Computing and Software for Big Science (2023) 7:3
https://doi.org/10.1007/s41781-023-00095-9

ORIGINAL ARTICLE

Fast Columnar Physics Analyses of Terabyte-Scale LHC Data on a Cache-Aware Dask Cluster

Niclas Eich[1] · Martin Erdmann[1] · Peter Fackeldey[1] · Benjamin Fischer[1] · Dennis Noll[1] · Yannik Rath[1]

Received: 19 July 2022 / Accepted: 21 February 2023

Abstract
The development of an LHC physics analysis involves numerous investigations that require the repeated processing of terabytes of data. Thus, a rapid completion of each of these analysis cycles is central to mastering the science project. We present a solution to efficiently handle and accelerate physics analyses on small-size institute clusters. Our solution uses three key concepts: vectorized processing of collision events, the "MapReduce" paradigm for scaling out on computing clusters, and efficiently utilized SSD caching to reduce latencies in IO operations. This work focuses on the latter key concept, its underlying mechanism, and its implementation. Using simulations from a Higgs pair production physics analysis as an example, we achieve an improvement factor of 6.3 in the runtime for reading all input data after one cycle and even an overall speedup of a factor of 14.9 after 10 cycles, reducing the runtime from hours to minutes.

Keywords Data analysis · Scaling · Cache · High-throughput computing

Introduction

Obtaining new physics results from LHC collider data at CERN ranks among the most challenging tasks in data analyses. Although the raw data of the experiments are processed centrally and quantities such as particle tracks are reconstructed, numerous tasks remain for small analysis teams to achieve a concrete scientific result. An example is the cross-sectional measurement of the Higgs boson production.

When developing a physics analysis, many different studies need to be performed. Examples include data-driven background estimations, efficiency measurements, training and evaluation of multivariate methods, and determination of systematic uncertainties. All of these studies typically require multiple processing of at least a significant portion of the data and simulations. In addition, analyses are subjected to an experiment-internal peer review process, which requires numerous further consolidation studies. Consequently, every data analysis is inevitably subjected to a large number of iterations.

✉ Peter Fackeldey
peter.fackeldey@rwth-aachen.de

[1] Physics Institute 3A, RWTH Aachen University, 52056 Aachen, Germany

Two challenges are of central importance: first, to perform the analysis in a reproducible manner, we rely on a workflow management system for data analysis [1]. Second, the turnaround time of each analysis cycle is critical to making progress. In the best case, the runtime of the analysis cycle harmonizes with the reflection phase, in which the physicist decides on the next action.

The duration of an analysis cycle has considerably increased due to the very successful LHC operation and the associated growth of recorded data. Typical analyzed data volumes are in the order of terabytes (TB). Without further developments in analysis technology, the prospect of the LHC upgrade for high luminosities will again significantly prolong analysis cycles. Three key concepts are exploited in this work to compensate this increase and improve the runtime of an analysis cycle.

The first concept tackles the way of processing events. While classically collision events are analyzed one after another, vectorized array operations can process multiple events simultaneously. The scientific Python ecosystem NumPy [2] provides these vectorized array operations using the processor-specific "single instruction multiple data" (SIMD) instruction sets.

Second, the programming paradigm "MapReduce" [3] is a key concept for this project. Operations, such as selection and reconstruction, are mapped to subsets of collision

Published online: 20 March 2023

events. Their partial output is then accumulated (reduced) to a single output. Software packages such as Dask [4] orchestrate this paradigm on any computing cluster.

The third key concept is caching. Caching increases the efficiency of repeated data access. Here, we present a caching mechanism that caches collision data on processor-near solid-state disks (SSDs). Subsequent analysis cycles benefit from this and show a substantial reduction in cycle time.

For the first two key concepts, there are already established software solutions, e.g., NumPy and Dask, that aim at accelerating computationally intensive operations by means of parallelization. With this speedup we uncovered a new limitation that is addressed by the third key concept: we noticed that the time spent on IO operations, especially transferring the collision data to the processors, accounts for a non-negligible portion of the total runtime.

This paper presents an IO bound benchmark leveraging the third key concept for speeding up analysis cycles. For the first two concepts, we employ the coffea [5] and Dask software packages, which are used for reading, decompressing, and interpreting NanoAOD columns, and an affine job-to-worker orchestration, respectively. As resources, we use the computing cluster of the VISPA project, which provides cloud services for scientific data analysis ([6] and references therein). For the third concept, we have substantially extended the computing cluster with solid-state storage disks, and developed a worker–job affinity mechanism to most efficiently utilize these disks. As an application example, we present data reading benchmarks using simulated collision events of a Higgs pair production analysis.

This work is structured as follows. We describe the upgraded VISPA platform, quote the software components, explain the caching, and conduct a quantitative survey on the runtime reduction for multiple consecutive analysis cycles.

VISPA Hardware and Software Systems

Cluster Setup

The setup used in the presented analysis is a small-scale computing cluster that is optimized for scientific data analysis and deep learning applications (Fig. 1).

It features various service nodes as well as three different sizes of worker nodes, which differ mostly in the processors (CPUs), the RAM capabilities, the graphics processing units (GPU), and their network connections. The service node vispa-portal is used for interactive working and the management of the batch system (see Sect. 2.2). It possesses a CPU with 64 logical cores and 128 GB RAM. The seven worker nodes have a combined CPU capacity of 224 logical cores and possess a total of 832 GB RAM and 16 TB SSD storage for caching purposes. The detailed configurations of the individual machines can be found in Fig. 1. All used processors are capable of the AVX2 SIMD instruction set [7]. The service node vispa-portal and each worker node additionally possess a 4 TB SSD used for local storage and caching. The caching strategy and its implementation is explained in Sect. 2.3.

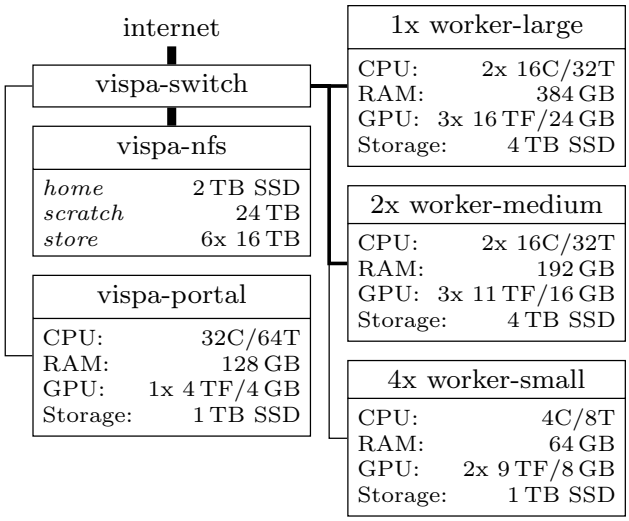

Fig. 1 Hardware setup of the used cluster. A total of ten different nodes are used of which three nodes are for service and seven nodes are utilized as worker nodes. Central processing units (CPU) are specified with their number of cores (C) and the number of threads (T). The capabilities of the graphic processing units (GPU) are expressed in their floating-point performance (FP32) and memory (VRAM). Network connections are drawn by lines, whereas their width corresponds to the provided bandwidth (1/4/l Gbit/s)

The switch (vispa-switch) is the central node of the local network. It is connected to the internet and the storage service node (vispa-nfs) with 10 Gbit/s, to the large and medium workers with 4 Gbit/s, and to each small worker node and the service node vispa-portal with 1 Gbit/s. In addition, each node has a fully isolated out-of-band management interface.

The service node vispa-nfs provides central storage capacity for the cluster. It possesses a total of 2 TB SSD and 120 TB HDD storage. The storage can be accessed via three different network-shared file systems implemented with the Network File System (NFS) protocol (version 4.2) [8]. The file system *home* is used as the working directories of the users. It is mirrored and backed up daily by the local computing authority. In addition, two different file systems, *scratch* and *store*, can be used for larger amounts of data. The *scratch* file system, which totals to 24 TB, is mirrored and used for experimental data and intermediate results, such as pre-processed experimental data. The *store* file system, which totals 96 TB, has the purpose to store reproducible data, such as local copies of raw experimental data or software installations. Because it is striped across six different 16 TB HDDs, it features fast reading and writing. It

is, therefore, suited for data, which is accessed frequently or with a high total throughput. The total shared file system bandwidth is limited by the vispa-nfs network bandwidth of up to 10 Gbit/s.

Operating systems are deployed on the different nodes using the open-source configuration management tool Ansible [9].

The provisioning of the user's working environment is done using the open-source package management system conda [10]. It ensures the stability and maintainability of each user's working environment, even in a heterogeneous and changing computing setup, and adapts to the multiple different needs of a large user base.

Job Distribution with HTCondor

Scaling analyses to run on the entire cluster requires a solution for workload management. While small jobs can be run interactively on the vispa-portal node, larger workflows are distributed to the worker nodes using HTCondor [11].

The HTCondor setup in VISPA consists of three main parts: a scheduler, a central manager, and workers. Users submit their jobs to the HTCondor scheduler. Each of these jobs defines requirements that specify the resources and it is expected to consume. The central manager then performs the matchmaking between these requirements and the available resources of the workers.

For the analysis presented here, the workload is split into chunks that can be distributed over the cluster using Dask and Dask-Jobqueue [12, 13]. The user launches a Dask scheduler on vispa-portal, and Dask workers are spawned on the worker nodes via HTCondor jobs. The Dask scheduler then distributes chunks of the total workload to these workers. This distribution requires unrestricted communication among the Dask scheduler and the VISPA worker nodes.

SSD Caching

Modern high-energy physics analyses need to analyze data on the terabyte scale. Using vispa-nfs for reading these data from *scratch* is strongly limited by the HDDs and network connections. This limitation is alleviated using appropriate caching mechanisms, as described in the following.

The caching is facilitated for each worker by the FSCache available within the Linux kernel [14]. Once enabled for a particular NFS mount-point, it operates transparently upon all IO requests for files therein. In particular, data is cached at a page-size granularity (4 kB) which enables selective caching, i.e., of only the accessed branches of a `.root` file. Since all IO operations (read and write) fill the cache, the occurrence of cache-trashing is minimized by only caching the *store* volume, which is predominantly used for write-once read-often data. The cache is configured to store its contents on the SSDs of the workers, thus profiting from their superior data transfer rates.

Since each cache will only contain the contents of data requested by its worker at some point prior, it is of utmost importance to route such requests—or rather the jobs that cause these particular requests—in a cache-hit maximizing manner. This is done through a worker–job affinity mechanism, where each worker and job is identified in a reproducible manner. The identifier consists of the hostname and a salt for workers, and the input file UUID and the range of event numbers for jobs. The salt for workers is a number $\in [1..8]$, such that each worker has eight identifiers. This is required to get more homogeneous assignments and migration patterns, since the number of workers is rather small.

These identifiers are then uniformly mapped into a 64-dimensional bounded space. There, their coordinates are inferred from their cryptographic hash value (i.e., SHA512) by interpreting it as a vector of integers (i.e., $[0..255]^{64}$). For any pair of such values, a distance D can be calculated as such: $D(\mathbf{a}, \mathbf{b}) = \sum_i d(|a_i - b_i|)$ where $d(x) = \min(x, 256 - x)$. Each job is then assigned to the closest worker, ensuring a reasonably even distribution. This mechanism is sketched in Fig. 2 for a two dimensional space.

The assignment is not strict, allowing idle workers to steal jobs from busy workers. This concept is referred to as work-stealing. It avoids trailing jobs due to heterogeneous job runtimes, thus improving the overall runtime. Especially, in

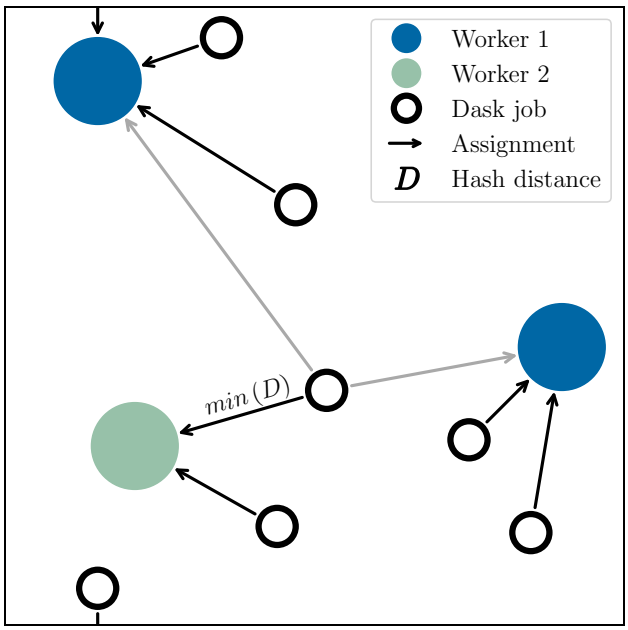

Fig. 2 Two-dimensional sketch of the worker–job affinity mechanism. The Dask job in the middle shows all possible assignments to the available workers. The assignment mechanism selects the smallest hash distance $min(D)$ (black arrow), and rejects other possible (gray arrows) assignments

the case of the addition or removal of workers, the affected jobs are redistributed homogeneously while avoiding the reallocation of all other jobs. In addition, the allocation ratio of jobs between workers can be changed smoothly by including a worker-specific distance factor—which is used to equalize the workload despite the varying processing power of all the workers. Multiple users can participate and profit from the data caching using the same files and affinity mechanism. The presented mechanism is independent of how the read data are further processed; it only relies on the input data.

Performance Benchmark

The performance of the VISPA computing cluster with on-worker SSD caching is measured for a subset of simulated data sets in the NanoAOD data format [15]. In total, the read data amounts to 1439 GB, which corresponds to the event information of 1.05×10^9 events in 120 columns of the NanoAOD file, including information from leptons, jets, generator particles, and various event-level quantities; no further processing, such as performing selections or reconstructions, is done with these events. Our benchmark is designed to be limited by the time spent in IO operations but at the same time as close as possible to the data reading of a realistic Higgs pair production analysis. All data sets are compressed with the level ten Zstandard compression algorithm [16], which has been changed from NanoAOD's default compression to reduce the decompression time. Our benchmark consists of multiple consecutive cycles. Throughout each cycle, 221 Dask workers carry out the processing with one thread and 1.5 GB RAM each. The Dask worker requirements are the same for each worker node. Figure 3 shows the performance benchmark for ten cycles.

The key message is that the runtime decreases substantially for the first few cycles. In total the improvement amounts to a factor of 14.9. The amount of data that is still read from the vispa-nfs is vastly reduced as more data is read from the on-worker SSD caches. This effect converges for later cycles until almost all data are cached directly in the on-worker SSDs. The close overlay of runtime and the amount of data, which is still read from the vispa-nfs, show a strong correlation between runtime reduction and caching. The cache usage gradually converges to a maximum since a work-stealing mechanism minimizes each cycle's runtime at the cost of slight degraded deterministic cache usage. Once the cache utilization maximizes, the CPU usage is practically 100% of which most is spent on decompression. In these scenarios, cache-independent overhead times from job

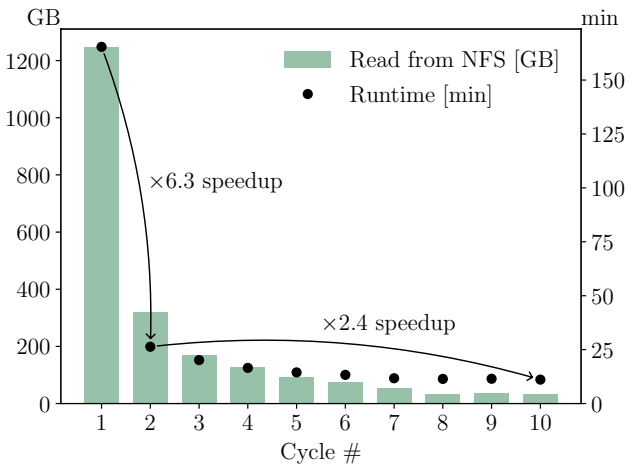

Fig. 3 Performance benchmark results for ten consecutive cycles

scheduling (≈ 1 min) and trailing jobs (≈ 3 min) become a non-negligible portion of the total runtime.

Conclusion

Modern LHC physics analyses need to deal with a large amount of recorded data, while analysts and collaborations require many analysis cycles for various studies in the shortest time possible. Even physics analyses using vectorized processing of events and the MapReduce paradigm can significantly benefit from a dedicated on-worker SSD caching strategy. On the VISPA system, a small scale computing cluster, we show that a consequent caching strategy significantly reduces the time spent in IO operations. In the scope of a real-world Higgs pair production analysis, our benchmark shows a speedup of a factor of 14.9 for the 10th analysis cycle, moving from a few hours to approximately ten minutes. This allows numerous more analysis cycles and diminishes the IO limitation of large-scale analysis projects.

The caching strategy described in this paper allows for overcoming IO bottlenecks of modern LHC physics analyses, enabling small-scale computing clusters to become a competitive choice for interactive, flexible, and high-performance physics analyses.

Acknowledgements This work is supported by the Ministry of Innovation, Science, and Research of the State of North Rhine-Westphalia, and by the Federal Ministry of Education and Research (BMBF) in Germany. N.E. gratefully acknowledges the support of the Deutsche Forschungsgemeinschaft.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability The simulated data used to demonstrate the new method serve only as an example of a general large data set, which is

subject to analysis. The exact data used in this study is therefore not publicly available.

References

1. Marcel R et al (2017) Design and execution of make-like, distributed Analyses based on Spotify's Pipelining Package Luigi. arXiv:1706.00955 [physics.data-an]
2. Harris Charles R et al (2020) Array programming with NumPy. Nature. https://doi.org/10.1038/s41586-020-2649-2
3. Jeffrey D, Sanjay G (2004) "MapReduce: Simplified Data Processing on Large Clusters". In: OSDI'04: Sixth Symposium on Operating System Design and Implementation. San Francisco, CA, pp. 137–150
4. Dask Development Team (2016) Dask: Library for dynamic task scheduling. https://dask.org. Accessed 26 May 2022
5. Lindsey G et al (2021) CoffeaTeam/coffea: Release v0.7.11. Version v0.7.11. https://doi.org/10.5281/zenodo.5762406
6. Martin E et al (2019) "Evolution of the VISPA-project". In: Forti A et al (eds) EPJ Web Conf. 214. p. 05021. https://doi.org/10.1051/epjconf/201921405021
7. Intel. Advanced Vector Extensions (2022) https://www.intel.de/content/www/de/de/architecture-and-technology/avx-512-overview.html. Accessed 26 May 2022
8. Haynes Thomas (2016) Network File System (NFS) Version 4 Minor Version 2 Protocol. RFC 7862. https://doi.org/10.17487/RFC7862.https://www.rfc-editor.org/info/rfc7862
9. Red Hat (2022) Ansible. https://www.ansible.com. Accessed 26 May 2022
10. Anaconda Software Distribution. Version 2.4.0. (2020) https://docs.anaconda.com/. Accessed 26 May 2022
11. HTCondor Team. HTCondor. Version 9.4.0. (Dec. 2021) https://doi.org/10.5281/zenodo.5750673
12. dask-jobqueue source code. https://github.com/dask/dask-jobqueue. Accessed 26 May 2022
13. dask-jobqueue blog entry. https://blog.dask.org/2018/10/08/Dask-Jobqueue. Accessed 26 May 2022
14. Kernel Linux (2022) General Filesystem Caching. https://www.kernel.org/doc/html/latest/filesystems/caching/fscache.html. Accessed 26 May 2022
15. Karl Ehatäht (2020) "NANOAOD: a new compact event data format in CMS". In: EPJ Web Conf. 245, p. 06002. https://doi.org/10.1051/epjconf/202024506002
16. Yann C, Murray K (2021) Zstandard Compression and the 'application/zstd' Media Type. RFC 8878. https://doi.org/10.17487/RFC8878.https://www.rfceditor.org/info/rfc8878