Going fast on a small-size computing cluster

Niclas Steve Eich, Martin Erdmann, Svenja Diekmann, Manfred Peter Fackeldey, Benjamin Fischer, Dennis Noll, Yannik Alexander Rath

RWTH Aachen University, Physics Insitute III A

E-mail: `peter.fackeldey@cern.ch`

Abstract. Fast turnaround times for LHC physics analyses are essential for scientific success. The ability to quickly perform optimizations and consolidation studies is critical. At the same time, computing demands and complexities are rising with the upcoming data taking periods and new technologies, such as deep learning. We present a show-case of the HH→bbWW analysis at the CMS experiment, where we process $\mathcal{O}(1-10)$TB of data on 100 threads in a few hours. This analysis is based on the columnar NanoAOD data format, makes use of the NumPy ecosystem and HEP specific tools, in particular Coffea and Dask. Data locality, especially IO latency, is optimized by employing a multi-level caching structure using local file storage and on-worker SSD caches. We process thousands of events simultaneously within a single thread, thus enabling straightforward use of vectorized operations. Resource intensive computing tasks, such as GPU accelerated DNN inference and histogram aggregation in the $\mathcal{O}(10)$GB regime, are offloaded to dedicated workers. The analysis consists of hundreds of distinctly different workloads and is steered through a workflow management tool ensuring reproducibility throughout the development process up to journal publication.

1. Introduction

Typical LHC physics analyses analyze $\mathcal{O}(1-10)$TB of recorded and simulated data. These large amounts of data allow new and more precise scientific findings. On the other hand, they come with the burden of new computational challenges. The reduction of analysis turnaround times is essential for scientific success, as more optimization cycles during development and consolidation studies during peer review processes can be performed. Additionally, analysts profit from increased throughput, balancing the time spent on computation and the scientific thought process requiring computational runtimes measured in minutes. In the upcoming data taking periods, the situation complicates as the integrated luminosity increases to $3000\,\mathrm{fb}^{-1}$.

This article shows how fast turnaround times for modern LHC analyses can be achieved on relatively small computing clusters. First, the software and hardware environment is described in detail. Then the data flow through the computing cluster is explained, including crucial aspects such as the multi-level caching structure are discussed. In the end, a benchmark measurement is shown, highlighting the runtime reduction achieved by a custom SSD caching strategy.

2. Experimental Setup

2.1. Software Environment

This analysis is purely based on python software. Especially the NumPy [1] ecosystem is leveraged for vectorized array processing, training of deep neural networks (DNN), and data visualization. HEP specific libraries, mainly developed by the Scikit-HEP [2] community, are also used, in particular coffea [3], uproot [4], boost-histogram [5], and awkward-array [6]. Transparent up-scaling of the analysis to a HTCondor [7] batch system is achieved through dask [8] and dask-jobqueue [9, 10]. The offloading of resource-intensive DNN evaluations to other machines is done by TensorFlow-Serving [11].

These libraries are installed together with python 3.8 [12] using the conda package manager from Anaconda [13].

2.2. Computing Cluster - VISPA

The computing resources of the VISPA [14] computing cluster have been used for this work. This cluster consists of 10 heterogeneous computing nodes. Seven of these nodes, the so-called small-workers, provide 64 GB of random access memory (RAM), eight logical CPU cores, and two GPUs with a video RAM (VRAM) of 8 GB each. Two nodes, the so-called medium-workers, have 192 GB of RAM, 64 logical CPU cores and three GPUs with 16 GB of VRAM each. The last worker, named large-worker, supplies 384 GB of RAM, 64 logical CPU cores, and three GPUs with 24 GB of VRAM each. The small-workers are additionally equipped with a solid-state drive (SSD) of 1 TB each, while the other three hold an SSD of 4 TB each.

Besides these 10 workers, an additional machine (portal) exists, used for interactive computations and the submission to the workers through the HTCondor batch system.

Storage, provided through a network file system (NFS), is linked to the portal and the small-workers with a bandwidth of $1\,\mathrm{Gbit\,s^{-1}}$ and the other three with a bandwidth of $4\,\mathrm{Gbit\,s^{-1}}$. In total the NFS storage accumulates to 96 TB, striped across six different 16 TB hard drive disks (HDD). This storage is managed by a Logical Volume Manager (LVM) and accelerated by an 1 TB SSD cache.

3. Data Flow

The flow of recorded and simulated data through the VISPA computing cluster can be divided into five steps and is shown in Fig. 1.

Fig. 1 can be read from left to right, following the black arrows, which denote the flow direction. In general, the processing is performed in the so-called "MapReduce" fashion. Filters and transformations are mapped onto multiple events (chunks) and then reduced into a single output. The five processing steps are explained in the following.

1. Data and simulation, in the NanoAOD data format [15], are stored on the NFS in the ROOT [16] file format. Dask workers fetch via network approximately 10^5 events, a so-called chunk, from the NFS storage. A pre-step calculates the start and stop event number, which is used to determine the chunk.

2. The NFS is backed by a 1 TB SSD LVM cache in order to speed up the access of frequently accessed ("hot") data. Here, copies of blocks of hot data are stored and can be read a lot faster from the SSD than from the HDDs. The blocks are stored with a granularity of 832 kB. The cache behavior does not follow a first-in-first-out (FIFO) policy but the so-called multi-queue (MQ) [17] policy. This cache behaves entirely transparent for the user.

3. Read chunks are cached on the SSD cache on the machines of the HTCondor cluster, on which they are processed. In contrast to the LVM cache, these SSD caches follow the FIFO cache eviction policy. A robust assignment of chunks to workers is achieved by a custom submission policy encoded into the Dask scheduler. The cached columns are stored in blocks of 4 kB. A more detailed description can be found in section 4. These on-worker SSD caches also behave completely transparent in usage for the user.

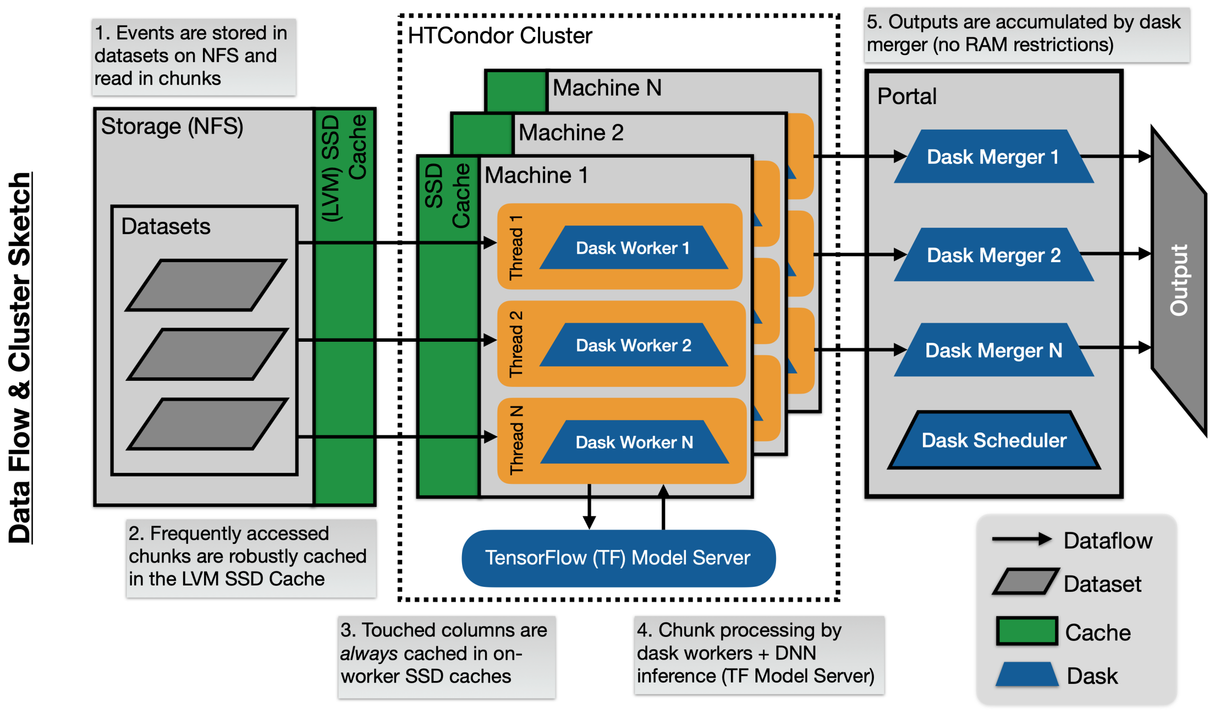
Data Flow & Cluster Sketch
1. Events are stored in datasets on NFS and read in chunks
Storage (NFS)
(LVM) SSD Cache
Datasets
2. Frequently accessed chunks are robustly cached in the LVM SSD Cache
HTCondor Cluster
Machine N
Machine 2
Machine 1
SSD Cache
Thread 1
Dask Worker 1
Thread 2
Dask Worker 2
Thread N
Dask Worker N
TensorFlow (TF) Model Server
3. Touched columns are *always* cached in on-worker SSD caches
4. Chunk processing by dask workers + DNN inference (TF Model Server)
5. Outputs are accumulated by dask merger (no RAM restrictions)
Portal
Dask Merger 1
Dask Merger 2
Dask Merger N
Dask Scheduler
Output
Dataflow
Dataset
Cache
Dask

Figure 1. A sketch of the VISPA computing cluster including the storage (NFS), the portal machine, the TensorFlow model server and the HTCondor cluster with all workers. The data flow is briefly explained in five steps.

4. The Dask workers process chunks of events in parallel. In a very first step, data is decompressed using the IO functionality of the uproot package and interpreted into awkward arrays. Vectorized operations on these chunks, using NumPy and awkward-array, leverage modern single instruction, multiple data (SIMD) instructions on the modern CPUs of the workers. After applying analysis-specific event filters and reconstructions, e.g., a deep neural network (DNN) is evaluated. This GPU- and memory-intensive evaluation is offloaded to a dedicated external server, which simultaneously serves all Dask workers.

5. Finally, the processed chunks can be accumulated into different storage types, such as arrays or histograms. Dedicated Dask workers run on the portal machine, the so-called Dask mergers, are used for this accumulation. These Dask workers have no RAM limitation as the accumulation of, e.g., highly dimensional histograms can be very memory intensive. The final output is pickled, compressed and stored on the NFS storage.

4. Caching Strategy

A deterministic assignment between chunks and workers is needed for the efficient and robust use of the on-worker SSD caches. This assignment is determined analytically using the L1-distance between the hashes of chunks and workers in a periodic N-dimensional space:

$$\text{L1 distance} = \sum_{i}^{N} min(|c_i - w_i|, 1 - |c_i - w_i|), \tag{1}$$

where c denotes a chunk coordinate and w a worker coordinate in the N dimensional space. This is visualized in Fig. 2 for two dimensions.

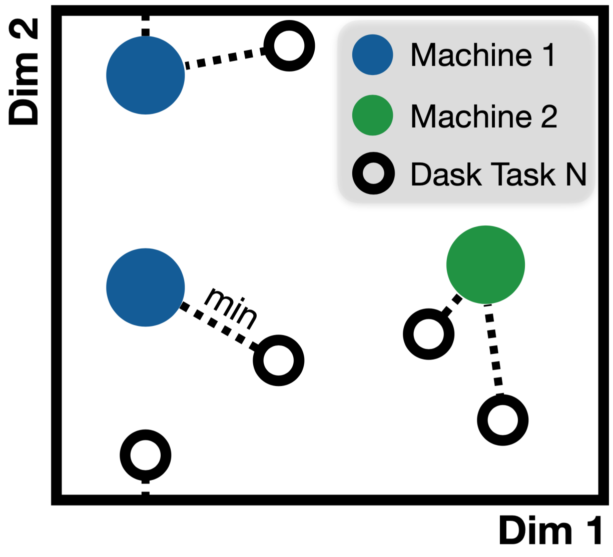

Figure 2. Sketch of the Dask task to machine assignment policy for a two dimensional space.

The coordinates of the chunks, or rather Dask tasks, are pictured as black rings, whereas the ones of the machines are blue/green circles. The assignment is realized by determining the minimal L1 distance between pairs of chunks and machines. Due to the significant difference in the amount of machine and chunk hashes, the worker hashes are replicated to balance this difference. Thus a smooth degradation of caching performance is obtained since the N-dimensional space, and the coordinates do not alter under hardware changes or the addition/removal of chunks. This policy does not depend on user information, thus automatically enables cache sharing between multiple users. This deterministic assignment is implemented into the scheduling policy of the Dask scheduler.

5. Benchmark Results
The performance of the on-worker SSD caches using the custom deterministic assignment described in section 4 is measured for a dataset of roughly 1 TB of NanoAOD simulation containing approximately 4.18×10^9 events. For this benchmark, 386 GB uncompressed simulations have been read.

The benchmark consists of multiple consecutive runs, of which the first has empty on-worker SSD caches. Each consecutive run fills the on-worker SSD caches. This is shown in Fig. 3 for five consecutive runs.

The blue bars depict the compressed data, which still needs to be read from NFS, that is not yet stored on the on-worker SSD caches. The black data points show the runtime in minutes. It can be seen that the runtime reduction follows the same trend as the reduction of data reading from the NFS as both show an almost perfect overlay. In total, a runtime improvement of a factor of 1.5 is achieved from the first to the fifth run. This measurement is limited by the CPU intensive decompression of the LZMA compression algorithm, which is the default NanoAOD compression.

The improvement comes gradually as a work-stealing algorithm breaks the custom deterministic caching assignment when certain conditions are fulfilled, such as a Dask worker is out of work and can take tasks from other Dask workers.

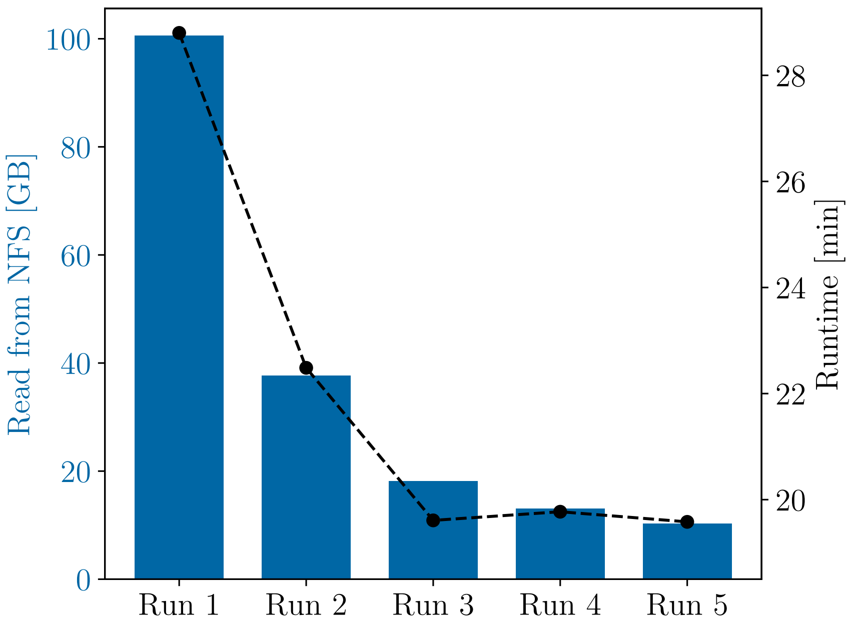

Figure 3. Performance results of on-worker SSD caches for five consecutive runs.

6. Conclusion

This article describes a custom SSD caching strategy, which reduces time spent in subsequent IO operations. A benchmark, using the small-scale VISPA computing cluster, has been performed. It shows an improvement of a factor of 1.5 in runtime reduction for IO operations, while vectorized array operations allow overcoming CPU-bound analysis. Numerous more analysis cycles become possible, resulting in more studies during analysis developments and peer-review processes.

Small-scale computing clusters can play a significant role in the current and upcoming LHC physics analysis, making them a highly competitive choice for analysts.

References

[1] Harris C R *et al.* 2020 *Nature* **585** 357–362 URL `https://doi.org/10.1038/s41586-020-2649-2`
[2] Scikit-hep website `https://scikit-hep.org` accessed on 12.01.2022
[3] Gray L *et al.* 2021 Coffeateam/coffea: Release v0.7.11 URL `https://doi.org/10.5281/zenodo.5762406`
[4] Pivarski J *et al.* 2021 scikit-hep/uproot4: 4.1.9 URL `https://doi.org/10.5281/zenodo.5767911`
[5] Schreiner H *et al.* 2021 scikit-hep/boost-histogram: Version 1.2.1 URL `https://doi.org/10.5281/zenodo.5548612`
[6] Pivarski J *et al.* 2022 scikit-hep/awkward-1.0: 1.8.0rc1 URL `https://doi.org/10.5281/zenodo.5828686`
[7] HTCondor Team 2021 HTCondor URL `https://doi.org/10.5281/zenodo.5750673`
[8] Dask Development Team 2016 *Dask: Library for dynamic task scheduling* URL `https://dask.org`
[9] dask-jobqueue source code `https://github.com/dask/dask-jobqueue` accessed on 12.01.2022
[10] dask-jobqueue blog entry `https://blog.dask.org/2018/10/08/Dask-Jobqueue` accessed on 12.01.2022
[11] Olston C, Fiedel N, Gorovoy K, Harmsen J, Lao L, Li F, Rajashekhar V, Ramesh S and Soyke J 2017 Tensorflow-serving: Flexible, high-performance ml serving (*Preprint* `1712.06139`)
[12] Van Rossum G and Drake F L 2009 *Python 3 Reference Manual* (Scotts Valley, CA: CreateSpace) ISBN 1441412697
[13] 2020 Anaconda software distribution URL `https://docs.anaconda.com/`
[14] Bretz H P *et al.* 2012 *Journal of Instrumentation* **7** T08005–T08005 URL `https://doi.org/10.1088/1748-0221/7/08/t08005`

[15] Ehatäht, Karl 2020 *EPJ Web Conf.* **245** 06002 URL `https://doi.org/10.1051/epjconf/202024506002`
[16] Brun R and Rademakers F 1997 *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **389** 81–86 ISSN 0168-9002
new Computing Techniques in Physics Research V
URL `https://www.sciencedirect.com/science/article/pii/S016890029700048X`
[17] Yuanyuan Zhou James F Philbin K L 2001 The multi-queue replacement algorithm for second level buffer caches URL `https://static.usenix.org/event/usenix01/zhou.html`