Design and implementation of computational storage system based on EOS for HEP data processing

**Minxing Zhang**1,2**，Yaodong Cheng**1**，Yaosong Cheng**1

1 Institute of High Energy Physics, CAS

2 University of Chinese Academy of Sciences

**Abstract.** The field of high energy physics is a typical data-intensive computing environment. There exists a kind of high statistical computation in the high energy physics computing paradigm, which requires access to a large amount of data for analysis. Under the background of large data volume, the traditional "computing-storage" separation system needs to carry out high frequency data movement, which tends to produce longer transmission delay and increase network load. Generally, storage nodes also have computing resources, such as cpus, necessary for deploying distributed file systems. However, the computing capabilities of these computing resources are often ignored. Therefore, offloading computing to computational storage on storage is a viable solution. This paper introduces a computational storage scheme implemented under the distributed file system framework commonly used in high energy physics, and tests its acceleration effect in typical data intensive applications. For a single test application, computational storage mode reduces computation time by 37% compared with traditional mode. Moreover, with the increase of parallel applications, the computational storage mode becomes more stable, and the computation time is reduced by 72% in the case of 40 parallel applications.

1. Introduction

The past 30 years has been a period of rapid development of large scientific devices related to high energy physics at home and abroad. At present, the global amount of high energy physics data has exceeded 1000 PB[1]. Therefore, at present, distributed computing environment based on storage cluster and computing cluster is generally adopted in the field of high-energy physics with high-speed network as the core [2]. The storage cluster and the computing cluster are connected by the core high-speed network, forming a typical "computing - storage" separation structure[3].

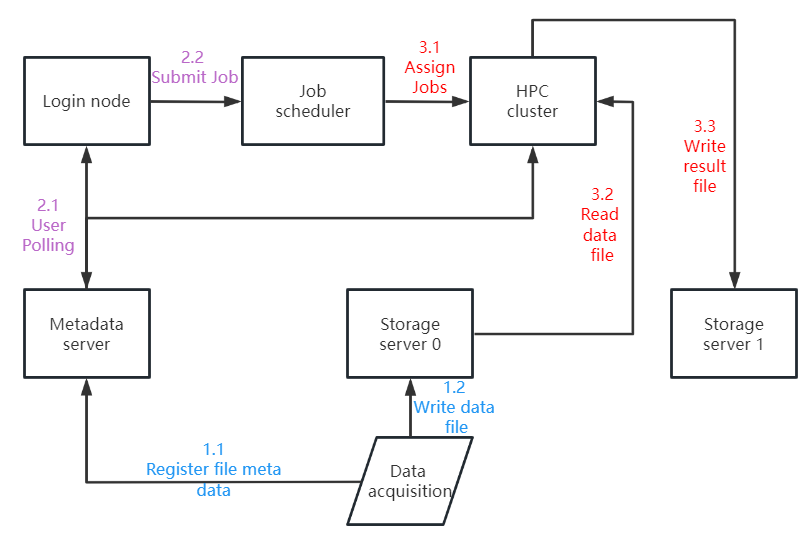
In high-energy physics, there exists a class of high statistical computation, which requires access to a large amount of data for analysis. For example, LHAASO experiment generates trillions of cases every year. Before analyzing these huge raw data, it requires decoding as a preprocessing, and in this process, very high I/O bandwidth is required. The I/O resource consumption is greater than the CPU consumption. Eliminating these bottlenecks on the basis of the separation structure requires higher hardware costs, such as higher bandwidth network links, more efficient switches, and enough compute nodes. The hadoop-like[4] clusters seem to solve this problem, but require a massive overhaul of the existing hardware and software ecosystem.

In recent years, a computational storage[5] solution has been proposed. The main idea of computational storage is to sink part of computing power into the data storage part, reduce data movement and improve computing efficiency. The definition of computational storage is constantly expanding, and it is widely believed that the architecture of storage coupled computing storage function (CSF)[6] is a typical feature of computational storage. Therefore, in a broad sense, it can be regarded as a computational memory that the storage node undertakes the computational task .

This paper provides a lightweight solution: According to the high-energy physics computing environment, a channel for invoking computing resources of storage nodes is built based on the EOS distributed file system[7] commonly used in this field. It can improve the computing resources at the disposal of users and reduce the network transmission of specific applications without increasing the hardware. Improve the overall computing efficiency.

1. Background and motivation

In this paper, the concept of computational storage is applied to the high energy physics data storage process, the XRootD plug-in structure is used to build the computational storage call path, the application algorithm is deployed on the storage node.We want to unload the application, which have small computation but large I/O consumption, on the storage server.



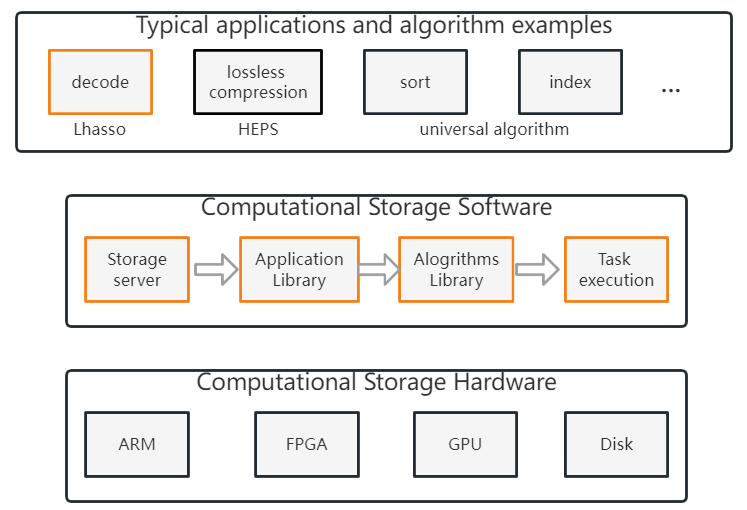
**Fig 1.** Traditional workflow of high energy physics

We choose to develop based on EOS distributed file system. The core of EOS system implementation is XRootD framework[8], which provides a remote access protocol with rich functions. XRootD framework is a common data access protocol in the field of high energy physics. It is compatible with ROOT, a common analysis software used in high energy physics.

The XRootD protocol provides the low-level transport and connection, and the EOS system implements the upper level file system. The EOS system structure is designed as three layers: client, metadata server (MGM) and file storage server (FST). The communication between each layer is realized by XRootD protocol. Files are not stored in fragments. Single file is stored on only one node.

1. Design and implementation

Based on the above motivation and related research, we propose the architecture of high energy physics computational storage system.



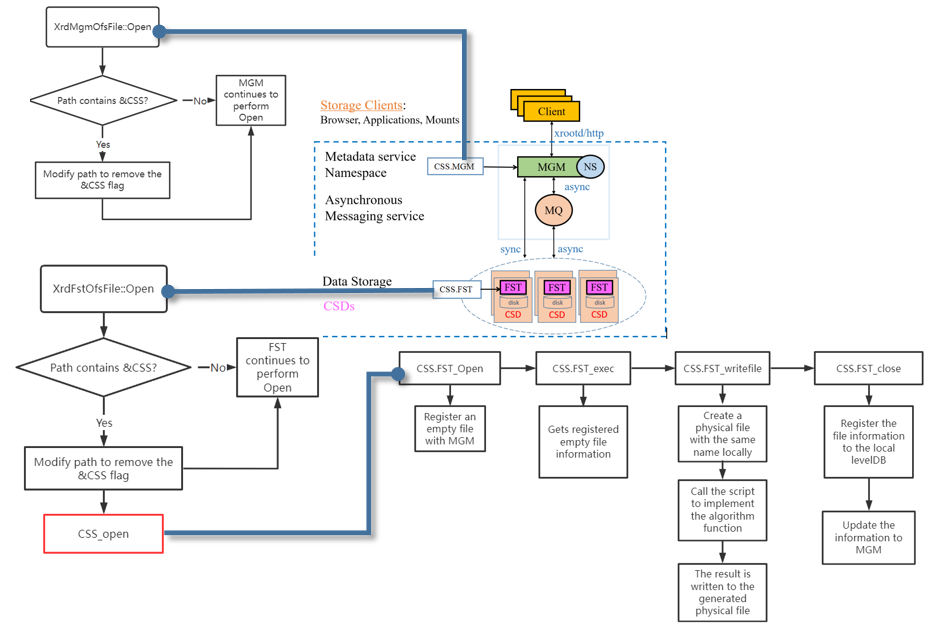
**Fig 2.** HEP Computational Storage

The main function of the system constructed in this paper is to offload a portion of the computations that are tightly coupled to the data from the computing server to the storage server. Without the need to use computational storage (CSS), it is still as normal as the EOS storage system read and write data, file management. When the CSS function is needed, the user only needs to add a special flag to the file path where the file is normally opened, and then the computational storage module can recognize and carry out the calculation in the local file storage.

In order to achieve the intended functionality, we investigated XRootD's plug-in structure[9] and the process of opening files in EOS. Custom modules can be loaded and recognized by XRootD in the form of dynamic link libraries.

Based on the EOS file opening process, we separately added plugins before MGM and FST. The convention uses "&CSS" as the flag for invoking a computational storage service. MGM provides redirection information including file locations and checks the path validity. A layer of filtering in front of MGM is necessary. When the "&CSS" flag is detected in the path, the path is temporarily restored to a common file access path, and then sent to MGM. The actual computational storage service call needs to be implemented on the FST where the file resides. When the client locates the responding FST through the redirection information, it will send the file access request to the FST, and add a layer of filtering before the FST. Then the computational storage service can be activated locally in the storage node through the message passing mechanism of EOS.

Our goal is to compute locally against the source file after the computational storage service is invoked, and generate a result to be stored as a file. The result file should be recognized by EOS as a normal data file, so it needs to be registered on MGM. When the CSS module is called, it registers a new empty file on the MGM after checking that the access call is valid. Each file registered into the MGM has a logical address. The logical address of the result file which be used for a logical file system on a distributed cluster, is in the same directory as the source file. MGM automatically assigns a physical file name to the result file. MGM returns information such as the physical file name assigned to the generated file as required. After obtaining the required information, a physical file corresponding to the new file is created on the storage device. The storage server performs data processing based on the obtained information and outputs the result to the new file. At this point, the empty file we created has been written to the information and is the result file. Then it need to register the result file with the local levelDB database and synchronize the information updated in the levelDB to the MGM. After the consistency check is passed, the CSS service ends.



**Fig 3.** CSS is deployed on the EOS file system

1. Test and evaluation

To test the actual performance of the CSS service, we tested it on a storage server where the storage disk was an SSD. The server SSD model is SAMSUNG MZ7KH960 960GB. The server uses Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz. The raw data file size used for the test is 953.7MB, and the calculation function used for the test is Decode, with the result file size being 337MB.

**Fig 4.** The completion time for the two computing paths

The line chart in Fig 4 nicely illustrates the advantages of CSS computing over traditional computing. The horizontal coordinate in the figure represents the number of parallel Decode tasks in a computation. The more parallel tasks, the greater the I/O consumption. The ordinate represents the time taken to complete tasks in a calculation. The longer the time, the less efficient the calculation mode is for the application.

It can be seen that the computational power of the storage node is sufficient to support the computational amount needed by the decoding program, and the difference in I/O performance between data transmission within the node and data transmission through the network is responsible for the higher efficiency of the CSS computing.

## According to the above test results, we can conclude that the CSS computing mode has a better acceleration effect in the large data environment for specific applications with high I/O consumption.

1. Summary and outlook

This paper introduces the framework of high energy physics computational storage system, and focuses on the implementation of a computational storage service based on the common distributed file system EOS and XRootD framework, which can accelerate some data intensive applications without increasing the hardware cost. Tests on a storage server show that a single decode application running in this way can reduce the run time by 36% compared to the conventional approach, and the improvement is more significant with the addition of parallel applications, with a 72.1% reduction in the run time when 40 Decode applications are running.

In the future, we hope to continue to improve the high energy physics computational storage system, and realize the computational storage path called by the physical analysis software and the computational storage service that is transparent to users. At the same time, add heterogeneous computing resources such as GPU or FPGA for storage nodes to extend the application scope of computational storage services.

## References

1. Chen G. Data and computing for high energy physics experiments (in Chinese). Sci Sin-Phys Mech Astron, 2021, 51: 092003, doi: 10.1360/SSPMA-2021-0108
2. Cheng Yaodong, Shi Jingyan, Chen Gang. An Overview of the High Energy Physics Computational Environment [J]. Information Technology and Application of Scientific Research, 2014,5 (03): 3-10.
3. Ma Yili, Fu Xianglin, Han Xiaoming, et al. Separation of storage and computation [J]. Computer research and development,2005.
4. Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system[C]//2010 IEEE 26th symposium on mass storage systems and technologies (MSST). Ieee, 2010: 1-10.
5. SNIA Technical Work Group on Computational Storage. https://www.snia.org/computational.
6. Computational Storage Architecture and Programming Model v0.8 rev 0. https://www.snia.org/sites/default/files/technical\_work/PublicReview/SNIA-Computational-Storage-Architecture-and-Programming-Model-0.8R0-2021.06.09.pdf
7. Peters A J, Sindrilaru E A, Adde G. EOS as the present and future solution for data storage at CERN[C]//Journal of Physics: Conference Series. IOP Publishing, 2015, 664(4): 042042.
8. Dorigo A, Elmer P, Furano F, et al. XROOTD-A Highly scalable architecture for data access[J]. WSEAS Transactions on Computers, 2005, 1(4.3): 348-353.
9. XRootD Configuration Reference 2021, Release 5.1.0 and above Andrew Hanushevsky.https://xrootd.slac.stanford.edu/doc/dev48/xrd\_config.pdf