AMBER Experiment's Online Filter System for Virtualised IT Infrastructure

Martin Zemko, Dominik Ecker, Vladimir Frolov, Stephan Huber, Vladimír Jarý, Igor Konorov, Josef Nový, Benjamin Moritz Veit, Miroslav Virius

Abstract—**High-energy physics experiments require significant computing resources to operate their high-level trigger systems. Typically, these systems are constructed as extensive computing farms with cutting-edge expensive hardware to provide sufficient computing power. Usually located on-site, these systems process detector data in real-time and minimize their latency. In this paper, we present an alternative high-level filter system specifically designed for the AMBER experiment at CERN. The novelty of our approach lies in its high efficiency, which eliminates the need for a dedicated on-site computer farm. Instead, it makes use of existing shared resources housed in the CERN data center. The proposed system efficiently handles the data generated by the medium-sized experiment and performs numerous parallel filtering tasks in an online fashion. All system components operate within a shared, fully virtualized environment, including databases, storage, and processing units. This flexible environment scales effectively, allowing adjustments to allocated resources based on agreements with service managers. We present the architectural design and the implementation of such a system. To demonstrate its capabilities, we have conducted various measurements assessing its performance, latencies, and stability under maximum (expected) loads. These results demonstrate the resilience and reliability of the filtering system while optimizing overall costs to a minimum.**

Index Terms—**Data acquisition, Data handling, High energy physics computing, Software performance, Readout systems, Parallel processing**

I. Introduction

THIS paper introduces a high-throughput filtering system for the AMBER experiment at CERN. The experiment's primary goal is to study strong interactions across a broad spectrum of four-momentum transfers and includes various research objectives. Initially, it focuses on measuring the cross-section of antiproton production, which is relevant for dark matter search. Subsequently, the experiment will study the proton radius by using elastic muon-proton scattering. Additionally, in its initial phase, the experiment will explore Drell-Yan and charmonium production utilizing conventional hadron beams. The following section compares our filtering system with other state-of-the-art systems utilized in different experiments.

II. Streaming acquisition system

In recent years, the progress in detector and acquisition technologies allowed us to read and process digitized data without any prior data reduction based on conventional L1 triggers. This progress led to the expansion of streaming acquisition systems in high-energy physics and allowed more efficient data selection. Major experiments such as ATLAS [1], CMS [2], LHCb [3], DUNE [4], and others already developed their own free-running acquisition schemes. However, the benefits of the streaming acquisition systems are usually traded off for increased data rates and higher demands for computing resources. The usual way to deal with this problem is to collect as much data as possible, then send them to computers and filter them using software tools. Such systems typically comprise at minimum two layers: the acquisition and selection layer. In addition, fast network infrastructures are also incorporated, serving as a medium for excessive data transfers.

The AMBER acquisition system operates on similar principles. It employs fast FPGA electronics in the acquisition layer, completely taking the task of sorting and merging data, followed by high-performance computing for data reduction. This hybrid scheme consisting of hardware and software processing, as depicted in Fig. 1, combines the advantages of both worlds.

We leverage the full potential of FPGA technology for data multiplexing and event building. The compact nature of FPGA modules reduces space requirements and costs. Furthermore, hardware-based acquisition offers rapid recovery and high reliability, as FPGA modules only require a quick reset signal to return to their initial state.

On the other hand, advanced software frameworks allow the employing of more elaborate and parametrized filtering algorithms. These systems are also more flexible and accessible by non-expert users, allowing them to participate and contribute to the development process without extensive knowledge of the underlying machinery and to achieve the ultimate goal of data reduction by about 100.

III. Data protocol

The new streaming architecture of the AMBER experiment builds upon the triggered COMPASS iFDAQ acquisition system [5]. In the triggered scheme, a subset of detectors

Manuscript submitted for review May 20, 2024.

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic (grant LM2023040), Charles University (grant PRIMUS/22/SCI/017), and the Grant Agency of the Czech Technical University in Prague (grant SGS23/190/OHK4/3T/14).

M. Zemko, V. Jarý, J. Nový, and M. Virius are with the Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague

D. Ecker, S. Huber, and I. Konorov are with Technische Universität München.

B. Veit is with the Johannes Gutenberg-Universität Mainz.

V. Frolov is with JINR.

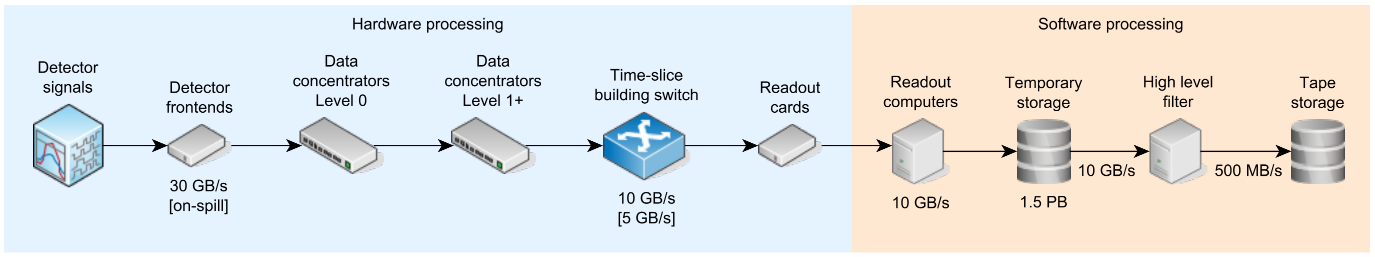


Fig. 1. Flattened structure of streaming acquisition system, relying on hardware processing in FPGAs followed by readout and filtering procedures implemented in software.

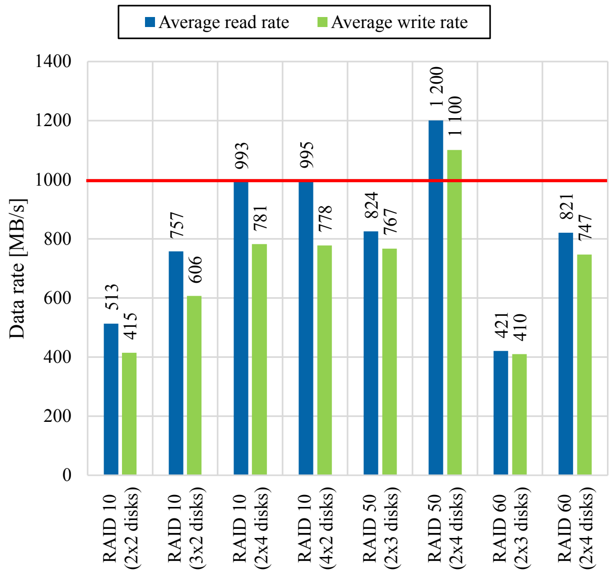


Fig. 2. Data throughput of various nested RAID configurations consisting of 8 or less disks.

generates triggers to initiate data readout. However, in the triggerless scheme, triggers are replaced with a clock signal, enabling each detector to operate at its own frequency. This approach allows fast and slow detectors to be installed in the same setup without a negative impact on each other.

We developed a multi-layered streaming protocol to unify the data format, focusing on reliability and efficiency while minimizing overhead. Its hierarchical structure can be observed in Fig. 3. This protocol is designed to include only essential data fields, reducing overall bandwidth usage. Additionally, we employ multiple levels of CRC-32 checksums to guarantee data integrity. To synchronize the data, we insert precise timestamps into each data packet. The frequencies at which detectors read out data are chosen carefully to align perfectly with time slice edges, using integer divisors of the global time slice signal.

The basic data units of this protocol are called time slices. They capture fixed periods of collected data, fully covering the entire time domain with data tiles – time slices. Since each detector has different time precision, time slices are further subdivided into finer units called images. The time slice length remains constant for the entire detector setup, in the order of hundreds of microseconds. However, the image length varies based on detector type, reflecting detector response times. To address any misaligned data at slice edges, the last image from the previous slice is always duplicated to the next slice. Later, during data filtering, time slices are treated as independent processing units.

IV. Streaming Acquisition

The data path begins with the frontend electronics formatting digitized values into this streaming protocol. Outgoing formatted data traverse through several layers of data multiplexers. These devices consolidate multiple links into fewer ones, optimizing the link bandwidth usage. Moreover, multiplexers contain large DDR3 buffers to distribute spill data peaks lasting 4.8 seconds [20] across the entire duty cycle of the SPS accelerator (at least 14.4 seconds), suppressing spikes on downstream devices. Eventually, the data stream reaches the timeslice builder, the device responsible for an event-building equivalent in streaming systems. It reorganizes data in such a way that related incoming time slices are serialized into a single output link. Both input and output links employ the 8b/10b Aurora protocol with native flow control to signalize backpressure to upstream multiplexers. With a maximum throughput of 5 GB/s, the builder's processing rate can be doubled by adding another parallel unit. [6]

The builder forwards data directly to readout cards known as spillbuffers, serving as the interface between hardware and software processing. These FPGA-based modules are commercially available Nereid Kintex 7 XC7K160T PCIe boards equipped with 4 GB DDR3 memory for buffering. Each spillbuffer card ensures a stable throughput of 1.6 GB/s via PCIe-2.1 interface to readout computers [6]. As for readout hosts, we utilize cost-effective Supermicro A+ 1014S-WTRT servers featuring AMD Epyc 7313 CPU and 64 GB DDR4 memory, capable of handling over 1 GB/s data rate per host according to our measurements [9].

Each readout host is connected to Promise VTrak J5800S external disk storage via Broadcom MegaRAID® SAS 9580-8i8e RAID controller. The storage chassis accommodates 24 Toshiba MG07ACA14TE hard drives, each offering a transfer speed of 248 MiB/s [7]. We observed that parallel access to spinning disks degrades their performance; therefore, we divided disks into three virtual drives – one for reading, one for writing, and one for idling. This approach avoids parallel access and ensures a smooth transition between volumes. The operations are rotated every 30 minutes to ensure equal disk wear.

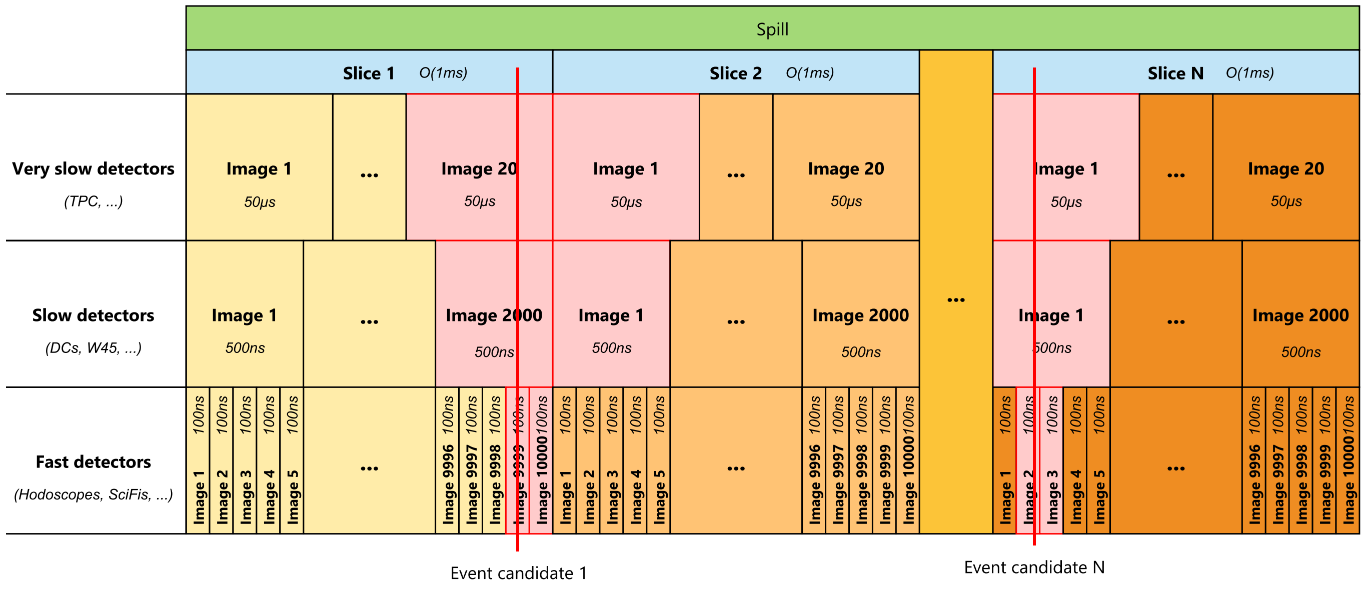


Fig. 3. Fundamental concept of data reduction in streaming DAQ is based on preserving images containing possible event candidates. We store two consecutive images for each event candidate. The remaining images are removed. [6]

After thorough benchmarking of various RAID configurations as illustrated in Fig. 2, we opted for the RAID 50 setup consisting of 2 spans per 4 disks. Our measurements show the average read rate of 1.2 GB/s and write rate of 1.1 GB/s for this particular configuration. Overall, these on-site buffers provide the capacity of more than 1 PB, sufficient to store up to three days of data-taking, seamlessly bridging weekend periods with limited support.

For further data processing, we rely on the virtualized infrastructure provided by the CERN laboratory, which is the primary goal of this paper. Data from local buffers are transmitted via 2×100 Gbps links to the data centre to the generic Exabyte-scale Open Storage (EOS) instance operated by CERN. We allocated 500 TB of the EOS storage to serve as our remote buffers. Due to its distributed design, EOS limits the ingestion rate of a single data stream to 1 Gbps, resulting in an effective throughput of approximately 100 MB/s per connection. This rate is not sufficient to comply with our requirements; therefore, we use 24 parallel streams to achieve the minimum threshold of 1 GB/s. To effectively manage large data transfers, we utilize a custom transfer service called AMBER Data Recording (ADR) tool, which is based on the XRootD transfer protocol [21] and incorporates other control mechanisms such as checksums and data validation.

Simultaneously, as the data are transferred to the EOS storage, ADR inserts metadata into the MySQL database hosted on the CERN Database-on-demand (DBOD) platform. This service offers virtualized databases running in Kubernetes clusters. In addition, users may benefit from many useful features such as remote monitoring, automatic backups, and replication [8]. In our acquisition system, we use these features to enhance the stability and reliability of the database. Furthermore, we adopted a clustered database design comprising three instances – primary database and two replicas – with automatic failover to replicas if the primary database crashes.

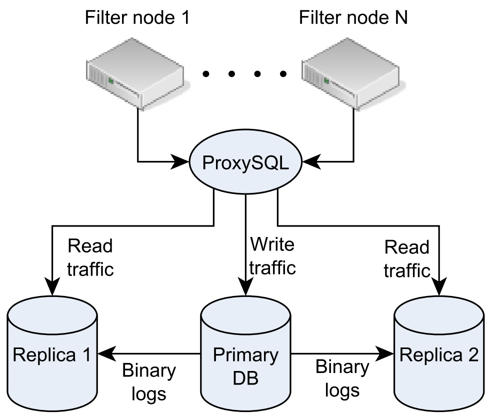


Fig. 4. Structure of the database cluster running on DBOD service providing traffic load-balancing, automatic backups, and recovery.

The deployed database cluster is depicted in Fig. 4.

V. High-level filter

Once the data files are transferred to the EOS remote buffers and logged into the database, we begin the filtering process. In our streaming acquisition system, we reduce the amount of data by removing certain images. We use identifiers of slices and images to calculate the exact timestamp of each hit and identify potential event candidates. Only the images that match these candidates are preserved, while those that do not match any are discarded. This helps reduce the size of the slice data, as shown in Fig. 3. The data reduction is performed by the High-Level Trigger (HLT) system, an extensive framework designed for efficient data manipulation and filtering. The HLT system includes two main components: the filtering framework and the filter management system.

A. Filtering framework

The filtering framework is a Qt-based software responsible for numerous data processing tasks, including decoding, partial track reconstruction, and data reduction. To handle these tasks efficiently, we have developed multiple libraries corresponding to specific steps, such as protocol parsing, detector alignment, database access, and state machines. These libraries are optimized for performance and dynamically linked to the main application.

The application implements dynamic multithreading, where it detects the number of CPU cores available and creates an equal number of processing threads. Moreover, on systems with multiple CPUs, it utilizes a non-uniform memory access (NUMA) approach known as siloing. This method isolates resources, avoiding data exchange between processors and crossing NUMA domains. This means that threads can only access their local memory, which improves the overall throughput. [9]

The fundamental principle of the filtering process is based on eliminating images without any physics events. Initially, we categorize images into two distinct groups: primary images, sourced from detectors actively involved in making filter decisions, and secondary images. Primary images undergo decoding and in-depth analysis to identify event candidates. Secondary images are filtered, but have no impact on the final decision.

To optimize the decoding process, we define a small set of primary detectors. Since track reconstruction relies solely on these primary images containing incomplete detector information, the resulting reconstructed tracks provide an estimate of real particle trajectories. We refer to this process as a partial track reconstruction.

When analysing primary images, the processing pipeline involves two main steps as illustrated in Fig. 5:

1) **Time alignment** (preprocessing): processing of timestamps using a shifted wavelet tree algorithm to detect hit bursts [10].
2) **Spatial analysis** (filtering): analysing of hit positions and particle trajectories to detect physics events with interesting signatures.

The algorithms included in these steps are modular and exchangeable, allowing for various implementations developed for specific physics cases. Moreover, the filtering pipeline can accommodate several parallel data paths that can be combined into a single binary decision.

After completing the analysis pipeline, we obtain a list of event candidates indicating potential physics events. Using this list, we conduct data reduction by eliminating images that do not intersect with any event candidate. For each event candidate, we always retain two consecutive images to address potential edge cases near image borders.

B. Distributing application

The filtering application is distributed as an executable file together with all its dependencies and delivered through the CernVM-FS file system. CVMFS is a service managed by CERN, designed to ensure smooth and reliable distribution of software. It was specifically created to deploy software across large computing grids that handle extensive data processing tasks.

Internally, CVMFS uses a content-addressable storage system and Merkle trees. Files are distributed solely through HTTP connections, which helps circumvent most firewall and network issues. Furthermore, the file system automatically verifies and maintains data consistency using various hashing algorithms. [11]

Our filtering system fully relies on this distribution channel and its caching capabilities. CVMFS can efficiently handle numerous connections without any throttling or bottlenecks, making it the perfect storage solution for our application, which runs thousands of instances simultaneously. Additionally, CVMFS is mounted as a read-only file system in user space via a FUSE module, making it seamlessly integrated with the filtering application.

VI. Filter management system

The second key component of the HLT system is the management platform, which handles the submission, execution, and control of filtering jobs. This platform is split into backend and frontend parts, both interacting with the HLT database as illustrated in Fig. 6. Despite using the same database, the frontend and backend components do not communicate directly. This separation allows them to operate independently on different platforms and services. Regarding the database, the management system depends again on the DBOD service as its primary metadata source.

A. Filtering backend

The HLT backend, developed in Python, ensures the management and submission of filtering jobs. It directly communicates with HTCondor, a high-throughput computing platform operated by CERN. HTCondor efficiently distributes computing tasks to connected machines, making it highly scalable and capable of handling thousands of hosts [12]. At CERN, users share a single HTCondor instance, competing for approximately 100,000 CPU cores [13]. To ensure fairness, HTCondor implements an advanced system of priorities and quotas.

During operation, HLT regularly checks the queue table for filtering requests. When a new request arrives, the backend generates a job definition and submits it to HTCondor using its Python API. This interface enables faster communication compared to regular system calls, which use plain text for exchanging information. However, submit files are still created and stored in the database for debugging purposes and for users who wish to submit filtering jobs manually.

Upon submission, HTCondor appends the request to its global queue, where it waits for an execution on a worker node. Each job's lifecycle is managed through a predefined set of states in the form of the state machine. HLT states are defined as a superset of HTCondor states, allowing state mirroring and synchronization between both systems. This synchronization happens every 2 minutes and involves only active jobs; those in final states are no longer synchronized.

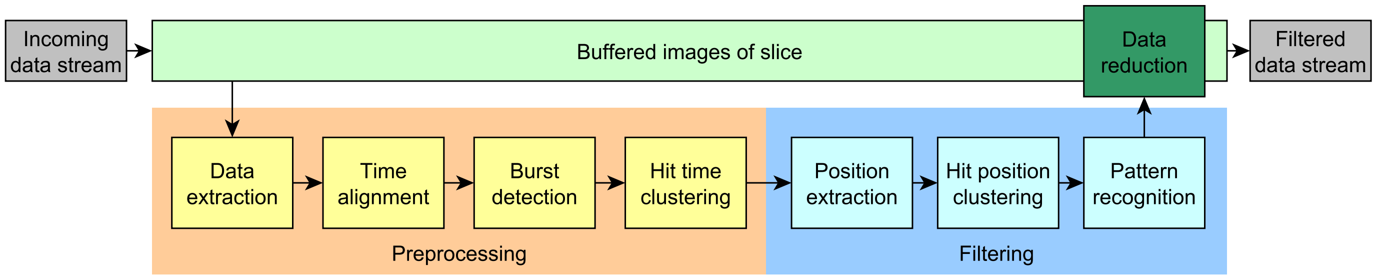


Fig. 5. Example of the filtering algorithm performed by the AMBER HLT system. The preprocessing and filtering stages can be adjusted or replaced according to the physics case.

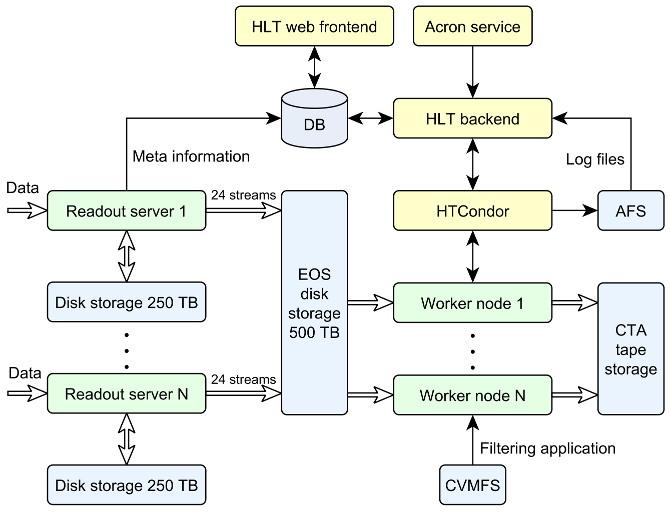


Fig. 6. Architecture of the AMBER HLT system, fully relying on shared services provided by CERN. On-site hardware is limited to readout servers and buffers, with all other services being virtualized and managed by CERN.

After a job is finished, HTCondor removes it from the queue. Then, the HLT backend reacts to this change and retrieves the job's log file from the Andrew File System (AFS). This is because HTCondor faces a global limitation in the EOS storage service that restricts frequent updates to small log files [16]. To overcome this issue, HTCondor utilizes the AFS file system as a temporary storage for log files. After job completion, HLT transfers these files from AFS back to EOS, following the recommendation of HTCondor administrators [13]. By examining these logs, the backend identifies the job's result and looks for any potential issues. If there are any errors, the job is resubmitted again for the second attempt. Eventually, the job's status is updated in the database to reflect the final result.

Although HTCondor offers a history overview of completed jobs, we chose not to utilize this feature due to its relatively low performance and high latencies. Our measurements indicate that a single query to the HTCondor history file takes approximately 6.44 ± 1.43 seconds. This result varies significantly depending on factors such as HTCondor settings, size of the history file, system and storage performance, among others. Thus, querying the history file has not proven suitable for us, considering the system must poll thousands of jobs every two minutes.

The HLT backend operates on a shared Linux terminal service, known as LxPlus, which comes pre-equipped with all necessary packages and libraries for submitting HTCondor jobs. Due to its small footprint, HLT does not negatively impact other users. Unlike typical daemons, the backend is not continuously active. Instead, it is periodically triggered by the CERN Acron service, offering cron-like scheduling capabilities with an added authentication layer. This service allows users to create crontab entries for Kerberos-authenticated cron jobs using the privileges of the authenticated user [14]. Acron effectively addresses authentication requirements when submitting jobs to HTCondor. In contrast, the PanDA production system developed by the ATLAS collaboration relies on a more complex setup involving virtual organizations authenticated with SSL certificates and OIDC tokens [15].

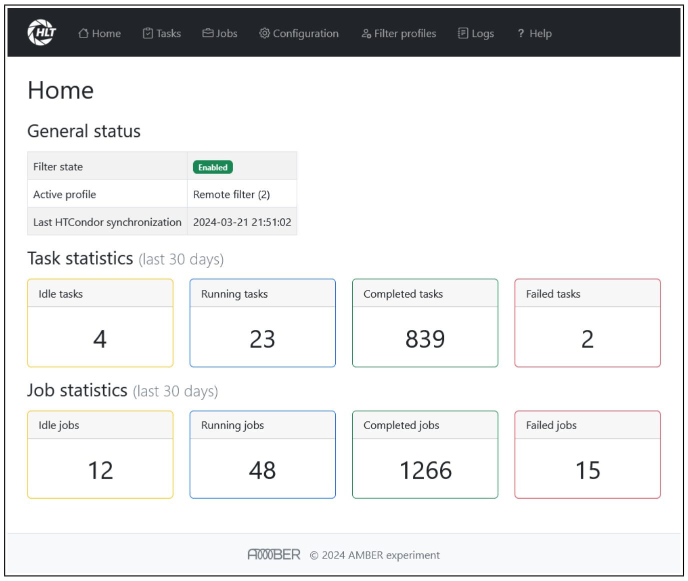
Regarding the output data files generated by HLT, we utilize the capability of worker nodes to send data to the CERN Tape Archive (CTA). More precisely, output files are transferred to another EOS instance (CTA-EOS) that is directly connected to the tape archive. Files stored on this special EOS instance are automatically backed up onto tapes, ensuring secure archival of filtered data for subsequent physics analysis.

In summary, the HLT backend performs tasks similar to the PanDA production system developed at ATLAS [17]. However, we found PanDA to be too complex and resource-intensive for our specific use case. With HTCondor as our primary target platform at CERN, we were able to significantly optimize the overall architecture and performance of the filtering system.

B. *Web frontend*

As its second main component, HLT offers a web interface – the frontend. This interface allows users to manage and control the filtering system. It gives a clear picture of filtering jobs and their states. Users are even allowed to submit their own filtering requests that are handled and executed by the filter management system. Jobs are grouped into tasks, typically corresponding to individual runs, making it easier to monitor them all together. The web interface also makes it easy to check and retrieve logs of completed jobs as they are automatically copied to EOS.

The web interface runs on the CERN Web Services, offering a convenient web hosting platform for users. It is accessible from both the CERN network and the internet, and is protected by CERN Single Sign-On, allowing HLT operators and shifters



Home

General status

Filter state	Enabled
Active profile	Remote filter (2)
Last HTCondor synchronization	2024-03-21 21:51:02

Task statistics (last 30 days)

Idle tasks	Running tasks	Completed tasks	Failed tasks
4	23	839	2

Job statistics (last 30 days)

Idle jobs	Running jobs	Completed jobs	Failed jobs
12	48	1266	15

AMBER © 2024 AMBER experiment

Fig. 7. Web frontend interface provides an illustrative overview of the current system status and allows user to fully control and configure the filtering system.

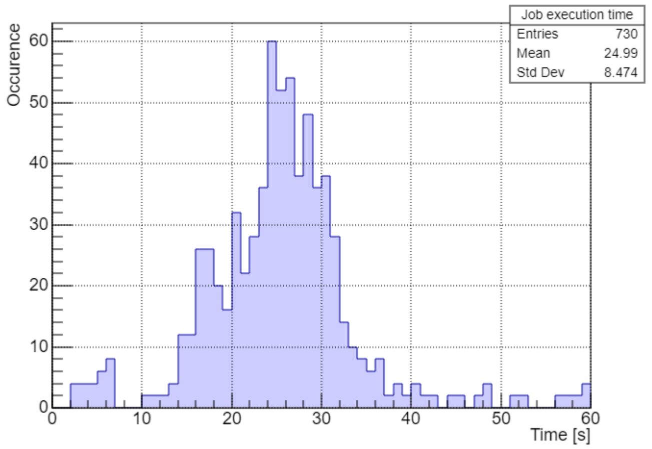


Fig. 8. Processing time distribution represents the net execution time for 1 GB files, each containing around 250,000 time slices.

to use it from anywhere. Additionally, it employs advanced access control lists based on user roles defined in the CERN LDAP directory [18].

Built with PHP 8 and the Bootstrap framework, the frontend scripts are hosted on EOS, with the database serving as the main source of information. Since the frontend does not process any raw data, it is very fast and responsive, offering quick access to a large number of users.

VII. Performance measurement

The filtering system depicted in the previous paragraphs has undergone extensive testing and validation during the initial phase of the AMBER experiment in 2023. We confirmed that our system can be scaled horizontally, primarily due to the scalable nature of the underlying HTCondor platform and other computing services available at CERN. These tests demonstrated that the HLT backend is able to manage over one thousand concurrent jobs effectively.

To gain a comprehensive understanding of how CERN's shared services operate, we aimed to evaluate and assess the latencies within the system. Our initial measurement focused on the processing time of individual filtering jobs. We performed tests using more than 1,000 data files, each 1 GB in size. Each file was processed by a single job utilizing 4 CPU cores. On average, filtering a single data file took 24.99 seconds with a standard deviation of 8.47 seconds. The distribution of measured times is shown in Fig. 8. When we translate these values into processing rates per CPU core, we achieve an average rate of 10.24 MB/s. Considering the 10 GB/s data rate of the full detector setup [9], we estimate the need for approximately 1,000 CPU cores to perform real-time filtering.

The second measurement examined the queueing time of the HTCondor platform at CERN, specifically its version 23.0.2. Our findings revealed that the average time spent in the queue was approximately 100.80 ± 48.78 seconds, as illustrated in Fig. 9. This indicates that each job typically waits around one and a half minutes in the HTCondor queue before execution. The variability in queueing time is significant, accounting for nearly 50 % of the measured average, which suggests that queueing time is a highly volatile metric and depends on various factors. We identified four main aspects influencing the queueing time:

- **Cluster occupancy** (available resources): When more jobs are submitted by other users, they compete for limited resources. Inevitably, higher user submission rate leads to increased latency.
- **Job and worker requirements**: Jobs with stricter requirements may only be compatible with fewer workers, further restricting available resources.
- **User priority**: Users who submit numerous jobs may have their priority reduced, while those with lower submission rates receive higher priority. This approach follows the HTCondor fair-share policy [19].
- **Quotas**: Users are often restricted to a certain number of parallel jobs.

To optimize system performance, we implemented several strategies. For instance, jobs are submitted with minimal requirements to increase the chances of earlier execution and lower the latency. The filtering application supports a variable number of worker threads and efficiently utilizes all available CPU cores regardless of their number. Furthermore, in case of a significant increase in queueing time, our backup strategy aims to approach HTCondor administrators and request higher priority for our jobs.

Our last measurement investigated the total job time, incorporating delays from the HLT backend. Given that synchronization occurs every two minutes, this mechanism introduces an average extra latency of one minute. Consequently, the average job duration was measured at 151.90 ± 82.28 seconds. The resulting distribution is shown in Fig. 10.

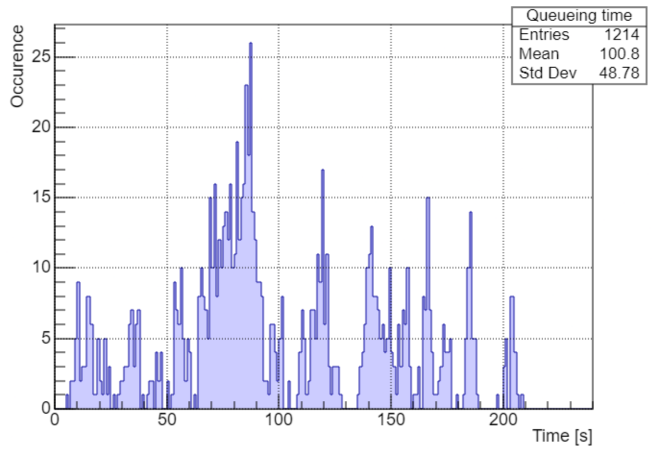


Fig. 9. Queueing time distribution in HTCondor shows highly volatile results that depend on external aspects such as cluster occupancy and user priorities.

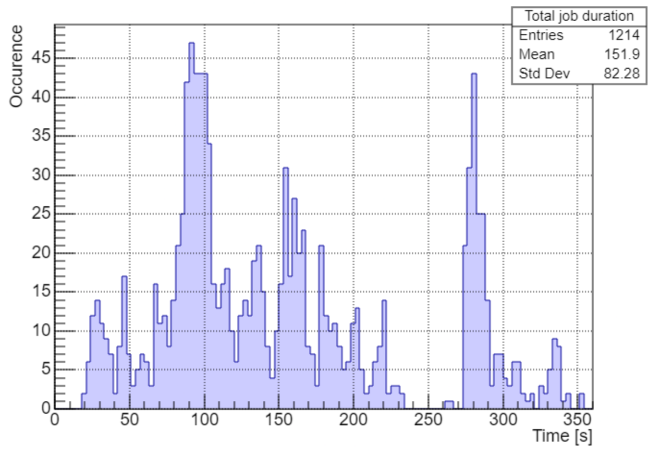


Fig. 10. Total job duration distribution gives the overall expected latency of the filtering system. It accounts for queueing latency, execution time, file transfers, and other aspects.

VIII. Conclusions

This paper introduces an extraordinary high-throughput filtering system designed for the AMBER experiment at CERN. Traditionally, high-energy physics experiments employ computing farms with costly hardware for real-time data processing. However, our system offers an alternative approach by utilising existing shared resources within the CERN data centre, eliminating the need for dedicated on-site computer farms.

Our proposed system efficiently handles the data generated by the medium-sized experiment with a throughput of 10 GB/s, performing numerous parallel filtering tasks in real time within a fully virtualized environment. Key components of the designed system include FPGA-based data acquisition, custom streaming protocols for data transmission, and the high-level filter system for data reduction.

The streaming acquisition system utilizes fast FPGA electronics for data multiplexing and event building, followed by readout cards and further processing in readout computers. Subsequently, data are reduced using a high-throughput filtering framework written in C++, ensuring partial track reconstruction and subsequent data reduction.

The automated filter management system is constructed entirely using shared services such as EOS, AFS, CVMFS, LxPlus, DBOD, and others. The system consists of a Python backend for job submission and retrieval, paired with a web frontend for user interaction and monitoring. The backend utilizes HTCondor as the underlying platform for load balancing and job distribution, whereas tasks are being executed on shared computing resources at CERN.

Performance measurements demonstrate the scalability and efficiency of the designed system, with individual filtering jobs processing data files in around 24.99 seconds on average. The queueing time in HTCondor averages around 100.80 seconds, influenced by factors such as cluster occupancy, job requirements, and user priorities. The total job duration, including queueing latency, execution time, and file transfers, averages around 151.90 seconds. These measurements indicate the system's capability to handle the high-throughput requirements of the AMBER experiment while optimizing resource utilization and reducing costs.

Moving forward, our plans involve further optimizations of the filtering system to reduce latencies and improve throughput. Improvements include implementing more detailed monitoring to identify bottlenecks. We also intend to fine-tune the HTCondor scheduler, aiming to stabilize job queueing times. Furthermore, we plan to investigate the potential of the filtering software running on GPUs, which could considerably improve the performance of the filtering system.

References

[1] The ATLAS TDAQ Collaboration, "The ATLAS Data Acquisition and High Level Trigger system", J. Inst., vol. 11, no. 06, pp. P06008–P06008, Jun. 2016, DOI. 10.1088/1748-0221/11/06/P06008.

[2] R. Covarelli, "The CMS High-Level Trigger", AIP Conference Proceedings, vol. 1182, no. 1000, pp. 188–191, 2009, DOI. 10.1063/1.3293780.

[3] R. Aaij et al., "Allen: A High-Level Trigger on GPUs for LHCb", Computing and Software for Big Science, vol. 4, no. 1, pp. 1–11, 2020, DOI. 10.1007/s41781-020-00039-7.

[4] R. Acciarri et al., "Long-Baseline Neutrino Facility (LBNF) and Deep Underground Neutrino Experiment (DUNE) Conceptual Design Report, Volume 4 The DUNE Detectors at LBNF". arXiv, Jan. 12, 2016. Accessed: Mar. 15, 2024. [Online]. Available: http://arxiv.org/abs/1601.02984

[5] D. Steffen et al., "Overview and future developments of the intelligent FPGA-based DAQ (iFDAQ) of COMPASS", Proceedings of Science, vol. Part F1285, pp. 0–3, 2016, DOI. 10.22323/1.282.0912.

[6] S. Huber et al., "Data Acquisition System for the COMPASS++/ AMBER Experiment", IEEE Transactions on Nuclear Science, vol. 68, no. 8, pp. 1891–1898, 2021, DOI. 10.1109/TNS.2021.3093701.

[7] Toshiba Electronic Devices, "MG07ACA Series - Product Specification". Jul. 2020. Accessed: Mar. 22, 2024. [Online]. Available: https://toshiba.semicon-storage.com/content/dam/toshiba-ss-v3/master/en/storage/product/data-center-enterprise/eHDD-MG07ACA-Product_Overview_rev3s.pdf

[8] A. C. Alonso, D. C. Polidura, and M. De Giorgi, "Database on demand User Guide". Aug. 15, 2023. Accessed: Mar. 22, 2024. [Online]. Available: https://dbod-user-guide.web.cern.ch/

[9] M. Zemko et al., "Triggerless data acquisition system for the AMBER experiment", in Proceedings of 41st International Conference on High Energy physics — PoS(ICHEP2022), Bologna, Italy: Sissa Medialab, Nov. 2022, p. 248. DOI. 10.22323/1.414.0248.

[10] Y. Zhu and D. Shasha, “Efficient elastic burst detection in data streams”, in Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, Washington, D.C.: ACM, Aug. 2003, pp. 336–345. DOI. 10.1145/956750.956789.

[11] J. Blomer, “CernVM-FS’s documentation”. Feb. 19, 2016. Accessed: Mar. 23, 2024. [Online]. Available: https://cvmfs.readthedocs.io/en/stable/

[12] T. Theisen, M. Coatsworth, G. Thain, and T. L. Miller, “HTCondor Version 23.5.2 Manual”. Mar. 20, 2023. Accessed: Mar. 23, 2024. [Online]. Available: https://htcondor.readthedocs.io/en/latest/

[13] B. Jones, S. Traylen, J. van Eldik, and N. Hoimyr, “Batch Docs - Batch Service Concepts”. Jul. 05, 2016. Accessed: Mar. 21, 2024. [Online]. Available: https://batchdocs.web.cern.ch/concepts/

[14] G. McCance and U. Schwickerath, “Acron documentation - Acrontab usage”. May 17, 2023. Accessed: Mar. 21, 2024. [Online]. Available: https://acrondocs.web.cern.ch/

[15] T. Maeno, “Identity and Access Management”. Jan. 06, 2021. Accessed: Mar. 21, 2024. [Online]. Available: https://panda-wms.readthedocs.io/en/latest/architecture/iam.html

[16] L. F. Alvarez, S. Traylen, N. Hoimyr, and J. van Eldik, “Batch Docs - EOS”. Jan. 16, 2020. Accessed: Mar. 22, 2024. [Online]. Available: https://batchdocs.web.cern.ch/troubleshooting/eos.html

[17] T. Maeno, “PanDA Introduction”. Mar. 01, 2021. Accessed: Mar. 21, 2024. [Online]. Available: https://panda-wms.readthedocs.io/en/latest/introduction/introduction.html

[18] P. Tedesco, H. Short, V. Brillault, and S. Lopienski, “Authorization Service - Role based permissions”. Oct. 02, 2020. Accessed: Mar. 22, 2024. [Online]. Available: https://auth.docs.cern.ch/applications/role-based-permissions/

[19] B. Jones, S. Traylen, and F. Protopsalti, “Batch Docs - Priorities in HTCondor”. Mar. 20, 2024. Accessed: Mar. 21, 2024. [Online]. Available: https://batchdocs.web.cern.ch/fairshare/fairshare.html

[20] P. Abbon et al., “The COMPASS experiment at CERN”, Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 577, no. 3, pp. 455–518, Mar. 2007, doi: 10.1016/j.nima.2007.03.026.

[21] A. Hanushevsky, “The XRootD Protocol Version 5.2.0”. National Accelerator Laboratory, Dec. 04, 2023.