Performance optimization method for distributed storage system based on hybrid storage device

Performance optimization method for distributed storage system based on hybrid storage device

Chen-han WU[1]

[1] Department of Software Engineering, Chengdu University of Information Technology, Chengdu 610225, China

E-mail: romdubu@foxmail.com

Abstract: In the era of big data and artificial intelligence, the storage data of data centers is growing rapidly. Big data storage systems with high-performance computing require high-speed data access and greater storage capacity. However, traditional storage technology based on hard disk media (HDDs) has been difficult to meet the current demand for high-performance storage, solid state drives (SSDs)' cost will be a lot for storing massive amounts of data. Therefore, in response to the above problems, this paper proposes a hierarchical hybrid storage technology for distributed file system, including an adaptive file migration strategy and client caching mechanism. Through experimental verification, the proposed technology fully combines the high I/O throughput characteristics of SSDs and the large capacity characteristics of disks, thereby providing a cost-effective high-performance distributed storage solution, and when processing real-time data, it has higher access performance.

1. Introduction

With the rapid growth of data volume in high-performance computing and big data storage systems, data storage and processing are facing increasingly severe challenges. Distributed file systems oriented to disk media such as Lustre[1], Ceph[2], and GlusterFS[3] are difficult to meet the needs of future high-performance storage. The distributed file system based on the optimized design of high-performance storage media is expensive as a whole and not suitable for large-scale use. The hierarchical storage technology integrates devices with different performance, capacity, and prices[4]. Compared with traditional storage systems, it can provide a high-performance, large-capacity, and low-cost storage environment. TH-TS[5] realizes the hierarchical storage of files between high-end data servers and low-end data servers, but it has the problem of low space utilization of high-end data servers. Hierarchical storage systems IBM STEPS[6], IBM Tivoli Storage Manager[7], EMC DiskXtender[8], EMC Symmetrix DMX[9] all implement customizable migration strategies and utilize the scalability and metadata information of the file system, to achieve efficient hierarchical storage. However, these systems also have shortcomings in terms of performance and transparency. Hierarchical storage system of user space is difficult to handle the load balancing of the storage space; it is difficult to block the process of issuing access requests during migration; its own performance is limited to the throughput performance of the client, and cannot meet the comprehensive requirements of high parallelism, high bandwidth, low latency, and large storage capacity.

This paper proposes a hierarchical hybrid storage technology for distributed file system, and realizes a distributed file system DHFS (distributed hierarchical file system). DHFS is combined with an adaptive file migration strategy to provide hierarchical storage management that is transparent to users. For the problems of low utilization of client resources and long response time during data transmission, the client cache mechanism is used to improve client storage access performance. The test results show that the proposed technology fully combines the high I/O throughput characteristics of SSDs and the large capacity characteristics of disks, thereby providing a cost-effective high-performance distributed storage solution, and make full use of client resources to provide high online data access performance and fast response time.

2. Design and Implementation of Hierarchical Hybrid Storage Scheme

The overall architecture design of the DHFS-based hierarchical hybrid storage system is shown in Figure 1. It mainly includes three parts: client, data server and metadata server (MDS). The DHFS client mounts to the data storage server managed by MDS through the VFS interface on the node that provides POSIX semantics, so as to communicate with data server for file access like a local Linux file system.

MDS is responsible for managing the metadata of all files in the system, as well as the scheduling control of file storage and migration. The metadata management module (Meta-DM) is the main function module used by DHFS to manage the entire file system, including managing the file metadata in all data servers in the system, and storing and maintaining the metadata information of all files in the system in the Metadata pool. The hierarchical storage management module (HSM) includes log management module (Log-M), system monitoring module (Sys-mon), and strategy execution module. Log-M includes operation logs, monitoring logs, checkpoint logs, etc. Sys-mon is used to monitor the usage and IO conditions of each storage node. The function of the strategy execution module includes data value evaluation and data migration execution. Data value evaluation is to dynamically evaluate the value of stored data and manage data storage; data migration execution is to perform data migration on servers with the same and different levels of data. Between storage scheduling to ensure the load balance of the entire system. So as to achieve high-performance access to data in the entire system and load balancing of data storage. When the accessed file is to be migrated, HSM initiates a file relocation request to the data server.

The data server is a cluster system, including high-performance storage server (HPS), large capacity storage server (LCS); HPS is an SSD-based server, used to store frequently accessed and High-value files; LCS is an HDD-based server used for infrequently accessed files and backup of massive files. The data server includes system information module and data server management module (DM). System information module provides the system information of each storage node of the data server, and the DM provides monitoring and operation of the file system and files under the storage node.

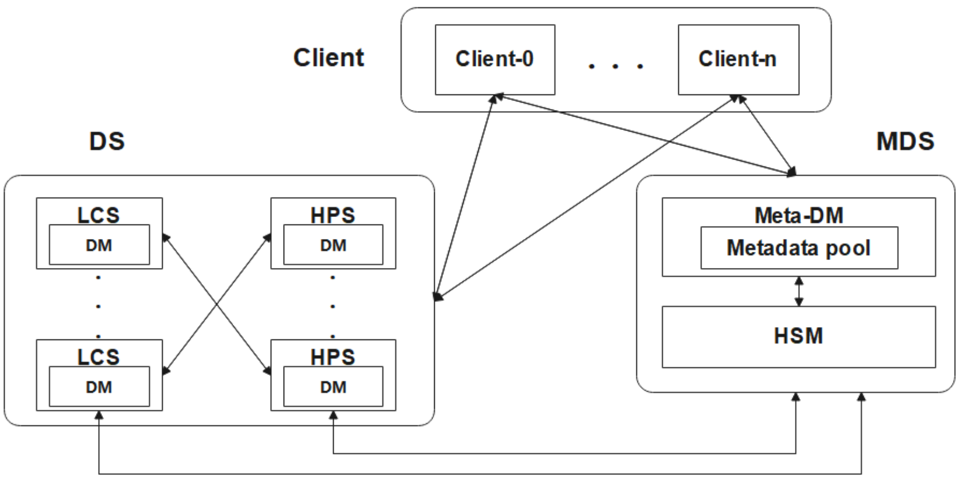


Fig.1 DHFS-based hierarchical hybrid storage system architecture

When the client transmits data, there are problems such as high network delay and low resource utilization. Therefore, set up a client cache on DHFS client to improve the I/O performance of the application client.

The cache mechanism is the relay station of the client. The client cache obtains the client data request through the VFS interface, and transmits the data requested by the application program through the transmission channel. The physical connection between the client and the remote server is processed in the background, and using the DIRECT_IO option when mounting the file system client, bypass the page cache of the VFS. All data will be temporarily stored in the cache space. The file in the cache space is converted into a hash value through a hash algorithm as a mapping of the file logical path, and the hash value is used as a key Value, uniquely corresponds to a file logical path, and divides the target file into multiple storage points to achieve load balancing. The multiple storage points under the target file are uniquely named according to the file read and write pointer position. The upper-level application can access the data entity corresponding to the cache space through the file logical path corresponding to the file view.

2.1. Adaptive file migration strategy

HSM monitors the data access of the entire system, and hands the real-time data access information to the strategy execution module, and the strategy execution module judges the migration of the accessed files according to the file migration mechanism. The adaptive file migration strategy of text design is to evaluate the value of files based on information such as file access frequency, recent access time, and file size within the business cycle T. The initial value of T is the empirical value based on the actual business system visit volume V_0, and is dynamically adjusted according to the system visit volume V in each cycle. The calculation formula of T in the nth business cycle is as follows:

$$T = \frac{V_0 \times \sum_{i=1}^{n} T_n}{\sum_{i=1}^{n} V_n} \tag{1}$$

The calculation formula for document value evaluation in the business cycle is as follows:

$$\mathrm{FP} = \frac{FAV + \sum_{k=1}^{m} \varepsilon_k FV_k}{Lat \times T} \tag{2}$$

Among them, FV_k is the inherent file value of the fixed attributes of the file based on the empirical value ε_k; Lat is the difference between the most recent file access time and the current time of the system; FAV is the access value of the file in the cycle, comprehensively evaluated by the ratio of read-write access, the specific formula is:

$$FAV = \frac{\mathrm{W}}{\mathrm{R}+\mathrm{W}} \times Vw + \frac{\mathrm{R}}{\mathrm{R}+\mathrm{W}} \times Vr \tag{3}$$

According to the calculated file value and the set threshold value, the nature of the file can be judged. The files whose file types meet the migration conditions are added to the queue to be migrated, and the file migration between the servers is completed by communicating with the corresponding migration destination server. Each storage server maintains an IO monitoring thread to monitor the IO queue in the storage server. When the IO queue depth is greater than the set threshold, the file migration thread in the server will be suspended until the IO queue is less than the set threshold.

Sys-mon monitors the usage of each storage node, and sets storage space occupancy thresholds for all levels of storage to ensure that the storage capacity of all levels is at a safe value. When the HPS' storage space is not at a safe value, it will trigger the downgrade migration or upgrade migration of the data. At the same time, in order to achieve load balancing of various storage devices, migration scheduling among server nodes at the same level is implemented. The performance attributes of the device include CPU load, memory usage, disk usage, IO queue, bandwidth, latency, etc. The calculation formula for the performance of storage nodes at the same level is as follows:

$$SP_i = \sum_{k=1}^{n} \varepsilon_k SV_k \tag{4}$$

Among them, SV_k is a certain performance attribute of the equipment, and ε_k is the corresponding calculation parameter. Sys-mon sorts the calculation results, divides the storage priority of the server nodes at the same level, and delivers the results to the strategy execution module. When the load is unbalanced in the same level, trigger the migration scheduling between nodes of the same level.

2.2. *Efficient and transparent metadata update method*

In order to perform operations on files correctly, KHFS needs to continuously update the metadata information maintained by MDS, including the physical storage path of the file, the time of last access, and the time of last modification. Because the file migration mechanism changes the status information of the migrated files, the metadata update scope is not limited to all file read and write requests, and the traditional file system metadata update method cannot meet the requirements. For this reason, KHFS selects which file metadata to update through the file migration update mechanism and the information monitoring of the data storage node to ensure the consistency of the metadata information.

The file migration update mechanism records the migration file information in real time through a log type, and at the same time checks whether the data storage node is successfully migrated, and then updates the metadata in time. In the process of file migration, if the file is accessed by the upper application, the corresponding server will be notified through the data management module to suspend the file migration, and it will be judged whether to continue the migration until the file access is over, so as to ensure the file migration between storage servers. Transparency.

2.3. *Cache replacement strategy based on time sliding window*

In the real-time I/O scenario, the client's access mode will change dynamically. In order to process real-time changing data requests, this article provides a cache replacement strategy based on a time sliding window. The sliding window mechanism is as follows: the current time window records and collects the corresponding access information, and retains the cached data information in the previous window. When the cache space meets a specific trigger condition, the client cache runs the LRU algorithm, and selects the closest time point. The least used cache file is replaced.

The cache replacement strategy designed in this paper reduces the access frequency of files in historical access records by a certain percentage and puts them in a new window to ensure the timeliness of cached data. For file x, assuming that in consecutive window periods W_i~W_j, x is stored in the cache space, the calculation formula for the access frequency of x in windows W_i~$W_j$is as follows:

$$FQ(x) = FQ_j(x) + \sum_{k=i}^{j} e^{-a(j-i)} FQ_i(x) \tag{5}$$

This strategy takes into account the proportion of historical access information, so as to avoid too much new data influx leading to a low cache hit rate; and considering the time locality of user access, the calculation weight of historical cache data in the next window is reduced proportionally, thereby reducing the impact on new cached data.

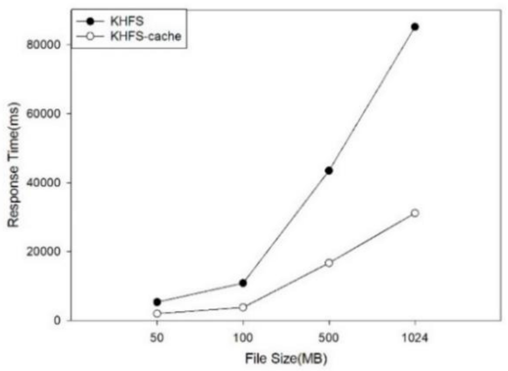
3. Test Results and Discussions

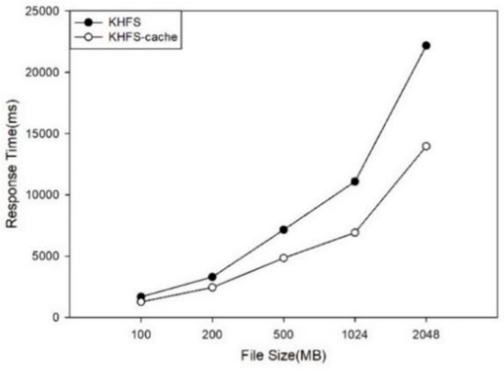
3.1. *test environment*

The hierarchical hybrid storage system based on the distributed file system DHFS consists of four parts, MDS, HPS, LCS, and the DHFS client. The server parameters are: The CPU model is Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz, the memory size is 8GB, and the disk is 4TB HDD. The server is connected through a gigabit switch. The HPS server has the same parameters as other servers, except that the disk is a 1T SSD disk. Client configuration: The CPU model is Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz, and the memory and hard disk sizes are 8GB and 120G respectively. The operating system of all devices is CentOS 7.0 x64 Linux, the kernel is Linux 4.4.241-rc8, and a special test tool designed according to the IO load characteristics of the distributed file system actually used is used. The compilation software used in the test is JDK 1.8.0_242 and GCC 7.5.0.

3.2. *Client-side caching effect analysis*

In order to verify the performance of the system client cache, the access performance of the DHFS native client and the DHFS cache client were compared and tested. When testing the performance of the DHFS native client, the client caching function is turned off to ensure that the test data is not cached in the client.





(a) Comparison of reading response time (b) Comparison of writing response time

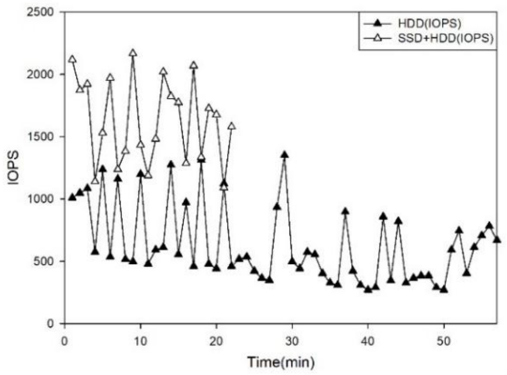
Fig.2 Comparison of access performance

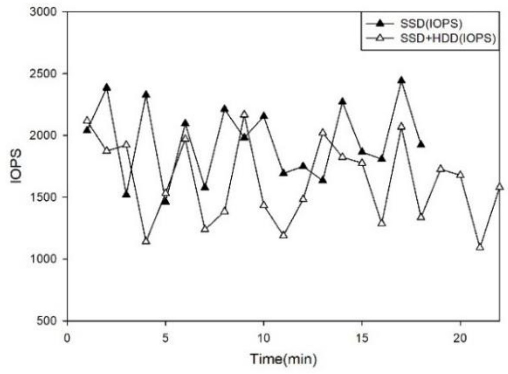
First, we tested 10 files with a size of 50MB~1GB, and randomly read about 5GB~100GB data response time to the cluster. The test result is shown in Figure 2-(a). Compared with the original DHFS, the response time of DHFS optimized by read buffer is reduced by up to 65.2%, and the average reduction is 63.1%. As shown in Figure 2-(b), we tested writing 100MB, 200MB, 500MB, 1GB, and 2GB files to the cluster. The test result is that the response time of the DHFS optimized by the write buffer is reduced by 24.7%~37.6% compared with the original DHFS. The results show that the client caching mechanism implemented in this paper can greatly improve the client's IO performance.

3.3. Performance comparison test of hybrid tiered storage

This section conducts a performance comparison test of full HDD, full SSD and SSD + HDD hierarchical storage. By comparing the overall performance of the hierarchical storage strategy and the unused hierarchical storage strategy, the impact of the tiered storage strategy on system performance is obtained. The test data of the experiment restores the real scene of the file system that meets the requirements of the tiered storage system evaluation, and generates all file access requests for this test.

In the full HDD environment, the pre-tested data is directly stored in LCS; in the full-SSD environment, the pre-tested data is directly stored in HPS without migration. In the SSD + HDD hybrid tiered storage test, the file migration strategy is set as follows: migrate files that have not been accessed in the last 10 days to a large-capacity storage server, and migrate files in descending order of file size. Through the same read and write test, the test results obtained are shown in Figures 3.





(a) Average IOPS of HDD and SSD+HDD (b) Average IOPS of SSD and SSD+HDD

Fig.3 Comparison of average IOPS

In a full HDD environment, the total test time is 57.45 minutes, and the average IOPS is 631.2. Under SSD + HDD hierarchical storage, the total test time is 22.26min, and the average IOPS is 1619. The test data shows that after the system introduces SSD and hierarchical storage functions, the overall IO performance is increased by 2.58 times.

In a full SSD environment, the total test time is 18.86 minutes, and the average IOPS is 1923.2. Under SSD + HDD hierarchical storage, the total test time is 22.26min, and the average IOPS is 1619. Experimental data shows that: on the hierarchical storage system of SSD + HDD, through the hierarchical storage of files, the performance can reach 84.7% of the system performance in a full SSD environment. It can be seen that through the hierarchical storage and management of data, the system only needs a small-capacity SSD and a large-capacity HDD to reach 84.7% of the system performance in a full SSD environment. At the same time, compared with a full SSD environment, the system only needs very low cost can provide more storage space.

4. Conclusion

This paper proposes a hierarchical hybrid storage technology for distributed file system, and implements a distributed file system DHFS. The file system interface provided by DHFS provides users with a unified file view. HSM realizes transparent hierarchical storage management for users and applications through calculation of the relative value of files, automatic judgment of data migration, and management of migration queues. It also provides a client-side caching mechanism to make full use of client-side resources to achieve high-efficiency and low-latency client-side storage access. The test results of the DHFS client show that the client caching mechanism can greatly improve the IO performance of the client. And using the trace generated by the real file system snapshot, the system performance was tested when the hierarchical storage strategy was used and the hierarchical storage strategy was not used on the system. The comparison of experimental results shows that the system fully combines the high I/O throughput characteristics of SSDs and the large capacity characteristics of disks to provide a high-performance distributed storage solution with reasonable cost performance.

Acknowledgments
The author is grateful to Lei YANG for proofreading this manuscript.

References

[1] Wang F, Oral H S, Shipman G M, et al. Understanding Lustre Internals[J]. Office of Scientific & Technical Information Technical Reports, 2009.
[2] Weil S A. Ceph: reliable, scalable, and high-performance distributed storage[J]. Santa Cruz, 2007.
[3] Gluster Whitepaper, 2010, “GlusterFS Architecture”, [Online]. Available: http://download.gluster.com/pub/gluster/documentation/Gluster Architecture.pdf. [Accessed: 22-Dec-2015].
[4] Advancing storage and information technology[EB/OL]. http://www.snia.org/education/dictionary/h/, 2013.
[5] Ao L, Yu D, Shu J, et al. A tiered storage system for massive data: TH-TS. Journal of Computer Research and Development, 2011, 48(6): 1089-1100.
[6] Askshat Verma, David Pease, Upendra Sharma, et al. An architecture for lifecycle management in very large file systems[C]. In: Mass Storage Systems and Technologies, Proceedings of 22nd IEEE/13th NASA Goddard Conference on. IEEE, 2005: 160-168.
[7] Charlotte Brooks, Peter McFarlane, Norbert Pott, et al. IBM tivoli storage management concepts [EB /OL]. http: //www.redbooks.ibm.com/redbooks/pdfs/sg244877, 2006.
[8] EMC DiskXtender [EB /OL]. http://china.emc.com/products/detail/software/diskxtender-unix-linux.htm, 2012.
[9] EMC Symmetrix DMX [EB /OL]. http: //www.emc.com/products/detail/hardware/symmetrix-unix-linux.htm, 2012.