapid acquisition of metadata.

Lastly, a comparison of total run times at LRZ with and without XCache enabled is shown in fig. 6. The figure shows measured times when running with 50 and 100 workers with XCache disabled, then a first run with an enabled but empty XCache and finally a second run with the cache enabled. A tiny increase in run time can be observed for the first run using an empty cache which can be explained by the overhead that consists of moving the data to the cache. That increase vanishes on subsequent runs with a full or partially full cache. However, no improvement in run time compared to the setup without XCache can be observed. As caches in general can only speed up processes that are limited by I/O we conclude that this specific task is in fact not limited by I/O.

3 Outlook

The AGC proves to not only be a powerful showcase of novel analysis workflows but also a useful tool to benchmark and test analysis facilities in the context of a realistic physics analysis at scale.

At the sites LMU, LRZ and Vispa we observed a scaling of run times inverse to the number of workers. The tests however suffer from a large overhead of 2-3 minutes, showing room for significant improvement.

We measured the overhead factor and found it to be rather high due to a few jobs with significantly longer run time. The cause of this is yet to be investigated.

Lastly we measured the effect of SSD based caching on the run time of the analysis. No reduction of the run time was observed, indicating that the analysis at hand is hardly limited by I/O operations. In future studies it would be interesting to apply alternative analysis algorithms with high I/O load in order to observe the effect of caches.

Additionally, one should investigate what the analysis is limited by, and what causes the observed offset in scaling.

References

[1] *Analysis grand challenge documentation*, https://agc.readthedocs.io/en/latest/, accessed: 2023-09-15

[2] Erdmann, Martin, Fischer, Benjamin, Geiger, Lukas, Geiser, Erik, Noll, Dennis Daniel Nick, Rath, Yannik Alexander, Rieger, Marcel, Schlüter, Felix, Schmidt, David Josef, Urban, Martin et al., EPJ Web Conf. **214**, 05021 (2019)

[3] *Analysis grand challenge source code*, https://github.com/iris-hep/analysis-grand-challenge/tree/main/analyses/cms-open-data-ttbar, accessed: 2023-09-15

[4] E. Rodrigues et al., EPJ Web Conf. **245**, 06028 (2020), 2007.03577

[5] L. Gray, N. Smith, A. Novak, P. Fackeldey, B. Tovar, Y.M. Chen, G. Watts, I. Krommydas, *coffea* (2023), https://github.com/CoffeaTeam/coffea

[6] *CMS open data guide*, accessed: 2023-09-15, https://cms-opendata-guide.web.cern.ch

[7] IRIS-HEP, accessed: 2023-09-21, https://servicex.readthedocs.io/en/latest/

[8] Dask Development Team, *Dask: Library for dynamic task scheduling* (2016), https://dask.org

[9] *XRootD*, aceesed: 2023-09-21, https://xrootd.slac.stanford.edu/

[10] *JupyterHub*, accessed: 2023-09-21, https://jupyterhub.readthedocs.io/en/stable/index.html

[11] HTCondor Team, *HTCondor*