EPJ Web of Conferences **295**, 02001 (2024) https://doi.org/10.1051/epjconf/202429502001
CHEP 2023

A High-Speed Asynchronous Data I/O Method for HEPS

Shiyuan Fu[1,*], *Lu* Wang[2], *Yaodong* Cheng[1,3,4], *Yu* Hu[1], *Rui* Liu[1], *Lei* Wang[1], *Shuang* Wang[1], *Jianli* Liu[1], *Haokai* Sun[1], and *Fazhi* Qi[1]

[1]Institute of High Energy Physics, CAS, 100049 Beijing, China
[2]National Science Library, CAS, 100190 Beijing, China
[3]University of Chinese Academy of Sciences, 100049 Beijing, China
[4]TIANFU Cosmic Ray Research Center, 610041 Chengdu, Sichuan, China

Abstract.
The High Energy Photon Source (HEPS) is expected to produce a substantial volume of data, lead to immense data I/O pressure during computing. Inefficient data I/O can significantly impact computing performance.
To address this challenge, firstly, we have developed a data I/O framework for HEPS. This framework consists of three layers: data channel layer, distributed memory management layer, and I/O interface layer. It mask the underlying data differences in formats and sources, while implementing efficient I/O methods. Additionally, it supports both stream computing and batch computing.
Secondly, we have designed a data processing pipeline scheme aimed at reducing I/O latency and optimizing I/O bandwidth utilization during the processing of high-throughput data. This involves breaking down the computing task into several stages, including data loading, data pre-processing, data processing, and data writing, which are executed asynchronously and in parallel.
Finally, we introduce the design of stream data I/O process. The primary objective of stream data I/O is to enable real-time online processing of raw data, avoiding I/O bottlenecks caused by disk storage. This approach ensures the stability of data transmission and integrates distributed memory management to guarantee data integrity in memory.

1 Introduction

1.1 HEPS

The High Energy Photon Source (HEPS)[1] is a significant scientific infrastructure project in China, constructed by the Institute of High Energy Physics. It has been designed to include more than 90 beamlines, with 15 beamlines planned for construction in the first phase. The HEPS will generate a substantial amount of data, as illustrated in figure 1, which will place a heavy burden on storage and computing resources. Among these beamlines, beamline 7 will generate the most data, estimated at around 250 TB per day.

*e-mail: fusy@ihep.ac.cn

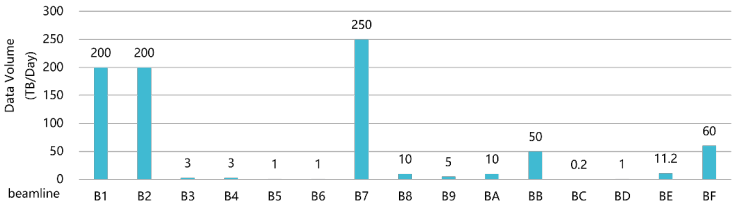


Figure 1: HEPS data volume

1.2 Data storage and formats

During the HEPS experiment, a large amount of raw data is generated, and the data processing is diverse. Therefore, the HEPS storage system is designed with three levels, as depicted in figure 2. The beamline storage has fast write speed to accommodate the demands of high-rate raw data generation. The central storage is used for data offline processing, while the tape storage is designed for long-term data storage. Upon acquisition by the DAQ(Data Acquisition) system, the data is persistently stored in files on the beamline storage via single or multiple streams, and is also directly transferred to memory for fast analysis to verify the raw data accuracy. The central storage is employed for large-scale reconstruction of raw data and can be shared across different experiments. In addition to storing the raw data, it also saves the derived data generated during processing, requiring high data readout performance. The data storage duration in the central storage is generally several months, after which the data will be transferred to tape storage.

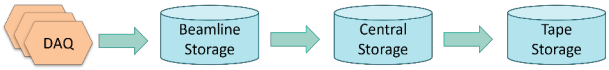


Figure 2: HEPS storage system

Therefore, the computation in HEPS is divided into two types: stream processing and batch processing. For stream processing, the data is received from the DAQ's data stream. In batch processing, the data is stored in files on disk, currently available in two formats: HDF5[2] and TIFF.

2 Architecture

The overall I/O architecture for HEPS is as shown in figure 3. This architecture primarily comprises of three layers: the data channel layer, the distributed memory management layer, and the I/O interface layer.

The data channel layer is primarily designed for stream processing and is responsible for receiving data streams from the DAQ. As HEPS includes multiple beamlines, the data channel layer is capable of efficiently handling multiple data streams simultaneously. Additionally it can perform some simple stream computing tasks.

The distributed memory management layer is mainly used for storing stream data. This is due to that certain computational tasks in HEPS requiring the complete collection of data before they can be executed. Consequently, a large amount of raw data needs to be stored

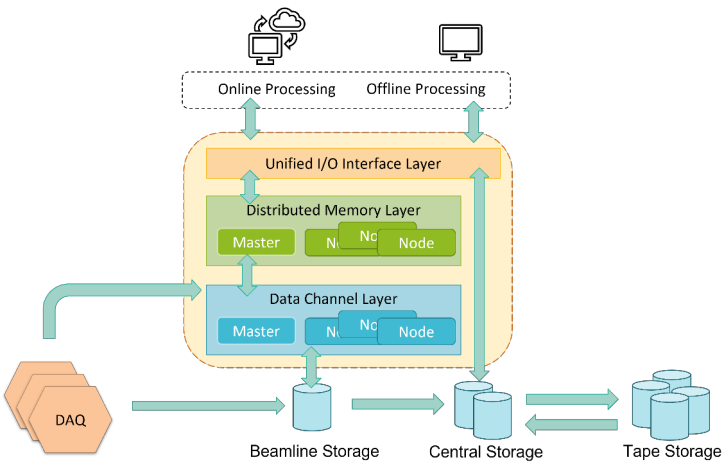


Figure 3: HEPS I/O atchitecture

before the computation commences. The memory management layer effectively addresses this issue and prevents performance loss caused by disk I/O.

The I/O interface layer is responsible for providing a unified I/O interface for computations. When receiving I/O requests from computational tasks, it determines whether the data is from the memory management layer or the storage layer, and calls the corresponding methods to complete the data reading tasks.

2.1 Batch data I/O

In beamline 7, CT reconstruction is the mainly computational task, known as HEPSCT. During the experimental process, scanning the sample generates a large volume of projection images. The role of HEPSCT is to reconstruct the sample using these projection images, which consist of numerous grayscale images. The smallest reconstruction unit is a layer of these images, with each layer comprising the same row of data from different images in the scanning sequence. And the processing of different units is independent of each other. Besides, due to memory capacity limitations, the task is divided into multiple batches, with each batch containing one or multiple units. The HEPSCT task encompasses several steps: reading, pre-processing, reconstruction, post-processing, and writing the results into storage. Each step requires varying time for execution, with the read and write operations constituting the predominant time-consuming steps, as illustrated in figure 4. It is evident that I/O bottlenecks significantly impact the computational speed.

Using the HEPSCT as an example, pipeline technology has been implemented to alleviate the impact of data I/O on computational efficiency through parallel asynchronous processing.

Firstly, the computational tasks are analyzed. Asynchronous I/O is used to overlap the data reading and writing, thereby reducing the overall waiting time, as illustrated in figure 5a. Secondly, as shown in figure 5b, for the TIFF format, multi-threading is utilized to read a batch of data, accelerating the data reading speed. As for HDF5, its kernel currently does not support multi-thread reading. However, its reading can be executed using multiple processes, and the data can be transferred to computational tasks through shared memory. Additionally, HDF5 offers various features such as chunking and compression, which also contribute to accelerating data I/O speed. Subsequently, as shown in figure 5c, since the data in different

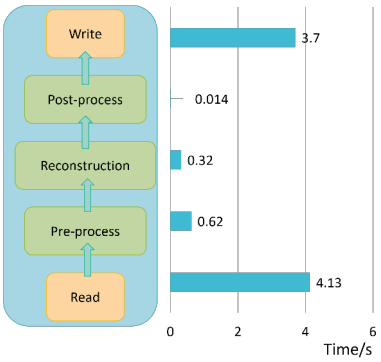
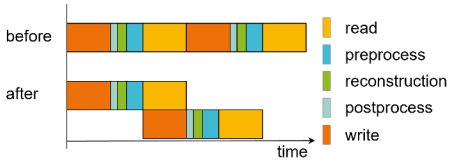
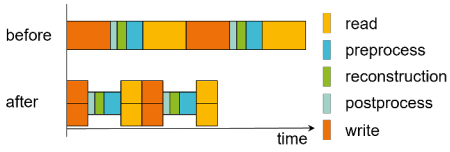


Figure 4: Time of every step in HEPSCT

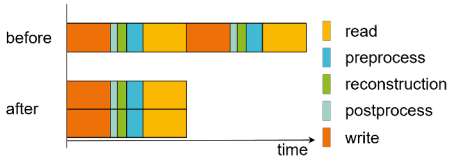
computing processes are independent of each other, further efficiency improvements can be achieved through parallel computing.



(a) Asynchronous I/O



(b) Multi-thread I/O



(c) Parallel computing

Figure 5: I/O optimization for batch processing

When receiving a data reading or writing request, the I/O interface can ascertain the storage location of the data, whether it is in the memory management layer or on disk, based on the path's prefix. Distinct namespaces are assigned for different storage locations. When reading or writing data on disk, users are only required to specify the file directory on the disk containing the raw data to be processed. The I/O interface automatically determines the data format based on the file extension in the directory and invokes the corresponding data format

reading function. This approach effectively shields users from the intricacies of underlying data formats and sources.

2.2 Stream data I/O

In contrast to the batch processing, stream processing eschews the practice of persisting data to disk. Instead, it directly stores the data in memory for real-time computation. Once the incoming data meets the computational requirements, the processing tasks are initiated.

In stream processing, the data is received from the DAQ system in a packaged format, initially structured as a dictionary type. This package includes metadata and data information, such as detector status, raw data, and other pertinent details required for stream processing. These details are then packed using JSON[3] and sent out via ZeroMQ[4].

In our architecture, as shown in figure 6, the data channel layer is designed to receive the packed data from the DAQ system. It determines the data processing flow based on the subsequent processing information contained in the data package. The data channel layer utilizes Flink[5] to submit a job for receiving data via ZeroMQ and awaits the data stream. Once receiving the data stream, if the current data stream can be computed directly, the corresponding processing program will be initiated. If not, the data will be stored in the memory management layer, awaiting completion before being processed through a distributed computing framework. The memory management layer, based on Alluxio[6], provides comprehensive memory management capabilities and presents a unified directory structure to the I/O interface layer.

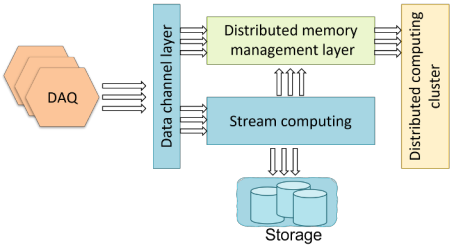


Figure 6: Stream data I/O architecture

3 Conclusion

In response to the challenge that managing the colossal volume of data within HEPS, we have designed an efficient I/O framework alongside optimization methods. This framework exhibits the capability to process both stream data and file data, furnishing a unified I/O interface for computations while masking the underlying data disparities. Leveraging distributed memory management, it adeptly oversees the management of large-scale raw data and enhances the I/O speed for various data formats. In the future, this approach will be seamlessly integrated into Daisy[7], a purpose-built software framework tailored for data processing within HEPS. The overarching objective is to propel the acceleration of scientific computing in the future.

References

[1] Q. Fazhi, H. Qiulan, H. Hao, T. Haolai, W. Lu, W. Yanming, Z. Haifeng, Z. Hongmei, Z. Shan, Frontiers of Data and Computing **2**, 40 (18) (2020)

[2] M. Folk, G. Heber, Q. Koziol, E. Pourmal, D. Robinson, *An overview of the HDF5 technology suite and its applications*, in *Proceedings of the EDBT/ICDT 2011 workshop on array databases* (2011), pp. 36–47

[3] json.org, *The json data interchange format*, https://www.json.org/json-en.html (2023-09-21)

[4] P. Hintjens, *The zeromq guide*, https://zguide.zeromq.org/ (2023-09-21)

[5] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, K. Tzoumas, The Bulletin of the Technical Committee on Data Engineering **38** (2015)

[6] H. Li, *Alluxio: A virtual distributed file system* (University of California, Berkeley, 2018)

[7] Hu, Yu, Li, Ling, Tian, Haolai, Liu, Zhibing, Huang, Qiulan, Zhang, Yi, Hu, Hao, Qi, Fazhi, EPJ Web Conf. **251**, 04020 (2021)