Front. Comput. Sci., 2014, 8(3): 367–377
DOI 10.1007/s11704-014-3499-6

RESEARCH ARTICLE

Hybrid hierarchy storage system in MilkyWay-2 supercomputer

Weixia XU[1,2], Yutong LU (✉)[1,2], Qiong LI[2], Enqiang ZHOU[1,2], Zhenlong SONG[2], Yong DONG[1,2], Wei ZHANG[1,2], Dengping WEI[2], Xiaoming ZHANG[2], Haitao CHEN[1,2], Jianying XING[2], Yuan YUAN[2]

1 State Key Laboratory of High Performance Computing, Changsha 410073, China
2 College of Computer, National University of Defense Technology, Changsha 410073, China

Abstract With the rapid improvement of computation capability in high performance supercomputer system, the imbalance of performance between computation subsystem and storage subsystem has become more and more serious, especially when various big data are produced ranging from tens of gigabytes up to terabytes. To reduce this gap, large-scale storage systems need to be designed and implemented with high performance and scalability. MilkyWay-2 (TH-2) supercomputer system with peak performance 54.9 Pflops, definitely has this kind of requirement for storage system. This paper mainly introduces the storage system in MilkyWay-2 supercomputer, including the hardware architecture and the parallel file system. The storage system in MilkyWay-2 supercomputer exploits a novel hybrid hierarchy storage architecture to enable high scalability of I/O clients, I/O bandwidth and storage capacity. To fit this architecture, a user level virtualized file system, named H^{2}FS, is designed and implemented which can cooperate local storage and shared storage together into a dynamic single namespace to optimize I/O performance in IO-intensive applications. The evaluation results show that the storage system in MilkyWay-2 supercomputer can satisfy the critical requirements in large scale supercomputer, such as performance and scalability.

Keywords supercomputer, storage system, file system, MilkyWay-2, hybrid, hierarchy

Received December 17, 2013; accepted March 6, 2014

E-mail: ytlu@nudt.edu.cn

1 Introduction

With the increasing demands of scientific research and industrial applications, various terabytes scale big data have been produced [1,2]. This rapid data growth and the requirements of various applications result in a serious imbalance between computation performance with rapid improvement and storage performance in high performance supercomputers [3]. To reduce the gap between computation capability and storage capability, storage system needs to grow in both performance and capacity that can be scalable into tens of petabytes range, corresponding to the increasing number of compute nodes and cores [4–6].

In distributed shared storage architecture, there are many more compute nodes than I/O servers, and an I/O server may receive tens of thousands of near-simultaneous I/O requests in petascale supercomputer. This would bring a tremendous burden to I/O server, and limits the scalability of I/O clients and applications. In addition, I/O requests are often "bursty" in nature [7], which means large number of compute nodes may compete for the I/O servers at the same time. These requests can quickly exhaust the resources of I/O servers causing I/O bottlenecks that affect the performance and reliability of every competing application and the whole system [8].

The I/O bottleneck increases significantly with the scale of supercomputers enlarging, but the conventional I/O structure and file system have their intrinsic limitations for the performance and scalability. For example, the serialization I/O API limits the scalability, the synchronization I/O sematic lim-

its the efficiency, the disk, always being the slowest part of the I/O path, limits the performance, and the solid state disk (SSD) is more costly and limits durability. We need breakthrough architecture and file system to address these issues.

The exact goal of storage system in MilkyWay-2 supercomputer system ranking first on the top 500 list in June, 2013, is to achieve the scalability, performance, reliability and ease-to-use. Therefore, we design a novel hybrid hierarchy storage system, to fit for various storage patterns, to enable high scalability of I/O clients, burst I/O bandwidth and storage capacity. In this architecture, a two-layer storage model is constructed from I/O nodes (ION) and storage servers, and hybrid storage is implemented with high-speed electronic storage devices and traditional magnetic storage devices. In the two-layer storage model, the local storage is composed of all the local storage devices in ION and the shared storage is composed of all the disk arrays in object storage servers. To fit this architecture, a user level virtualized file system [9], named H^{2}FS, is designed and implemented which can cooperate local storage and shared storage together into a dynamic single namespace to optimize I/O performance. The I/O stack of H^{2}FS is constructed from several key components including user-level client, I/O path management, RPC communication, request scheduling, and so on. H^{2}FS provides locality-aware data layout and flexible management to serve diverse requirements of both data-intensive and computation-intensive applications.

The rest of this paper is organized as follows. First, we introduce the hybrid hierarchy storage system of MilkyWay-2 supercomputer. And then the file system on MilkyWay-2, named H^{2}FS, is introduced in Section 3. In Section 4, we describe the design and implementation of I/O stack in H^{2}FS. The evaluation of the storage system is presented in Section 5. Section 6 describes related work of storage system in HPC field. Finally, we conclude the work in Section 7.

2 Hybrid hierarchy I/O system

In traditional shared storage system, a large number of I/O Clients often request the shared I/O resources of one storage server and bring serious burden to the shared storage devices, which make tremendous challenges in performance, scalability and reliability of I/O system. Therefore, we design a hybrid two-layer storage model, called hybrid hierarchy storage architecture (shown in Fig. 1), to support the high scalability of I/O clients and improve the burst I/O bandwidth.

2.1 Architecture

The architecture of storage system in MilkyWay-2 (shown in Fig. 1) adopts two-layer storage model, in which the upper layer is composed of ION and the bottom layer is composed of storage servers. MilkyWay-2 can be configured with as few as one ION per 64 compute nodes, or as many as one ION per eight compute nodes.

The I/O requests can firstly be handled in the ION and then transmitted to storage servers. In the upper layer, the local storage, built from all the local storage devices of ION, provides sufficient burst performance for application as it is closer to compute node than external storage. The bottom layer includes metadata servers (MDS) and object storage servers (OSS). The shared storage, built from all the arrays of storage servers, provides storage capability with large capacity, high bandwidth and low latency. By employing the

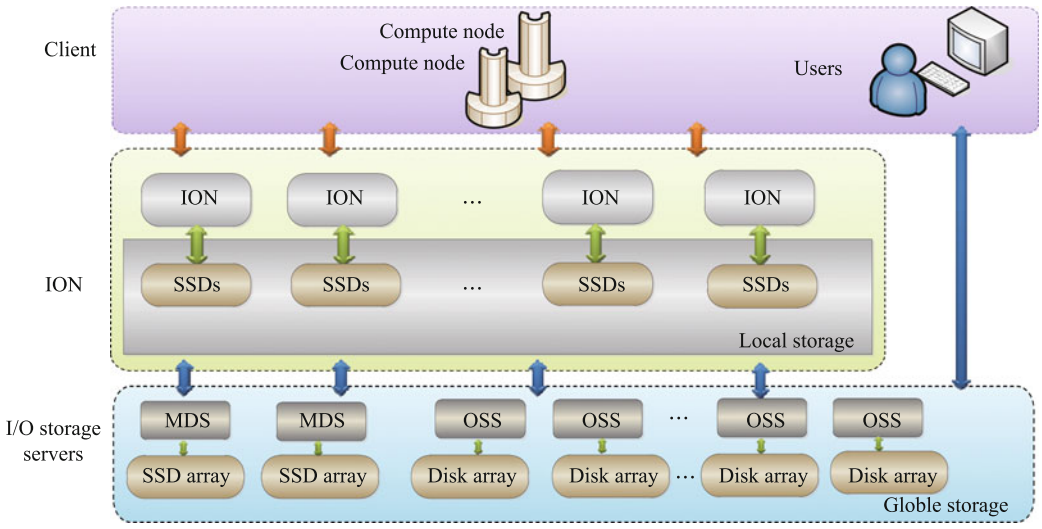


Fig. 1 Architecture of MilkyWay-2 storage system

ION in this storage architecture, compute nodes and storage system are loosely coupled. Therefore, the client number of parallel file system can be enlarged and the scalability of the storage system is improved.

In MilkWay-2, the local storage devices of ION are PCIe SSDs and the storage device installed in MDS is an SSD array. The storage system, therefore, implements hybrid storage with high-speed electronic storage devices and traditional magnetic storage devices, and has the capability to fit for various application patterns. ION equipped with SSDs provide much faster access rates than disk based backend storage and sufficient storage capacity. With the support of H^{2}FS, they can efficiently drain burst data and make applications return to computation without waiting for data to be transferred to final disk storage, thus the slow disk storage is hidden from applications. The data management and movement between these two layers are taken charge by H^{2}FS and transparent to applications. Besides, the capability of ION in handling the I/O requests is enhanced by exploiting the high-performance SSDs. Especially, the high I/O throughput and bandwidth of high-speed SSDs makes ION handle the burst I/O requests much more effectively.

ION share the work of handling I/O requests and provide some request preprocessing, e.g., read-ahead and write back, and thus reduce the burden of storage servers and improve the I/O throughput. Meanwhile, the burden on the communication network between the compute nodes and the storage servers is transformed into two parts, i.e., that between compute nodes and ION and that between ION and storage servers. Thus, the communication burden imposed on storage servers by I/O clients is alleviated. On the other hand, ION can reduce I/O congestion and I/O conflict, since ION are closer to I/O clients and can handle I/O requests effectively by utilizing their temporal and spatial localities.

From the point of fault tolerance, the compute process and the storage process are loosely coupled by adding the layer of ION. The fault propagation among compute nodes and storage nodes is alleviated and the reliability of the whole system is improved. Moreover, to enhance the ease-of-use of the storage system, we design a file system based on this architecture that can merge these two layers into a unified namespace and accelerate the I/O process of applications transparently. The file system also provides several extended interfaces for programmers to further optimize the I/O process of applications and get better performance.

2.2 Design of ION

The key issues of ION design are focused on high performance, high throughput, high capacity and low power consumption. The logic structure of ION is shown in Fig. 2. One ION is configured with 2 Intel Xeon Ivy Bridge CPUs, 2 PCIe SSDs and 64 GB memory. The two processors are connected with each other by a QPI path directly and connected with I/O controller PCH and proprietary network interface controller NIC. NIC connects with proprietary interconnection network and implements high-speed data transportation between the ION and the compute nodes. PCI SSDs are connected with processors through PCI interfaces. Meanwhile, the ION implements high-speed Infiniband host channel adapter (HCA) based on PCI Express 3.0 to connect with Infiniband storage network.

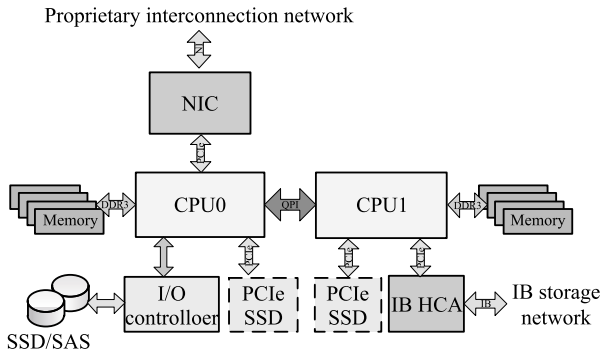


Fig. 2 Logic structure of ION

In large scale supercomputer, the parallel RDMA data access requests for an ION are usually from multiple compute nodes, indicating that many-to-one characteristics of communications between compute codes and ION may make the RDMA read/write engine in traditional ION helpless in handling these parallel requests effectively. In MilkyWay-2, eight RDMA read/write engines are designed and implemented in communication controller module of NIC, which can handle multiple RDMA requests in parallel and improve the capability of ION in handling I/O requests.

PCIe SSD, designed by National University of Defense Technology, China (NUDT), is composed of 64 NAND flashes of 16 GB with read bandwidth up to 2.5 GBps, write bandwidth up to 3.1 GBps, IOPS up to 450 000 and maximum power consumption 28 W.

3 H^{2}FS architecture

Performance, scalability, reliability and ease of use are critical requirements for scientific applications in large scale supercomputer. For the hybrid hierarchy storage architecture of MilkyWay-2, we design and implement a user level virtual-

ized file system, named H^{2}FS, which can satisfy the above requirements. H^{2}FS constructs an integrated storage space that merges local storage of ION and shared storage of storage servers to optimize I/O performance of the whole system. Local storage in H^{2}FS fulfills many requirements of large scale application, such as burst I/O bandwidth, data access latency. The external shared storage is used to fulfill requirements of large storage capacity and high reliability.

Figure 3 illustrates the architecture of H^{2}FS file system. H^{2}FS takes an ION and its local storage as a data processing unit (DPU), which is the first layer of H^{2}FS and is closer to compute node than external shared storage. A DPU can take full advantage of data access locality and support high efficient data access. Moreover, it can improve the scalability of the whole storage system by alleviating the data access pressure on storage system. Benefiting from PCIe based SSD in ION, DPU has higher read and write performance and provides sufficient booster performance for applications. In H^{2}FS, one DPU provides service for N computing nodes, which means the number of compute nodes is N times of the number of DPUs, N is determined by system deployment.

Shared storage based on Lustre file system [10] provides sufficient storage space and high sustaining performance for applications in H^{2}FS. To take advantage of both local storage and shared storage, we integrate several DPUs and shared storage to construct a virtualized storage space which we call a hybrid virtual namespace (HVN). The number of DPUs in an HVN is determined by system deployment and storage requirement of the job.

In each HVN, the directory tree structure is stored in shared storage, and the metadata of data file is maintained by DPUs and shared storage jointly. The file data in such unified namespace can be located in DPUs, shared storage or duplicated in both, which is determined by program at runtime. The access of data in unified namespace is consistent and transparent for application.

All HVNs are assigned to running application instances exclusively, isolated from each other and maintained by the HVN management server. Each namespace can own its customized data layout, metadata management policy and optimization approach.

The dynamic management of HVN, including creation and deletion, offers more flexibility for applications, which makes storage system easy to use. So it is convenient for programmers to optimize data access performance within HVN.

H^{2}FS has its own specific data layout interface and policy interface. Layout interface is designed for guiding the data layout in unified namespace, by which data can be stored in local storage of DPUs or shared storage. Policy interface is mainly used for choosing scheduling policy, hierarchy management policy, data storage and migration policy.

For ease of use, the typical interfaces of H^{2}FS include:

1) **Conventional POSIX interface** There is no need to modify application code for data access.
2) **Extended POSIX interface with data management policy** Besides conventional POSIX interface, data management policy is introduced to improve data access performance efficiently, which can take full advantage of local storage in DPUs.

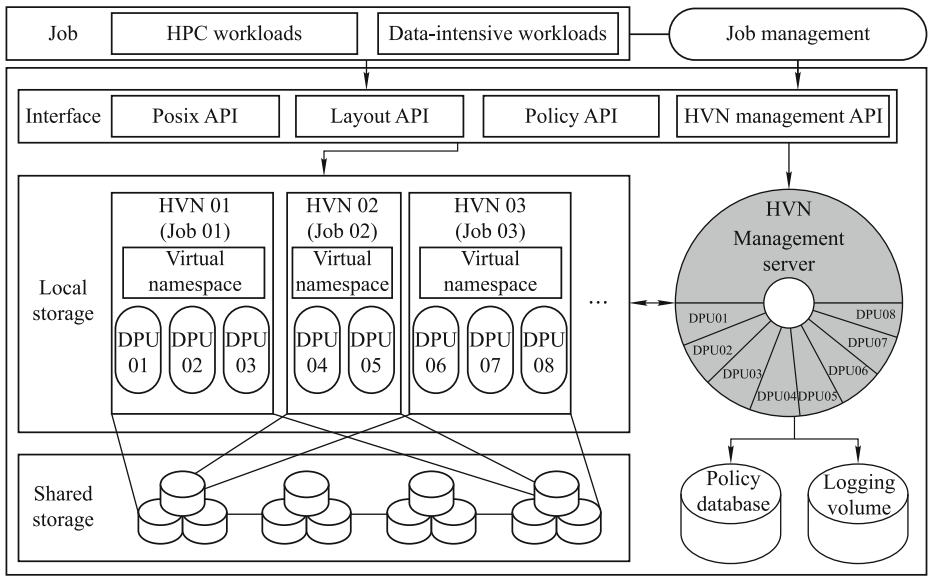


Fig. 3 Architecture of H^{2}FS

3) **HDF5/MPI-IO interface** H^2FS supports HDF5/MPI-IO interface, which also has been optimized for hierarchy architecture. Applications use HDF5/MPI-IO interface with the form of dynamic library to further improve I/O performance.

All these interfaces are implemented as dynamic library, by which application can be linked when compiling.

With the above interfaces, H^2FS supports three predefined I/O patterns: global I/O, local I/O and hybrid I/O. With global I/O mode, H^2FS directly forwards I/O requests of applications to external storage servers without depositing data into local storage. With local I/O mode, H^2FS absorbs burst data into local storage and asynchronously moves data to external storage. For the above two scenarios, local storage is completely transparent for applications. With hybrid I/O mode, H^2FS unifies all local storage of SSDs and external storage of disks into a single name space, thus application can manage its data stored on local storage, take it as burst buffer or data analysis store.

4 I/O stack design and implementation

The key to take full advantage of MilkyWay-2 I/O architecture is a software I/O stack which provides an efficient data pipeline between memory of compute nodes and storage. Based on the hybrid hierarchy storage architecture, we design and implement a flexible I/O stack shown in Fig. 4. Several key components including user-level client, I/O path management, RDMA based RPC communication and request scheduling are designed mainly for efficiently moving data into burst buffer. Unified storage management service is in charge of providing single namespace for ease of use.

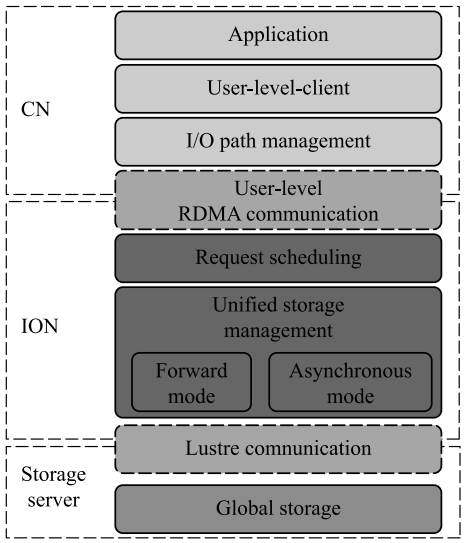


Fig. 4 I/O stack of H^2FS

Each layer of I/O stack implements a specific function module in the I/O request handling path from applications to storage devices in H^2FS. These modules effectively improve the characteristics of storage system, such as performance, scalability, reliability and ease-of-use. Table 1 lists the contributions of these components in I/O stack.

Table 1 Contributions of components in I/O stack

	Performance	Scalability	Ease-of-use	reliability
User-level client		√	√	
I/O path management				√
RPC communication	√			
Request scheduling	√	√		
Unified storage management	√	√	√	√

4.1 User-level client

User-level client is adopted in H^2FS with conventional POSIX interface which simplifies the file system protocol. User-level client cuts out client cache so as to reduce computing noise produced by I/O operation, which solves the inconsistency problem introduced by the huge number of clients in parallel file system.

I/O forwarding is implemented by user-level client within HVN. At the same time, user-level client supports optimization of data layout, which includes querying file's information, finding healthy DPUs and guiding metadata service to choose storage position, i.e., local storage or shared storage, for a new file. By using data layout interface, clients can balance the overhead among different DPUs to prevent multiple files locating on one single DPU, which may form an access "hot point" and affect the application's execution time. In such a way, the scalability of H^2FS is improved.

4.2 I/O path manager

I/O Path manager is in charge of the mapping from compute nodes to a group of DPUs according to the topology of interconnection network. It maintains the status of I/O path and chooses proper DPUs for clients. Two mapping modes are supported in our I/O path management, i.e., static mapping and dynamic mapping.

In static mapping mode, the mapping relationship is determined by network topology when system is initialized, or specified by applications. That is to say, application can specify the mapping according to I/O pattern, with policy interface. Static mapping provides a way for application and pro-

grammer to guide I/O path selection directly.

Dynamic mapping selects I/O path by the weight of each path that reflects the real-time overhead of DPUs. DPUs transports the path weight to compute nodes in response message of an I/O request. If the overhead difference of different DPUs is in reasonable range, round-robin policy is taken by H^{2}FS to specify DPUs for computing nodes, which is called static balance policy. On the contrary, if the difference is quite large, dynamic mapping adjusts I/O path and allocates DPUs for every file dynamically to avoid performance decline caused by congestion.

From the point of fault tolerance, path manager monitors the status of I/O path to detect DPUs failures and it can choose healthy DPUs to replace the failed one.

4.3 RPC communication

RPC implements the data transportation between compute nodes and DPUs, based on our proprietary interconnection network. For different communication patterns of requests in H^{2}FS, RPC makes use of multiple RDMA engines [11] in NIC to implement data transportation protocol efficiently. RPC combines control package and RDMA, and transports messages concurrently. Since the length of metadata request is usually quite short, the metadata transportation protocol with low latency is designed to get better latency of control message. As a consequence, the response time of metadata request is reduced and the transportation efficiency is optimized.

There are three optimizations of RPC:

1) Registered memory pool We register a memory pool before it is used for communication. The pool is divided into multiple buffer chains with different sizes, and each chain has a number of buffers. The buffer with appropriate size is located according to the requirement of the transportation. In such a way, the time overhead of memory register is reduced, the transportation performance is thus improved.
2) Buffer management RPC implements buffer zero-copy, in which the data of application is transferred directly by RDMA operation. In such a way, the memory copy between user space and kernel space is eliminated for high efficiency.
3) Metadata request combination The message of metadata request is short, and it is sensitive to the communication latency. So, several metadata requests for the same target are combined into one RDMA buffer and the transportation can be finished within one single control message. Thus the total number of transportations of these requests is reduced and the efficiency of transportation is improved.

4.4 Request scheduling

In large scale parallel computers, there are plenty of jobs running concurrently which issue data requests for lots of files. The I/O balance among different jobs has great impact on the running time of these jobs. So the I/O requests from multiple jobs, nodes and tasks must be scheduled carefully to guarantee the throughput.

The scheduling in H^{2}FS is organized in two layers including macro-scheduling and micro-scheduling. The macro-scheduling decides the location of files in H^{2}FS, which can be created in DPUs or shared storage and migrated between them. The micro-scheduling selects requests from request queues to serve, which also determines the order of request processing in DPUs or storage server. In order to satisfy different demands on scheduling, H^{2}FS provides several scheduling policies including FIFO scheduling, fairness scheduling, priority scheduling and synthesis scheduling.

4.5 Unified storage management

The I/O efficiency and the data reliability are at odds commonly. To solve such problem, we adopt unified storage management technology which takes full advantage of parallel processing of local storages and reliability of shared storage. The unified storage management framework manages shared storage and local storage and combines these two kinds of storage resource, which constructs an effective and reliable unified namespace. To improve the applicability, two modes are provided in unified storage management: synchronous mode and asynchronous mode. In synchronous mode, the accesses to unified namespace will be pre-processed and forwarded to the shared storage. The main difference between asynchronous mode of unified storage management and traditional direct accessing mode of shared storage is the I/O gathering process. We recognize and gather some I/O requests together during the forwarding stage to improve the overall throughput.

In asynchronous mode, the output data is firstly scheduled in local storage space, and an asynchronous thread is started to transport the data to shared storage in background. Due to the affinity and parallel nature of local storage, the I/O conflict between different tasks can be alleviated and the overall

I/O performance can be improved efficiently. If computation and I/O operations of tasks are interleaved, asynchronous accelerating mode will make them overlap well and further improve the data access performance. There are two synchronization patterns for different data consistency requirements: close-forced synchronization and non-close-forced synchronization.

In close-forced synchronization, the synchronization thread is triggered as soon as output data is transferred. And the process makes sure that the data on local storage and shared storage are consistent when that file is closed.

While in non-close-forced synchronization, the synchronization thread is triggered as soon as output data is transferred. And the consistence of data in local storage and shared storage might not be guaranteed when that file is closed. There is a short period of time between the file closing and the completion of synchronization. However, such pattern reduces the waiting time for file closing and the running time for that job.

In addition, unified storage management provides some fault tolerance methods. For example, if the SSD in ION failed and there is a copy for the data file in shared storage, the read request for this file will be migrated to shared storage. In another case, if a local SSD is full and there is a write request which continues to write on this SSD, unified storage management will migrate the request to the shared storage, and a synchronous process for this file is triggered to move other parts of the file to the shared storage at the same time.

5 Evaluation

5.1 Evaluation environment

The evaluation is running on MilkyWay-2 storage system with 256 ION, 4 MDSs, 60 OSSs and 30 disk arrays. Two OSSs form a failover pair and each OSS has two Intel Xeon Ivy Bridge CPUs with 12 cores clocked at 2.2 GHz and 64 GB memory.

We demonstrate the high performance of I/O system with a comprehensive evaluation that consists of a number of widely used parallel I/O benchmarks and some typical I/O intensive applications. The benchmarks we use in this evaluation are *IOR* and *mdtest*. *IOR* is used for performance test and *mdtest* is used for metadata test. The real I/O intensive application we use is a seismic data processing software Geoeast which is developed by Bureau of Geophysics Prospecting in China National Petroleum Corporation (CNPC BGP).

5.2 Evaluation results

5.2.1 Evaluation of ION

In each ION, there are two PCIe SSDs with total capacity of 2 TB. This test is running on one ION with different number of processes. The file size and the read/write block are set to 16 GB and 512 KB respectively.

The performance of ION is shown in Fig. 5. With the number of testing processes increasing from 1 to 64, the read/write bandwidth increases accordingly. When the number of processes is 32, we got the best performance for both read and write operations, which are 3.2 GBps and 2.8 GBps respectively.

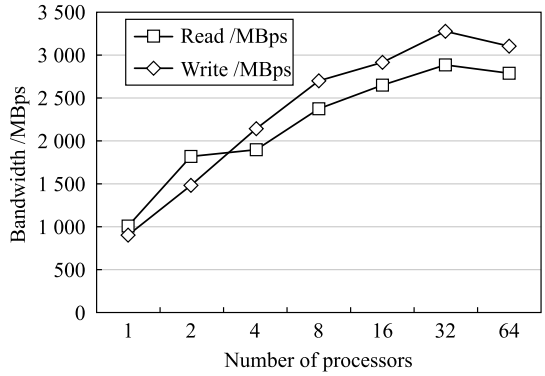


Fig. 5 Result on single ION

To evaluate the scalability of ION, we conducted some further experiments to show the average I/O bandwidth with increasing number of ION. The number of processes running on each ION varies from 1 to 32. As shown in Fig. 6, the I/O bandwidth is above 600 GBps with 256 ION, which proves that the ION has good performance and scalability.

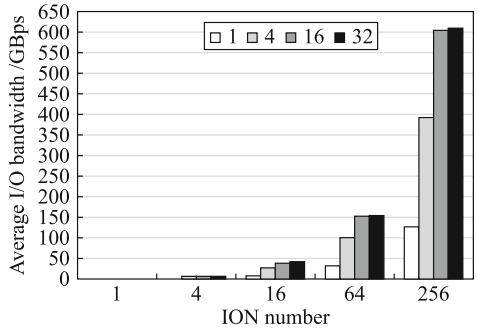


Fig. 6 Scalability of ION

5.2.2 Evaluation of the whole storage system

The evaluation of the whole storage system includes metadata test and file write test.

The result of metadata test is shown in Fig. 7. With syn-

chronous forwarding, the best result for file creation is about 30 000/s which is the sustained metadata performance of shared storage. Meanwhile, with asynchronous accelerating, the performance scales better as the number of clients increase, and the best result is nearly 81 000/s.

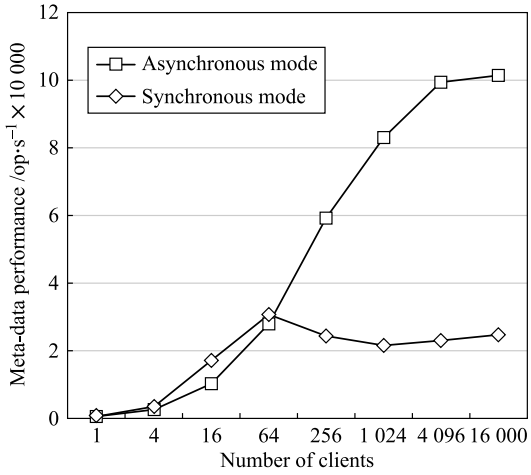


Fig. 7 Results of *mdtest*

The result of file write test is shown in Fig. 8, in which each process has its own file. With synchronous forwarding, the best write bandwidth is about 50 GBps when the number of clients is 1 024. With asynchronous accelerating, the best write bandwidth is near 500 GBps when the number of clients is 16 000. Similarly, the bandwidth also has better scalability than synchronous forwarding.

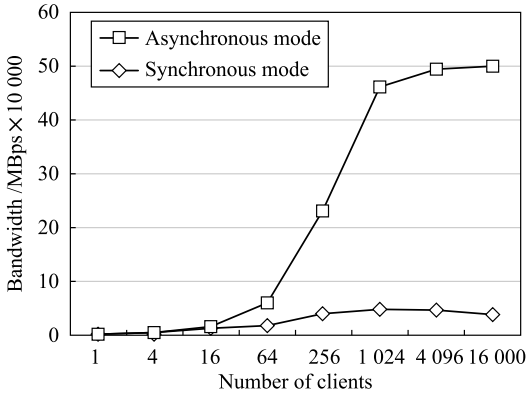


Fig. 8 I/O bandwidth of H^{2}FS

5.2.3 Evaluation of typical applications

We use a seismic data processing software Geoeast to test the performance for typical I/O intensive applications in MilkyWay-2 supercomputer storage system. Geoeast is developed by BGP, the largest seismic data processing service provider in China. We extract the I/O mode of Geoeast and run the main process in H^{2}FS architecture, the result is shown in Fig. 9.

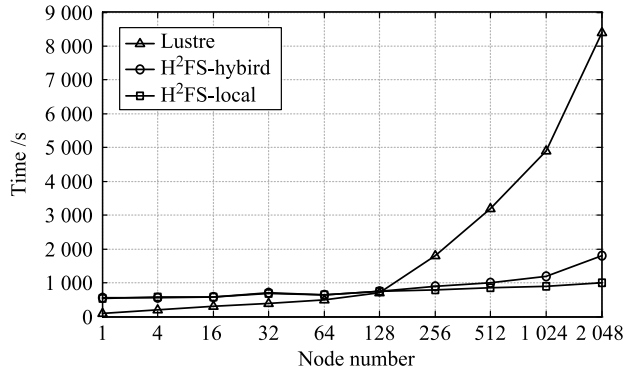


Fig. 9 Seismic data processing performance with H^{2}FS

In the test, we compare three I/O modes to run Data-In and Data-Out Modules in Geoeast. Among them, Lustre mode is the traditional mode, i.e., reading and writing data directly from and to shared storage. In H^{2}FS-local mode the data is stored only in local storage space, and in H^{2}FS-hybrid mode, the data is stored both in local and global storage space. The number of nodes involved in the tasks changes from 1 to 2 048 and the data processed by every node is around 20 GB.

The results show that two H^{2}FS modes provide much better scalability than Lustre at larger scale for its maximum locality. In the situation of 2 048 nodes, H^{2}FS can reduce the execution time by 70% compared to the traditional Lustre mode.

The above tests demonstrate the advantages of the architecture of H^{2}FS, especially asynchronous accelerating, improve the performance and scalability of the storage system effectively.

6 Related work

In HPC field, storage system has been considered more and more important by both academic communities and the industries. To provide I/O services with high performance and high reliability, distributed objective storage architecture and file systems, such as Luster [10], Panasas [12], etc., are often utilized in supercomputers. According to the recent top 500 lists of supercomputers [13], Lustre file system dominates the market. BlueGene/Q [14], Cray Titan [15,16], Jaguar [17] and K computer [18] all employ Lustre file system to construct the I/O system. Nevertheless, with the increasing client number in large scale supercomputer, the metadata performance provided by the single MDS server is becoming a bottleneck problem [19] in Lustre file system. The scalability and reliability of distributed file system in supercomputer

also face severe challenges to keep the usability of large scale I/O system [20,21].

To solve the existing problems, many researches are conducted and many efforts are made to find proper ways to improve the overall I/O system performance. In [22], an I/O forward scheme is introduced in BlueGene/P I/O system. By setting burst buffer in ION, the data transferred from compute nodes are collected through the forwarding software and the overall throughput of I/O system could be increased effectively. Another asynchronous data transfer method named Delayed Commit Protocol is proposed in [23], to reduce I/O latency in parallel file system. They remove the synchronous local write process from the critical path of an update operation, and hand it over to the file system as background processing. This method can effectively improve the I/O performance in the one layer storage architecture. The asynchronous mode in our H^{2}FS has the similar idea, but it focuses on the data transferring between two storage layers and keeping the data consistency and usability during the transfer process.

As for optimization in file system, NFS v4 compounds several RPC requests for a single operation into one RPC which reduces the number of network requests [24,25]. Carns takes collective communications of servers to avoid the consistency checking among clients [26]. In [27], the operation of file create, metadata handle and metadata setattr are compounded. Meanwhile, in [28], metadata synchronization is delayed when metadata server is busy.

Another hierarchical I/O system is realized in K computer [29]. The file system in K computer also divides the I/O system into two parts and manages these two parts with local file system and global file system respectively. The local file system holds the data under or waiting for execution and the global file system stores the input and output data of jobs. The data transfer between local and global file systems could be performed automatically by staging functions which are specified by users in a job script. By this means, the burden on shared storage is alleviated effectively.

The technology of user-level I/O interface is often used to improve the ease of use in storage system as well. In adaptable I/O system (ADIOS) [30,31], an external XML file is utilized to describe data information, e.g., data type, data hierarchy, and the way of handling data. Users can change the XML setting to adjust the I/O behavior without recompiling their source code. By employing the level of adaptability, the whole I/O system becomes more flexible and easier to use. Parallel netCDF is a scientific I/O interface allowing parallel writing to the netCDF datasets, thus enables gains from collective I/O optimizations [32].

Some similar ideas are adopted in our H^{2}FS, the main difference of our work comparing with other's related works is that we provide a unified file system among hierarchy storage structure, provide more functions and more ways for users to tune the I/O process in use-level interfaces.

7 Conclusions

This paper describes the storage system of MilkyWay-2 supercomputer, which includes hardware and H^{2}FS. This system adopts hybrid storage devices including SSDs and disk arrays and is layered by ION and storage servers. H^{2}FS cooperates local storage of ION and shared storage together into a dynamic single namespace. The local storage of SSD devices in ION provides high burst performance and disk arrays in shared storage provide enough storage space and sustaining performance. The evaluation of the storage system shows its high performance and good scalability.

In the future, we will keep working on this storage system with more storage devices and optimization on H^{2}FS, which will make more applications benefited from our storage architecture.

Acknowledgements This work was supported by the National High-Tech Research & Development Program of China (863 Program) (2012AA01A301), and by the National Natural Science Foundation of China (Grant Nos. 61120106005, 61202118, 61303187).

References

1. Franks B. Taming the Big Data Tidal Wave: Finding Opportunities in Huge Data Streams with Advanced Analytics. www.wiley.com, 2012
2. Verta O, Mastroianni C, Talia D. A super-peer model for resource discovery services in large-scale grids. Future Generation Computer Systems, 2005, 21(8): 1235–1248
3. Bent J, Grider G, Kettering Br, Manzanares A, McClelland M, Torres A, Torrez A. Storage challenges at Los Alamos National Lab. In: Proceedings of the 2012 Symposium on Massive Storage Systems and Technologies. 2012: 1–5
4. Watson R W, Coyne R A. The parallel I/O architecture of the high-performance storage system. In: Proceedings of the 14th IEEE Symposium on Mass Storage Systems. 1995, 27–44
5. Lofstead J, Zheng F, Liu Q, Klasky S, Oldfield R, Kordenbrock T, Schwan K, Wolf M. Managing variability in the IO performance of petascale storage system. IEEE Computer Society, 2010: 1–12
6. Zhuge H. The Knowledge Grid. Singapore: World Scientific, 2004
7. Oldfield R A, Maccabe A B, Arunagiri S, Kordenbrock T, Riesen R, Ward L, Widener P. Lightweight I/O for scientific applications. Technical Report of Sandia National Laboratories, 2006, 1–11
8. Liu N, Cope J, Carns P H, Carothers C D, Ross R B, Grider G, Crume A, Maltzahn C. On the role of burst buffers in leadership-class storage

systems. In: Proceedings of the 2012 Symposium on Massive Storage Systems and Technologies. 2012: 1–11

9. Zhou E Q, Lu Y T, Zhang W, Dong Y. H^2FS: a hybrid hierarchy file-system for scalable data-intensive computing for HPC systems. Poster paper in International Supercomputing Conference. 2013
10. Lustre: A scalable, high-performance file system. Cluster File Systems Inc. Whitepaper, Version 1.0, November 2002. http://www.lustre.org/docs/white paper.pdf
11. Xie M, Lu Y T, Liu L, Cao H J, Yang X J. Implementation and evaluation of network interface and message passing services for TianHe-1A supercomputer. In: Proceedings of the 19th Annual IEEE Symposium on High Performance Interconnects. 2011, 78–86
12. Welch B, Unangst M, Abbasi Z, Gibson G, Mueller B, Small J, Zelenka J, Zhou B. Scalable performance of the panasas parallel file system. FAST, 2008, 8: 1–17
13. Top500 Lists, http://www.top500.org/lists/
14. Ryu K D, Inglett T A, Bellofatto R, Blocksome, M. A, Gooding T, Kumar S, Mamidala A R, Megerian, M G, Miller S, Nelson M T, Rosenburg B, Smith B, Van O J, Wang A, Wisniewski R W. IBM Blue Gene/Q system software stack. IBM Journal of Research and Development, 2013, 57(1/2): 1–12
15. Rogers J. Power efficiency and performance with ORNL's cray XK7 Titan. Star Craft Companion, 2012: 1040–1050
16. Yu W, Vetter J S, Oral H S. Performance characterization and optimization of parallel i/o on the cray XT. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing. 2008, 1–11
17. Yu W, Oral S, Vetter J, Barrett R. Efficiency evaluation of Ccray XT parallel IO stack. Cray User Group Meeting, 2007, 1–9
18. Miyazaki H, Kusano Y, Shinjou N, et al. Overview of the K computer system. Fujitsu Scientific and Technical Journal, 2012, 48
19. Xing J, Xiong J, Sun N, Jie M. Adaptive and scalable metadata management to support a trillion files. In: Proceedings of the ACM Conference on High Performance Computing Networking, Storage and Analysis. 2009, 26: 1–11
20. Surendra B, Chou J, Rübel O, Prabhat, Karimabadi H, Daughton W S, Roytershteyn V, Bethel E W, Howison M, Hsu K J, Lin K W, Shoshani A, Uselton A, Wu K. Parallel I/O, analysis, and visualization of a trillion particle simulation. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press, 2012, 59: 1–12
21. Payne M, Widener P, Wolf M, Abbasi H, McManus S, Bridges P G, Schwan K. Exploiting latent I/O asynchrony in petascale science applications. In: Proceedings of the 4th IEEE International Conference on EScience. 2008: 410–411
22. Ali N, Carns P, Iskra K, Kimpe D, Lang S, Latham R, Ross R, Ward L, Sadayappan P. Scalable I/O forwarding framework for high-performance computing systems. In: Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops. 2009, 1–10
23. Lu Y Y, Shu J W, Li S, Yi L T. Accelerating distributed updates with asynchronous ordered writes in a parallel file system. In: Proceedings of the 2012 IEEE International Conference on Cluster Computing. 2012, 302–310
24. Shepler S, Callaghan B, Robinson D, Thurlow R, Sun Microsystems Inc., Beame C, Hummingbird Ltd., Eisler M, Doveck D, Network Appliance Inc. Network file system version 4 protocol. Network, 2003, 3530
25. Goodson G, Welch B, Halevy B, Black D, Adamson A. NFSv4 pNFS extensions. Technical Report, 2005
26. Carns P H, Settlemyer B W, Ligon W B III. Using server-to-server communication in parallel file systems to simplify consistency and improve performance. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing. 2008, 6
27. Devulapalli A, Ohio P W. File creation strategies in a distributed metadata file system. In: Proceedings of the 2007 IEEE International on Parallel and Distributed Processing Symposium. 2007, 1–10
28. Carns P, Lang S, Ross R, Vilayannur M, Kunkel J, Ludwig T. Small-file access in parallel file systems. In: Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing. 2009, 1–11
29. Sakai K, Sumimoto S, Kurokawa M. High-performance and highly reliable file system for the K computer. FUJITSU Science Technology, 2012, 48(3): 302–209
30. Liu Q, Klasky S, Podhorszki N, Lofstead H, Abbasi C S, Chang J, Cummings D, Dinakar C, Docan S, Ethier R, Grout T, Kordenbrock Z, Lin X, Ma R, Oldfield M, Parashar A, Romosan N, Samatova K, Schwan A, Shoshani Y, Tian M, Wolf W, Yu F, Zhang F, Zheng F. ADIOS: powering I/O to extreme scale computing. 1–6
31. Lofstead J, Zheng F, Klasky S, Schwan K. Adaptable, metadata rich IO methods for portable high performance IO. In: Proceedings of the 2009 International Parallel and Distributed Processing. 2009, 1–10
32. Li J, Liao W, Choudhary A, Ross R, Thakur R, Gropp W, Latham R, Siegel A, Gallagher B, Zingale M. Parallel netCDF: a high-performance scientific I/O interface. In: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing. 2003, 39

Weixia Xu received his MS in computer science from National University of Defense Technology (NUDT), China in 1992. Currently he is a professor at this university. His research interests include system structure of parallel computing and system structure of multi-core microprocessor. He has been working in College of Computer in NUDT since 1984. He is the deputy chief designer of TianHe-1 supercomputer and MilkyWay-2 supercomputer.

Yutong Lu received her MS and PhD in computer science from National University of Defense Technology (NUDT), China. Currently she is a professor at the university. Her research interests include parallel system management, high speed communication, distributed file systems, and advanced programming environments with MPI. She is a director designer of MilkyWay-2 supercomputer.

Qiong Li is currently a professor in College of Computer, National University of Defense Technology (NUDT), China. She received her MS and PhD in computer science from NUDT, China. Her current research interests include high performance computing and storage technology.

Enqiang Zhou is currently an associate professor in College of Computer, National University of Defense Technology (NUDT), China. His current research interests include high performance computing, storage technology and distributed file system.

Zhenglong Song is currently an assistant professor in College of Computer, National University of Defense Technology (NUDT), China. His current research interests include high performance computing and storage technology.

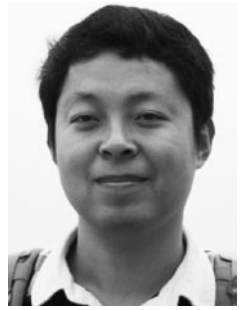
Yong Dong is currently an assistant professor in College of Computer, National University of Defense Technology (NUDT), China. His current research interests include high performance computing and distributed file system.

Wei Zhang received his BS, MS and PhD in computer science from College of Computer, National University of Defense Technology (NUDT), China. He is currently an assistant professor in College of Computer, NUDT, China. His current research interests include high performance computing and distributed simulation.

Dengping Wei received her MS and PhD in computer science from National University of Defense Technology (NUDT), China, in 2005 and 2011, respectively. Currently she is an assistant professor in College of Computer, NUDT, China. Her research interests mainly involve high performance computing, storage technology and big data.

Xiaoming Zhang is currently an associate professor in computer science in National University of Defense Technology (NUDT), China. His research interests mainly involve high performance computing and interconnection network.

Haitao Chen is an associate professor in computer science in National University of Defense Technology (NUDT), China. His research interests mainly involve high performance computing, performance evaluation.



Jianying Xing received his MS and PhD in computer science from National University of Defense Technology (NUDT), China in 2007 and 2011, respectively. Currently he is an assistant professor in College of Computer, NUDT, China. His research interests mainly involve high performance computing, performance evaluation.

Yuan Yuan received his MS and PhD in computer science from National University of Defense Technology (NUDT), China in 2007 and 2011, respectively. Currently he is an assistant professor in College of Computer, NUDT, China. His research interests mainly involve high performance computing, performance evaluation.