Graphical Abstract

XHDF5: a high-performance HDF5 remote data access system for HEPS

Shichao Feng, Yaodong Cheng, Yaosong Cheng

Highlights

XHDF5: a high-performance HDF5 remote data access system for HEPS

Shichao Feng, Yaodong Cheng, Yaosong Cheng

- We developed the XHDF5 system to enhance data processing efficiency for remote users accessing HDF5 files in the High Energy Physics Data Center (HEPS).
- By integrating HDF5 VOL and XRootD services, XHDF5 demonstrates exceptional performance in high-latency environments, optimizing data access speeds while maintaining high stability and throughput.
- This research offers an innovative solution for large-scale data management, particularly for applications requiring high-performance remote data access, outperforming H5serv and NFS in remote environments.

XHDF5: a high-performance HDF5 remote data access system for HEPS

Sir Shichao Feng[a,b], Yaodong Cheng[b,c,*] and Yaosong Cheng[b]

[a]School of Computer and Artificial Intelligence, Zhengzhou University, 450001, Zhengzhou, Henan, China
[b]Institute of High Energy Physics, Chinese Academy of Sciences, , 100049, Beijing, China
[c]University of Chinese Academy of Sciences, 100049, Beijing, China

ARTICLE INFO

Keywords:
XRootD
HDF5
HEPS
NFS
Parallel

ABSTRACT

The High-Energy Photon Source (HEPS) is is the first fourth-generation synchrotron light source facility in China, which will generate huge amounts of scientific data. HDF5 is a widely used file format for scientific data storage and management, which has been established as the standard data format by domestic synchrotron facilities. As the storage scale of the HEPS data center expands and international collaboration increases, remote access systems have become essential for remote users. Therefore, it is urgently needed to develop an efficient and adaptable remote access solution based on the HDF5 data format to improve the efficiency and performance of scientific computing and data analysis. Based on in-depth research into HDF5 technology and the XRootD distributed file system, we designed and implemented the XHDF5 system. Compared with the traditional NFS access mode, XHDF5 has higher efficiency and stability in cross-region data sharing and collaboration, and provides a more reliable remote data access platform for researchers.

1. Introduction

The large scientific facilities, such as the High Energy Photon Source (HEPS) [1], serve as a platform for fundamental research services across various disciplines in China. These facilities generate a large amount of scientific data of great research value during their scientific activities. HEPS project will build 14 beam stations and one test beam station in Phase I, and 90-plus beam stations in Phase II, with an estimated daily data volume of 40TB [7]. Faced with the huge amount of data generated by these large scientific facilities every day, efficient management and utilization of these data are crucial for scientific research [12].

HDF5 (Hierarchical Data Format version 5) [17] is a file format used for storing and managing scientific data, which is increasingly adopted by fields such as biology, astronomy, and geosciences due to its efficiency and flexibility. It has also been established as the standard data format for domestic light source facilities. The application of HDF5 can effectively address the storage and access needs of large scientific data [3].

For the large amount of data generated by HEPS, there are currently three data access methods:

- Analysing data at the HEPS data center.
- Analysising on the user's computing platform.
- Analysising on a public cloud. The latter two methods both require remote access to data.

Utilizing HDF5 format and XRootD-related technologies, we have designed and implemented a high-performance remote data access system (XHDF5) to support the data processing needs of large-scale scientific facilities such as HEPS. This system incorporates the HDF5 library, parallel HDF5 [10], and network transmission protocols to ensure efficient processing of massive scientific data in remote computing environments. By studying and applying these methods, the efficiency of scientific data analysis has been significantly improved, thereby driving progress in multidisciplinary research.

*Corresponding author
chyd@ihep.ac.cn (Y. Cheng)

2. Background and Motivations

2.1. NFS Performance Analysis

Currently, the main way for HEPS data center to access HDF5 files is through Network File System(NFS) mounting. For the same data center, the synchronization mechanism, caching mechanism, and high consistency checking of the NFS [14] protocol can effectively ensure data consistency. For remote environments, the synchronization mechanism and request/response model of NFS will have a significant impact on transmission performance [15].

The synchronization mechanism of NFS is specifically designed to coordinate the client and server regarding data consistency and write operations. This ensures the accuracy and integrity of data write processes. In synchronous operations, each operation must wait for the server to confirm before proceeding, which will cause the operation to become slower as network delay increases. When the latency of each write operation is high, the overall performance will be seriously affected [20].

Moreover, the request-response model of NFS based on RPC, the client must wait for the server to return a response before continuing the next operation after sending each request. When memory-mapped file I/O operations are utilized, frequent page fault exceptions may occur when accessing the file. Each small block of data necessitates a request to be sent to the network and subsequently confirmed before it can be accessed. By tracking the NFS data packets, it is found that for each page fault access based on memory-mapped file I/O, NFS needs to initiate a request. The system must then wait for the server to process each request serially.

In terms of parallel access, NFS supports a certain degree of parallel access, but is subject to many limitations. NFS usually uses a single server to handle file requests, although throughput can be improved by using multi-threading, but the server's own performance and bandwidth limitations will still be the bottleneck. Moreover, when multiple clients operate on the same file, NFS needs to frequently synchronize and lock the cache, which may reduce performance in parallel access. In high-latency networks, especially when clients and servers need to interact frequently, parallelism cannot be fully utilized.

2.2. HDF5 Storage

HDF5 is a file format used for storing and organizing large amounts of data. It supports various types of data and allows data to be stored in a flexible and scalable way through a hierarchical structure. The underlying storage model of HDF5 is implemented using a set of metadata and actual data blocks, where each file contains multiple different data structures.

The storage layout in HDF5 can adopt different configurations, primarily including three modes as follows:

(1) Continuous storage. In this mode, data is stored linearly within a block, but the size cannot be dynamically adjusted.
(2) Block-based storage. This mode facilitates the dynamic adjustment of the dataset's size.
(3) Compact storage. In this mode, data is stored directly in the lower object header (metadata information) and is only suitable for small datasets.

HDF5 files of HEPS predominantly use a contiguous storage layout rather than a block-based storage layout, which affects read efficiency during contiguous reads.

In HDF5, the default method for reading files employs memory-mapped file I/O (mmap) [19]. This technique enables the direct mapping of file contents into memory, thereby facilitating efficient data access. The process begins with the retrieval of the dataset's metadata. In cases that necessitate random access, an indexing structure is utilized to determine the appropriate data offset. Once this offset is established, data can be accessed directly from the file using the specified file offset. This approach optimizes the reading process, enhancing performance and reducing latency.

2.3. HDF5 Virtual Object Layer (VOL)

The HDF5 VOL (Virtual Object Layer) connector can be categorized into two distinct types: terminal and through. Terminal VOL connectors intercept the API calls and subsequently store the data on hardware. For instance, the DAOS VOL connector functions as a terminal connector, utilizing the DAOS [11] object format for storing HDF5 objects. Conversely, the asynchronous I/O VOL connector operates as a through VOL; it employs the HDF5 "local" VOL connector in its role as a terminal VOL to facilitate data storage in HDF5 file format.

The HDF5 VOL connector can be divided into two types: terminal and through. Terminal VOL connectors intercept the API and then store the data on hardware, usually in a file format different from HDF5. For example, the DAOS VOL connector is terminal and uses the DAOS object format to store HDF5 objects. On the other hand, the asynchronous

I/O VOL connector is a through VOL that uses the HDF5 "local" VOL connector as a terminal VOL to store data in HDF5 file format [21].

We implements remote access to HDF5 files using both local VOL connections and remote VOL connections, with specific details available in sections 3.5 and 3.6.

3. Design and Implementation

In response to the limitations of NFS access methods and the characteristics of HDF5 data storage, we designed and implemented the XHDF5 system with the objective of enabling secure and rapid access to remote data. The XHDF5 system incorporates techniques such as asynchronous I/O, parallel access, data compression, token authentication [9], and permission control. The use of asynchronous I/O, concurrent access, and data compression ensures high-performance data transmission in remote environments, while token authentication and permission control measures safeguard the security of data within the data center.

3.1. Asynchronous I/O

The NFS analysis indicates that frequent synchronization operations occur during the remote transmission of NFS, leading to suboptimal transmission performance in remote environments. In contrast, XHDF5 employs asynchronous I/O to offer users data access interfaces that facilitate smooth transmission in such settings.

In the realm of high-energy physics, XRootD is particularly well-suited to address the demands of data-intensive applications [5]. With its high-performance and highly scalable architecture, it can manage vast amounts of data while maintaining stable concurrent request-response capabilities in multi-user environments. Consequently, the XHDF5 system utilizes the XRootD server to provide a set of asynchronous read and write interfaces for HDF5 files, thereby mitigating frequent I/O operations akin to those experienced with NFS during data transmission. To enhance system universality, HTTPS protocol is adopted within XRootD as the communication medium between users and data centers [6].

Compared to NFS, HTTPS sustains high transfer efficiency under conditions of elevated latency through streaming techniques and more effective window management. Furthermore, the HTTPS protocol supports automatic encryption and enjoys widespread compatibility along with standardized certificate management.

3.2. Parallel Operator

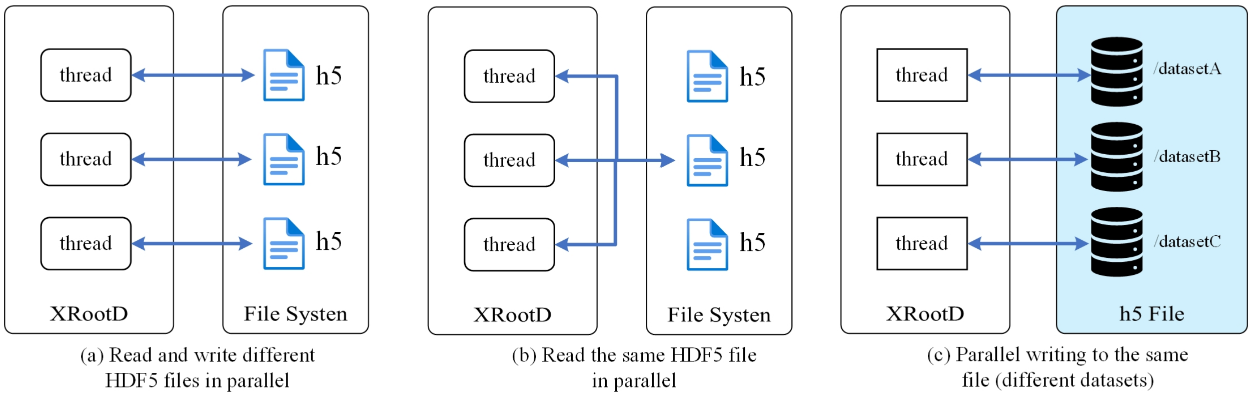

thread
thread
thread
h5
h5
h5
XRootD
File Systen
(a) Read and write different HDF5 files in parallel
thread
thread
thread
h5
h5
h5
XRootD
File Systen
(b) Read the same HDF5 file in parallel
thread
thread
thread
/datasetA
/datasetB
/datasetC
XRootD
h5 File
(c) Parallel writing to the same file (different datasets)

Fig. 1. Parallel operator

NFS is subject to certain limitations during parallel access, whereas XHDF5 utilizes XRootD as a server to facilitate parallel data transfer, thereby enhancing system throughput and improving performance in remote environments.

XRootD deploys multiple nodes, providing users with various data transfer endpoints. On the client side, users can configure the server nodes for parallel transmission; the VOL connector utilizes multithreading to simultaneously read data from multiple server nodes. Users have the flexibility to dynamically set the number of remote server nodes and their addresses through environment variables.

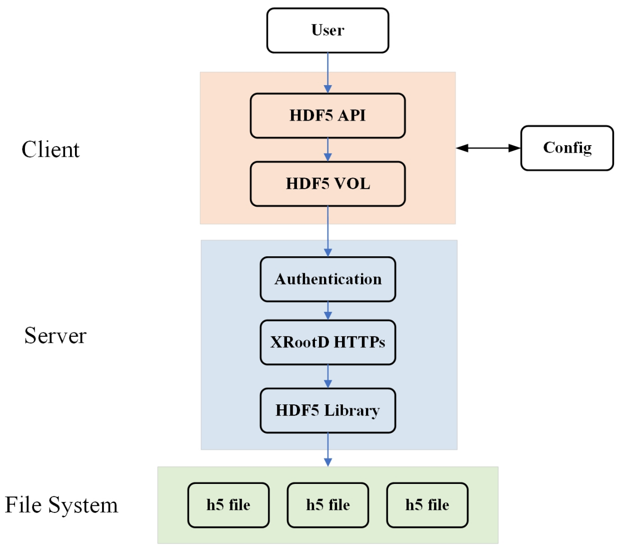


Fig. 2. System architecture

For users, XHDF5 enables concurrent read and write operations on files. These concurrent operations can be performed on the same file or across different files; however, it is important to note that during concurrent write operations, only distinct datasets within HDF5 may be manipulated simultaneously.

3.3. Data Compression

In the context of big data transmission, XHDF5 employs lossless compression algorithms to perform data compression on the server side [4]. Upon receiving the data, the client subsequently decompresses it. The application of these compression algorithms effectively reduces the time required for data transmission.

3.4. Data Security

Firstly, the system employs the HTTPS protocol, which encrypts data transmission using an SSL/TLS [18] protocol to ensure the security and privacy of data during transmission.

For the HEPS data center, user authentication is essential to determine user permissions. The system uses a token method for user identity verification and employs XRootD's access control lists to set read, write, and execute permissions for specific users or user groups on designated directories [8]. After authentication, the server checks the user's access permissions to files or directories based on their identity and group.

A tokens file exists on the user's local server. During the token authentication process, the system first retrieves the user's tokens from the local machine and transmits them to the server during communication. The server then verifies the tokens, and upon successful verification, returns the user's username.

For remote users, access to remote data is transparent. Users only have access to certain specific files and can only access the system's data through the server. Therefore, through the aforementioned operations of user authentication, user permission management, and data encryption, the security of remote data can be effectively ensured.

3.5. System Implementation

XHDF5 uses the HTTPS protocol for data access. The system architecture is illustrated in Fig. 2, where the top layer is the user layer. Users send data access requests through the provided API interfaces. User requests are processed at the HDF5 VOL layer and are then sent to the remote server via the HTTP client.

On the server side, the user requests are parsed, and user permissions are verified. This verification mainly follows the unified authentication method used by the High Energy Physics Institute [7], which involves validating the user's identity using the tokens code carried by the client. Based on the information in the user request, the server determines the operation to be executed and ultimately performs the corresponding actions on the HDF5 files in the underlying file system using the HDF5 local I/O library.

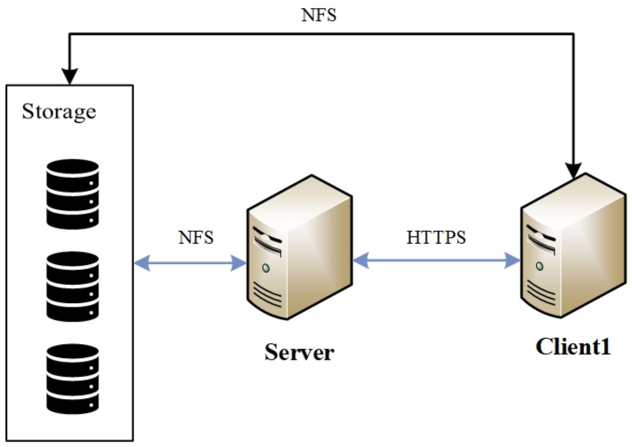


Fig. 3. Test environment

Table 1
The configuration of server.

	Server	**Client1**
Memory	503G	503G
Mount disk	2.5P	2.5P
CPU	112 cores	128 cores
CPU info	Intel 6338	Intel 6330
Operating System	Linux/CentOS7	Linux/CentOS7

3.6. Enabling XHDF5 in an application

For the XHDF5 system, the client applies the VOL framework, stacking both remote and local connectors, requiring users to only set environment variables without much code modification. HDF5_PLUGIN_PATH and HDF5_VOL_CONNECTOR are used to specify the locations of the connectors, while SERVER_NUM and XRD-MULTI_ENDPOINT are used to designate the number of server nodes and their URLs.

Similarly, in XRootD, the read and write operation interfaces are provided to users by loading shared libraries, without the need to compile the entire software. It is only necessary to specify the location of the shared libraries in the configuration file.

4. Experiment

4.1. Performance Evaluation

In the HEPS platform environment, the XHDF5 system is compared with NFS, and the performance is evaluated under different network delays. The specific structure of the test environment is shown in Fig. 3.

The test environment consists of two servers, with the storage system using the NFS network protocol to mount the underlying file system from the data center onto the servers. One server is located in the HEPS data center, and the other in Beijing Huairou. The default network latency between the server and client is within 1ms. Table 1 presents the configuration details of the two servers.

The test dataset for storage performance comprises light source images, each with a resolution of 2048×2048 uint16 (approximately 8MB). A total of 500 light source images were stored in the HDF5 file dataset, and the storage times for 100, 200, 300, 400, and 500 images were recorded during the testing process. The test environment and relevant device configurations are detailed in Table 1 and illustrated in Fig. 3.

Data transmission between the client and server is conducted using the HTTPS protocol; therefore, the iperf command was employed to assess network configuration and bandwidth between these two entities [2]. System test

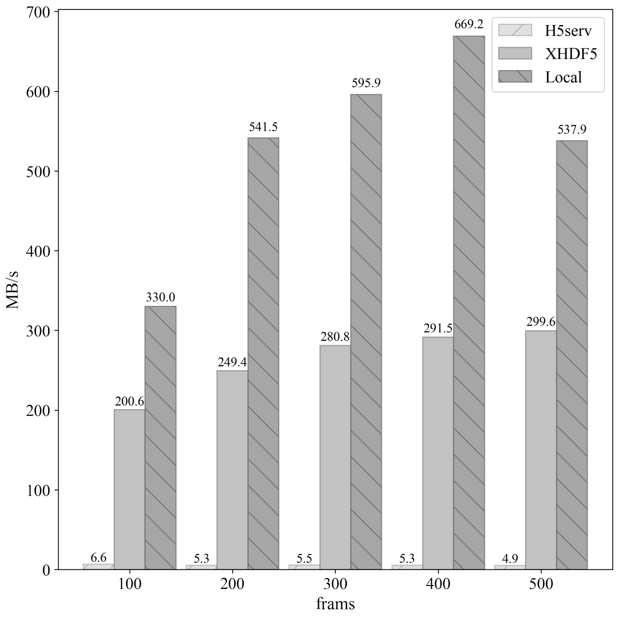


Fig. 4. Test results of XHDF5 vs H5serv

results indicate that under default network latency conditions, the bandwidth between the two servers is approximately 10 Gbit/s.

XHDF5 vs H5Server

H5serv [16] is a REST-based implementation of HDF5 data storage that provides an HTTP-based interface as a Web service, enabling users to access HDF5 files through a Web browser or other HTTP client.

During the test, we deployed both the H5serv Server and the Xrootd-based server on the same server machine and configured the corresponding client on Client1. In this setup, the network latency between Server and Client1 is kept under 1 millisecond. We recorded the time it took to read 100, 200, 300, 400, and 500 pieces of data from this dataset [13]. In the test program, the above operations were repeated 50 times to obtain an average read time. The final result is the storage speed calculated in MB/s. As shown in Fig. 4, the X-axis represents the read file size and the Y-axis represents the file transfer speed, which is measured in MB/s.

The light gray bar chart shows the read speed using H5serv as the server, the medium gray bar chart shows the speed using XDF5, and the dark gray chart shows the speed when reading local data on the client. As can be seen from the figure, the transmission speed is 5MB/s using H5serv, and about 300MB/s after switching XHDF5. Transfer speeds for local I/O operations can vary between 380MB/s and 620MB/s.

H5serv is slow to read large files because H5serv transmits data in json format, while HDF5 stores data in binary. The I/O time during the format conversion and database storage process is very large, so the data access speed is very slow. For XHDF5, access to remote file systems in a 1ms latency environment is 50 to 75 percent faster than local file access.

XHDF5 vs NFS

Before conducting performance tests on the system, we first evaluated the performance of the NFS-mounted file system. The file system was mounted on two servers (Server and Client1) using NFS. The performance of read, write, and mixed read/write operations was tested using the iozone command. The test program is written to perform the same store operation 50 times constantly. Test results are shown in Fig. 5. The x-axis represents different read/write modes, while the y-axis indicates data access speeds in MB/s.

Since Server and Client1 are located in the same data center, the read and write speeds on Server were higher than those on Client1. In the mounted mode, Client1 achieved a read speed of 890 MB/s and a write speed of 1134 MB/s. However, the IOzone test only evaluates the read and write performance of standard files. Given that HDF5 files use a

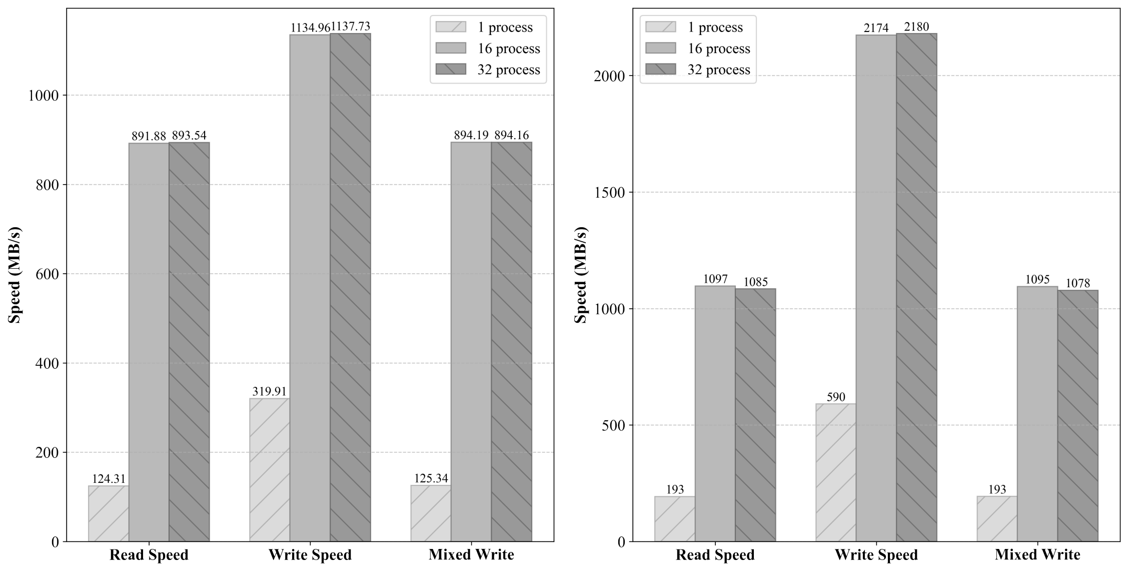
Table 2
The Data transmission speed of XHDF5.

Size (MB)	1ms	5ms	10ms	20ms	30ms	50ms
100	204.2 MB/s	240.7 MB/s	219.5 MB/s	211.7 MB/s	182.9 MB/s	144.8 MB/s
200	275.4 MB/s	262.5 MB/s	264.9 MB/s	267.2 MB/s	230.8 MB/s	195.4 MB/s
300	299.4 MB/s	335.3 MB/s	317.8 MB/s	286.4 MB/s	238.2 MB/s	207.7 MB/s
400	287.1 MB/s	321.5 MB/s	346.2 MB/s	300.8 MB/s	267.4 MB/s	226.0 MB/s
500	340.7 MB/s	323.2 MB/s	328.4 MB/s	319.1 MB/s	265.9 MB/s	220.2 MB/s

Table 3
The Data transmission speed of NFS.

Size (MB)	1ms	5ms	10ms	20ms	30ms	50ms
100	410.5 MB/s	223.4 MB/s	159.8 MB/s	99.4 MB/s	76 MB/s	49 MB/s
200	429.4 MB/s	228.7 MB/s	169 MB/s	112.4 MB/s	79.3 MB/s	50.9 MB/s
300	504.6 MB/s	232.9 MB/s	168.3 MB/s	111.5 MB/s	78.9 MB/s	51.9 MB/s
400	412.7 MB/s	246.6 MB/s	169.8 MB/s	109.4 MB/s	80.3 MB/s	52 MB/s
500	462.4 MB/s	252.9 MB/s	172.5 MB/s	112.4 MB/s	80.3 MB/s	52.4 MB/s

dedicated I/O library for reading and writing operations, the comparative tests will be conducted using this specialized I/O library for more accurate evaluation.



(a) Test results of NFS in Client1 **(b)** Test results of NFS in Server

Fig. 5. NFS Test Results

When conducting comparison tests with the NFS-mounted method, the underlying file system was mounted on Client1. The HDF5 I/O library was used in Client1 to record the time taken to transfer 100, 200, 300, 400, and 500 h5 files containing light source images. Additionally, for remote users, network latency is a significant factor affecting performance. Therefore, the tc command was used to introduce network latency between the two servers during testing. The test results are shown in Table 2-3.

Tables 2-3 show the transfer speeds for different image sizes under varying network latencies, while Fig. 6 presents a line graph of the transfer speeds for 500 images. The x-axis represents network latency, and the y-axis represents the

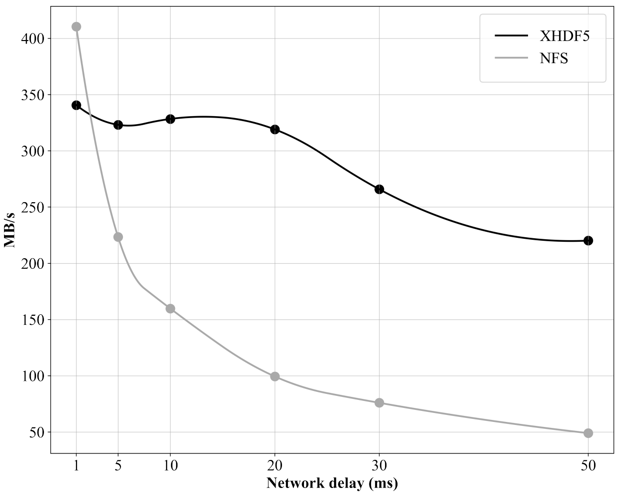


Fig. 6. Test results of NFS vs XHDF5

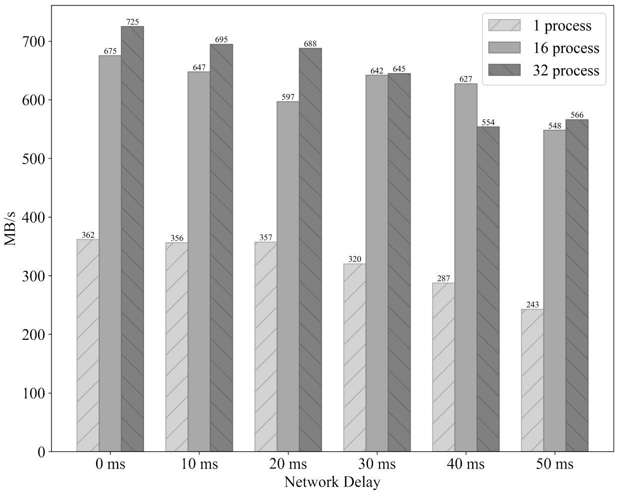


Fig. 7. Test results of multi process

corresponding data transfer speed in MB/s. From the graph, we can see that when the network latency between the two machines is 1ms, the NFS mount is significantly more efficient, but when the network latency exceeds 5ms, the transfer performance effect of the XHDF5 system significantly exceeds that of the NFS mount. For remote users, when accessing data, the change of network environment significantly affects the transmission efficiency. In particular, when the network delay reaches 20ms and above, the transmission speed of NFS drops sharply and cannot meet the demand for efficient data transmission.

The XHDF5 system demonstrates superior stability and transmission performance in the face of high latency, giving it a clear advantage in high-latency network environments. Specifically, under latency conditions of 30ms and 50ms, XHDF5 maintains a relatively stable transmission speed, indicating its suitability for remote data access and transfer.

With 1ms of latency, XHDF5 delivered 340 MB/s versus 462.4 MB/s for NFS. As the latency increases, the transfer speed of NFS gradually decreases, for example to 252.9 MB/s at 5ms latency, compared to 323MB/s for XHDF5. When the delay is further increased to 10ms and 20ms, the transfer speed of NFS is reduced to 172.5MB /s and 112.4MB /s, respectively, while that of XHDF5 is 328 MB/s and 319 MB/s, respectively. Within 20ms, the speed of XHDF5 is not

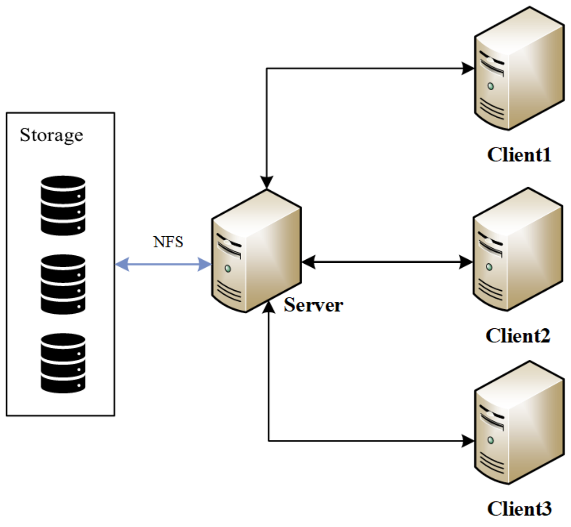


Fig. 8. Test environment of stress test

significantly degraded. At 50ms latency, NFS slows down to 52.4 MB/s, while XHDF5 remains at 220 MB/s, which is sufficient for normal access.

To evaluate the performance of the system under multithreading conditions at various latencies, the client concurrently accesses the server with concurrent thread counts set to 1, 16, and 32, each thread accessing 500 source image data files. The overall transmission performance of the system is then observed.

In Fig. 7, the light gray bars represent the speed when a single concurrent process is used, the gray bars represent the speed with 16 concurrent processes, and the dark gray bars indicate the speed with 32 processes.

From the graph, it is evident that when the number of concurrent processes reaches 16, the access speed approaches nearly 680 MB/s, with only a slight increase when reaching 32 processes. This indicates that the machine has already reached its maximum bandwidth at 16 processes. As latency increases, the overall transmission speed of the system is not significantly affected, remaining relatively stable, and even at 50ms latency, the system can still achieve a transmission speed of nearly 600 MB/s.

These data clearly demonstrate that while NFS performs exceptionally well under low latency conditions, its performance quickly declines as network latency increases. In contrast, XHDF5 exhibits much greater stability under high latency conditions, which is particularly important for users needing remote data transmission across regions or in unstable network environments.

For remote transmission of scientific data, selecting an appropriate transmission protocol and system is crucial. In low-latency local area network (LAN) environments, NFS is useful, but for wide-area networks (WAN) or cross-regional data transmission scenarios, XHDF5 undoubtedly has greater advantages.

4.2. Stress Test

To comprehensively assess the performance of system under high concurrency conditions, a multi-client concurrency testing scheme is designed. In this test, three clients were configured, with each client simultaneously initiating 32 requests, each requesting 4GB of data from the server.

In this testing scenario, the focus was on the following performance metrics:

- Network Throughput: This measures the amount of data transmitted per second to evaluate whether the network can support such large-scale data transfers. It also involves observing the overall network utilization, specifically the aggregate bandwidth at the server side.

- System Resource Consumption: This includes monitoring the CPU and memory usage of the server, particularly when multiple clients are simultaneously requesting large amounts of data, to determine whether the system experiences resource exhaustion or performance degradation.

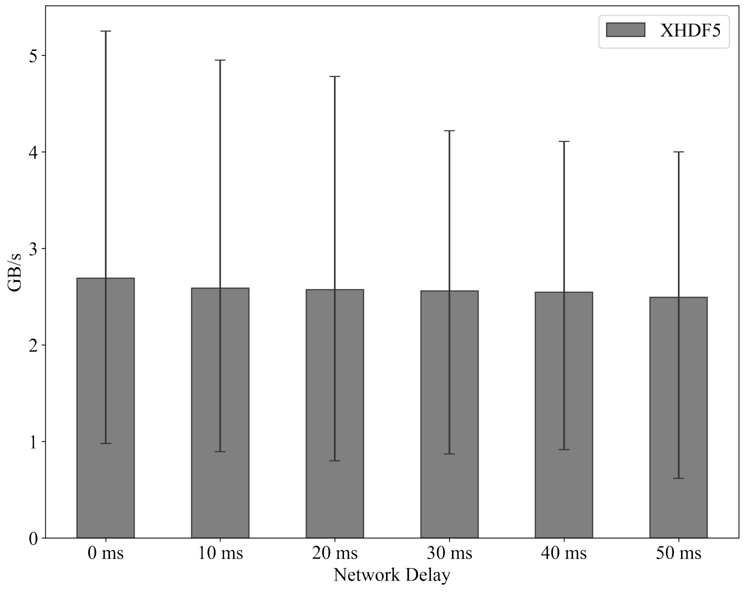


Fig. 9. Test result of stress test

- Stability: This involves evaluating the system's operational stability under prolonged high concurrency, recording whether any requests fail, timeout, or encounter other errors, as well as the frequency and causes of these issues.

Based on these metrics, the following experimental design was established:

(1) The HDF5 clients were deployed on three machines, with the XRootD-based server deployed on the server machine, configured similarly to the setup described in Table 1. The deployment of the machines is illustrated in Fig. 8.
(2) The number of threads for each client was set to 32, with each thread accessing 4GB of data from the server.
(3) Different network latencies were set (1ms, 10ms, 20ms, 30ms, 40ms, and 50ms). Under the same thread count and data size conditions, the overall aggregate performance and resource usage were recorded.

The experimental results are illustrated in Fig. 9. As we can observe from Fig. 9, The average network bandwidth remained at 2.7 GB/s, reaching the upper limit of the network capacity of Server. Even if network latency increases, the maximum network bandwidth of the system will only slightly decrease without any errors. This indicates that the network architecture of server and hardware design can maintain stable performance under high load conditions without exhibiting significant bottlenecks.

Additionally, the CPU and memory utilization during the operation of program is monitored, with results shown in Fig. 10. The blue line indicates system memory utilization, the yellow line represents cache utilization, and the green line indicates CPU utilization.

From the fig. 10, it shows that for the server side, the CPU utilization significantly increases during the operation of the storage program and subsequently decreases after completion, but throughout the entire process, the CPU utilization never exceeds 65%. The trend of memory utilization is similar to that of the CPU, peaking at around 65%, indicating that memory load is also not a bottleneck for the system.

For the three clients, the maximum CPU utilization is only 10%, while memory utilization does not exceed 7.5%. Therefore, both CPU and memory loads are not critical factors limiting the throughput on both the client and server sides.

Based on these observations, it can be inferred that the performance of HDF5 file I/O may have a significant impact on transmission performance. This is closely related to the efficiency of HDF5's I/O read and write libraries. Therefore, optimizing HDF5 I/O operations would help enhance overall storage and transmission performance.

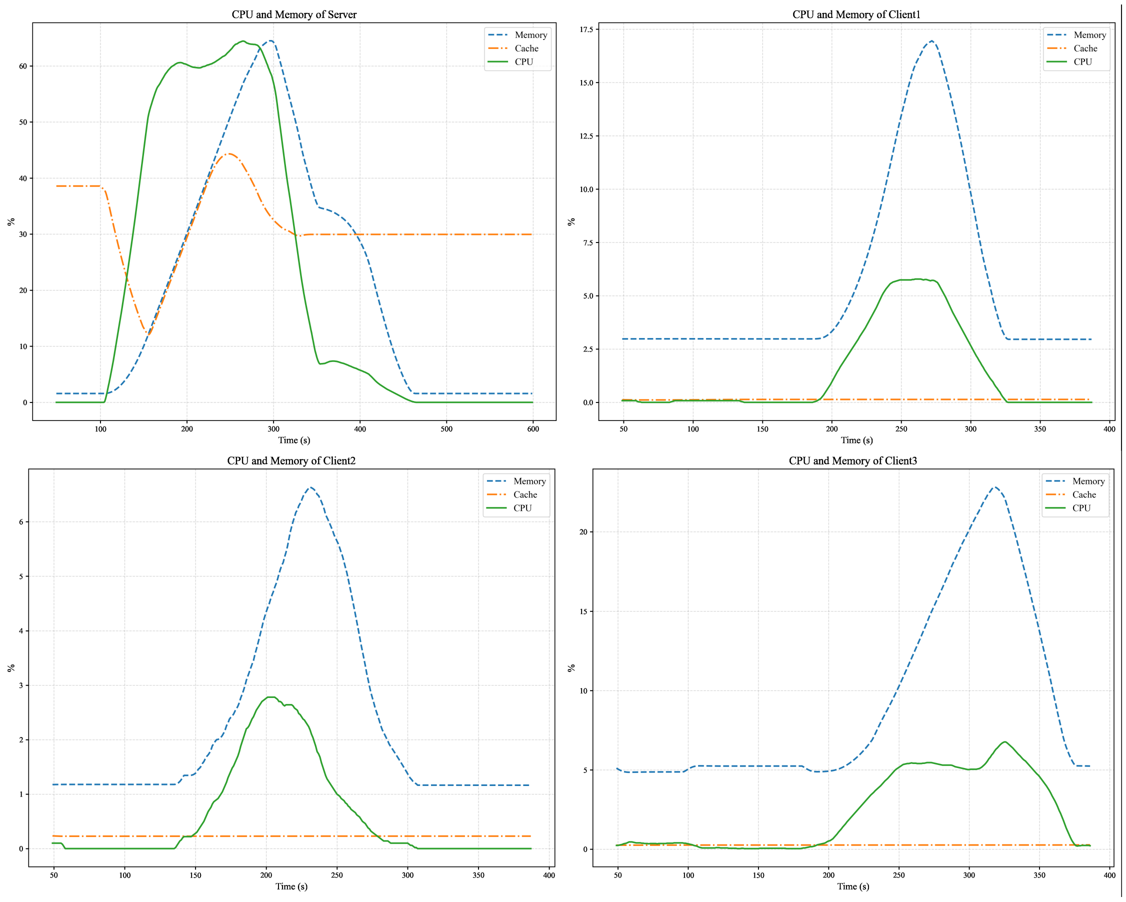


Fig. 10. CPU and Memory utilization

4.3. Summary

In the comparative testing of the XHDF5 system, XHDF5 is compared with H5serv and NFS. The performance of XHDF5 significantly exceeds that of H5serv. Under low latency conditions (less than 1ms), the mount method performs well for data access. However, when latency exceeds 5ms, the performance of XHDF5 surpasses that of the mount method. At latencies greater than 50ms, the performance of the mount method is considerably lower than that of XHDF5. Therefore, in practical remote environments, the performance of XHDF5 is far superior to the other two modes.

Subsequently, a stress test is conducted to evaluate the performance when hundreds of processes simultaneously accessed a single server for large-scale data. The experimental result indicates that, in high-performance mode, the server can still maintain maximum bandwidth output, and the system operates stably. This demonstrates that the system is also suitable for scenarios involving large-scale data access.

5. Conclusions

High-energy synchrotron radiation sources generate up to 300PB of data annually. An efficient, scalable, and user-friendly remote access system can significantly enhance analytical efficiency. By implementing XRootD and HDF5 VOL extension technology, a remote access system(XHDF5) has been developed to meet the demands of numerous users for remote data access.

During the experimental phase, tests are conducted comparing different access methods. In terms of remote access requirements, XHDF5 demonstrates excellent performance. Additionally, during stress testing, with over a hundred users making real-time data access requests for hundreds of GBs of data, the system maintained good performance and stability. The system also supports dynamic scaling of the server, allowing for more flexible responses to users' access needs.

XHDF5 meets the remote access needs of users, significantly improving the efficiency of scientific data analysis while ensuring data security.

References

[1] , . HEPS URL: http://english.ihep.cas.cn/heps/ah/bi/. accessed on 20-Sep-2023.

[2] Barayuga, V.J.D., Yu, W.E.S., 2015. Packet level tcp performance of nat44, nat64 and ipv6 using iperf in the context of ipv6 migration, in: 2015 5th International Conference on IT Convergence and Security (ICITCS), IEEE. pp. 1–3.

[3] Byna, S., Breitenfeld, M.S., Dong, B., Koziol, Q., Pourmal, E., Robinson, D., Soumagne, J., Tang, H., Vishwanath, V., Warren, R., 2020. Exahdf5: Delivering efficient parallel i/o on exascale computing systems. Journal of Computer Science and Technology 35, 145–160.

[4] Delaunay, X., Courtois, A., Gouillon, F., 2019. Evaluation of lossless and lossy algorithms for the compression of scientific datasets in netcdf-4 or hdf5 files. Geoscientific Model Development 12, 4099–4113.

[5] Dorigo, A., Elmer, P., Furano, F., Hanushevsky, A., 2005. Xrootd-a highly scalable architecture for data access. WSEAS Transactions on Computers 1, 348–353.

[6] Hanushevsky, A., 2023. The xrootd protocol version 5.2. 0. National Accelerator Laboratory, Dec 4.

[7] Hu, Qingbao, Xu, Jiping, Cheng, Yaosong, Luo, Qi, Fu, Shiyuan, 2024. Design and implementation of experimental data access security policy for heps container computing platform. EPJ Web of Conf. 295, 04017.

[8] Jones, R., Doidge, M., Hand, G., Love, P., Simpson, S., 2024. A blueprint for a contemporary storage element, building a new wlcg storage system with widely available hardware and software components: Ceph, xrootd, and prometheus, in: EPJ Web of Conferences, EDP Sciences. p. 01002.

[9] Khan, H.Z.U., Zahid, H., 2010. Comparative study of authentication techniques. International Journal of Video & Image Processing and Network Security IJVIPNS 10, 09–13.

[10] Lee, S., yuan Hou, K., Wang, K., Sehrish, S., Paterno, M., Kowalkowski, J., Koziol, Q., Ross, R.B., Agrawal, A., Choudhary, A., keng Liao, W., 2022. A case study on parallel hdf5 dataset concatenation for high energy physics data analysis. Parallel Computing 110, 102877. URL: https://www.sciencedirect.com/science/article/pii/S0167819121001174, doi:https://doi.org/10.1016/j.parco.2021.102877.

[11] Lofstead, J., Jimenez, I., Maltzahn, C., Koziol, Q., Bent, J., Barton, E., 2016. Daos and friends: A proposal for an exascale storage system, in: SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 585–596. doi:10.1109/SC.2016.49.

[12] Lv, H., Yan, Y., Wang, H., 2021a. The data storage system for shine. NUCLEAR INSTRUMENTS & METHODS IN PHYSICS RESEARCH SECTION A-ACCELERATORS SPECTROMETERS DETECTORS AND ASSOCIATED EQUIPMENT 1002. doi:10.1016/j.nima.2021.165285.

[13] Lv, H., Yan, Y., Wang, H., 2021b. The data storage system for shine. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 1002, 165285.

[14] OSADZINSKI, A., 1988. The network file system (nfs). COMPUTER STANDARDS & INTERFACES 8, 45–48.

[15] Peyrouze, N., Muller, G., 1996. Ft-nfs: an efficient fault-tolerant nfs server designed for off-the-shelf workstations, in: Proceedings of Annual Symposium on Fault Tolerant Computing, pp. 64–73. doi:10.1109/FTCS.1996.534595.

[16] Preuss, N., Staudter, G., Weber, M., Anderl, R., Pelz, P.F., 2018. Methods and technologies for research-and metadata management in collaborative experimental research. Applied Mechanics and Materials 885, 170–183.

[17] The HDF Group, 1997. Hierarchical data format, version 5 URL: http://www.hdfgroup.org/HDF5.

[18] Thomas, S., 2000. SSL andTLS essentials. New York, Wiley.

[19] Van Essen, B., Hsieh, H., Ames, S., Pearce, R., Gokhale, M., 2015. Di-mmap—a scalable memory-map runtime for out-of-core data-intensive applications. Cluster Computing 18, 15–28.

[20] Xiao-hui, C., Qi-liang, L., Jun-quan, H., 2013. Research on objective weight determine method of zigbee network performance index. Application Research of Computers/Jisuanji Yingyong Yanjiu 30.

[21] Zheng, H., Vishwanath, V., Koziol, Q., Tang, H., Ravi, J., Mainzer, J., Byna, S., 2022. Hdf5 cache vol: Efficient and scalable parallel i/o through caching data on node-local storage, in: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), IEEE. pp. 61–70.