Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2012, Article ID 835616, 10 pages
doi:10.1155/2012/835616

Research Article

An Improvement of Static Subtree Partitioning in Metadata Server Cluster

Tan Zhipeng, Zhou Wei, Sun Jianliang, Zhan Tian, and Cui Jie

Wuhan National Laboratory for Optoelectronics, School of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, China

Correspondence should be addressed to Tan Zhipeng, zhipengtan@163.com

Received 22 May 2012; Accepted 11 July 2012

Academic Editor: Liguo Zhang

Efficient metadata management is a key issue in distributed file systems. Currently a single metadata server (MDS) was used to provide metadata service in many distributed file systems. As a result the MDS is likely to be the bottleneck of the system with the expanding scale. To deal with this problem, MDS cluster has been advanced. There are subtree partitioning and hashing partitioning to realize MDS cluster and so on. In view of the strengths and weaknesses of the previous methods, the paper proposes a half-dynamic subtree strategy that supports the replication of metadata, the dynamic addition and failure recovery of MDS. Finally we implement it in the open source pNFS file system and demonstrate its high performance and availability.

1. Introduction

During the last two decades, tremendous development and growth have happened in the distributed parallel system. As a key component, the large-scale distributed parallel file system has been attracting a great deal of attention from both industry and academia. To improve the performance, scalability and availability of it become more and more important. NFS is the most common network file system in industry. However, with the rapid development of the internet, especially the emerging of Web 2.0 technology, the information is growing explosively [1]. Therefore, conventional network file system cannot meet this situation. In order to improve the performance and availability of the network file system, a number of distributed file systems appeared to separate the data and metadata management [2], by setting up the MDS metadata server (MDS) for metadata service and data servers (DS) for parallel data access [3]. A lot of distributed file systems such as GFS [4], TFS, and Lustre [5] all use this kind of architecture. Consequently, based on the NFSv4.0 protocol, the industry puts forward the NFSv4.1 [6] protocol and realizes it in the open source pNFS file system in 2006. Also the system based on the NFSv4.1 increases the performance, but single MDS design in pNFS suffers from the problems of single server bottleneck, single node failure, and MDS scalability. In order to solve this problem, we implement pNFS MDS cluster which can improve the scalability and fault tolerance of MDS.

To achieve MDS cluster, the entire namespace should be partitioned and mapped to different metadata servers. So far the popular methods can be classified into two categories: one is the directory subtree partitioning and the other is hashing-based methods. The former including static partitioning and dynamic partitioning. In static subtree partitioning [3], the entire namespace of the file system is partitioned into disparate subtrees. Despite the maturity, simplicity, and the popularity of the static partitioning, it is not able to cope with the load-balancing issue. Dynamic subtree partitioning [7] overcomes the drawback of the static partitioning by repartitioning the hot spots subtree using dynamic subtree partitioning. Yet dynamic partitioning suffers from frequent metadata movement which impacts the stability and increases the complexity of the system. The latter includes pure hashing which stores the corresponding metadata according to the file identifier (file pathname) hashing and normally the metadata could be found in only one step.Hashing method [8] fulfills the uniform distribution of metadata but damages the metadata locality,

the operation "ls" will be low efficiency. Moreover, hash value of the directory and the files under the directory will be changed due to the operation of renaming a folder, leading to large amounts of metadata movements. Another is called "LazyHybrid" which combines the advantages of subtree partitioning and pure algorithm. Meanwhile, lazy metadata migration and lazy updating of access control lists (ACLs) policies are adopted to offer more efficient metadata management. However, the system uses lazy relocation to relocate the metadata, migrating the metadata when being actually accessed. Although this kind of method can reduce the effect of performance due to instant migration, it raises the difficulty of MDS deletion. After the MLT was modified, an uncertain time is required to complete the migration of the deleting metadata.

We propose a solution based on the static subtree method which we called half-dynamic subtree partitioning strategy; we improve the scalability and availability by the metadata replication, MDS live join, and son on.

2. Related Work

In distributed file system, subtree partitioning and hashing partitioning are the common methods to partition the namespace of the system. And subtree partitioning is simple and natural.

Currently the MDS cluster [5] architecture has not been employed in some distributed file system. For example, Lustre only uses a MDS and a backup MDS, and it improves the performance of the MDS by delegating some mechanisms to client and DS (Data storage Server), such as client delegation and DS byte-range lock. However, GFS [4] from Google use another way, instead of storing the metadata in local file system in the directory format, the metadata is specially designed aiming at the demand of its applications. All the metadata are saved in the memory by prefix compression such that the performance of the MDS is greatly improved.

Few distributed storage systems in the industry environment employ the MDS cluster technology. Expect two systems: Storage Tank [9] from IBM and Panfs from Panasas [10], both adopt the simple static subtree partitioning to manage metadata. The static partitioning improves the ability of the metadata service in some degree but is short of effective load balancing.

The MDS cluster of distributed file system in academia tends to be more flexible and radical. Ceph [7], developed by University of California, Santa Cruz, applies a dynamic subtree partitioning strategy to implement MDS cluster and now the Ceph client has been integrated into Linux Kernel since 2.6.34. Dynamic subtree MDS cluster solution has incomparable advantages in scalability and load balance, but it is relatively difficult to realize. And the subtree division and migration make the system not very stable. Ceph is not suitable for production environment, but it is still significant to test and research on it.

Based on the analysis of those researches, we propose a way of half dynamic subtree partitioning strategy to manage metadata and realize the pNFS MDS cluster.

Table 1: MLT structure.

Bucket number	Subtree pathname	Metadata server IDs	Subtree status
0	/	1, 2, 3	0
1	/a/	2, 3, 1	0
2	/a/c/	3, 1, 2	0
3	/b/	1, 2, 3	0
4	/a/c/d/	2, 3, 1	0

3. Design and Implementation

The distributed file system is the core of large-scale storage system in cloud storage. With the purpose of providing safe and stable cloud file storage platform, it is vital to ensure high availability and high scalability during system design. Furthermore, all the mechanisms must be integrated into the system seamlessly without affecting the working of applications. These new demands bring out higher requirement of system design. Our system is designed to meet the following general goals.

(1) Multiple MDSs on line provide service simultaneously.

(2) New MDS node can be added into the MDS cluster automatically without influencing the running of the system.

(3) Supporting MDS fails. If we set the number of copy to 3, the system can endure at most two node down which contains the copy.

(4) MDS can be recovered on line and continue to provide service without influencing the running of the system.

(5) MDS cluster load balancing.

In order to reach the goals above, the paper proposes a replication-based subtree partitioning strategy. Our design not only offers high-performance metadata service by taking advantage of the static subtree partitioning, but also can tolerate MDS failure through replication technology.

3.1. MLT. In order to implement the MDS cluster, the metadata lookup table (MLT) is designed to partition the entire namespace of the file system (as shown in Table 1). Each item in the MLT records the partitioned subtree's pathname, the MDS that manages the subtree, and the current status of the subtree.

In order to provide service during MDS failure, the system provides metadata replication mechanism. As shown in Table 1, Each subtree contains three replications. The first ID represents the location of primary replication and the ther two are the location of slave replications.

Metadata replications also achieve load balance of metadata accessing. Meanwhile the availability and reliability of the MDS cluster are also improved.

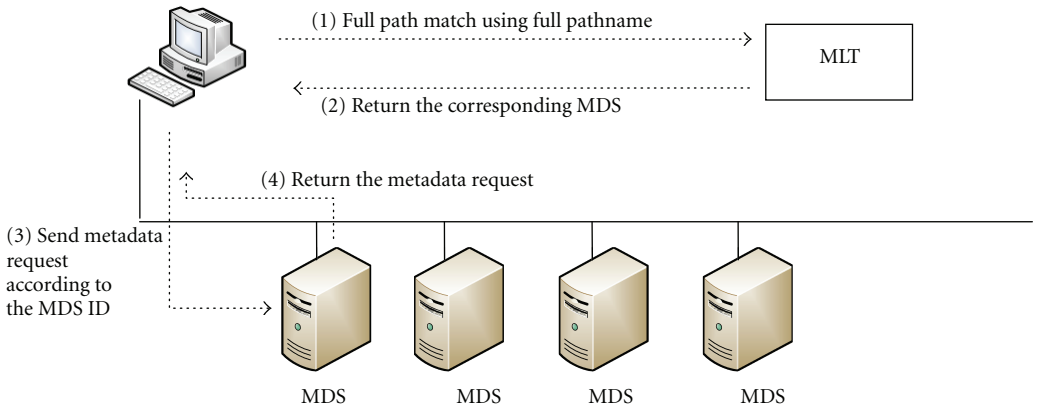

Figure 1: Metadata Access.

3.2. Metadata Access. All the clients and MDSs maintain the MLT. Figure 1 illustrates the client's accessing process. Firstly, the client makes full path matching between the cached MLT, and the full pathname of the file. Secondly find the corresponding MDS which the metadata belong to. Thirdly send the request to the MDS. Finally the correct metadata is obtained.

For instance, the current-client-cached MLT is showed in Table 1. The upper layer operation of the client is stat("/a/c/d/foo.txt",buf). The pNFS client makes longest path matching with the cached MLT. It can be found that the path "/a/c/d/" is managed by the subtree in bucket4. Then get a MDS that store the subtree, such as MDS2 and find the IP address of MDS2 according to its ID. Next send a request to MDS2. Once the MDS confirms that the metadata is managed by itself, it sends back the metadata information to the client.

4. pNFS MDS Cluster

In order to achieve the design goals and minimize the modification of open source pNFS file system. We have not changed the way of storing file metadata in pNFS and continue to use empty file to store file metadata. Then a half-dynamic subtree partitioning strategy is proposed to meet the goals.

Figure 2 shows the basic framework of the system. The MDS consists of metadata migration module, MLT maintenance module, load information collection module, failure detection module, primary selection module, new MDS join module, and pNFS's original modules. pNFS client consists of MLT maintenance module, MDS selection module, MDS connection maintenance module, and pNFS original modules. The communication middleware is Remote Procedure Call (RPC).

When a new node joins in, the main MDS cuts or copies a metadata subtree to the new node. The load information collection module is in charge of collecting load information of all the MDSs in cluster.

The failure detection module, new MDS addition module, and primary MDS selection module are mainly responsible for monitoring the cluster status including the node failure and new node addition. In addition, when the primary MDS is down, a new MDS will be selected.

The client's MLT maintenance module and MDS connection maintenance module are mainly used for tracking the changing status of the metadata cluster by refreshing the cached MLT and maintaining the sessions with MDSs. The session operations include creating session with the new nodes and destroying the stale session with the failed nodes.

4.1. MLT and Consistency Maintenance. MLT maintains the partition information of the file system. And in order to let the client find the metadata correctly, we must maintain the consistency of MLT tables both client side and MDS side should be considered.

All the MLTs cached in the MDSs must be updated when the MDS cluster is changing so as to enable the client to track the newest modifications. For instance, the MLT changes and the version upgrades when a new MDS is added into the cluster. The primary MDS actively pushes the newest MLT to all the MDSs one by one.

Instead of pushing the updated MLT from the MDS to the clients, lazy policy is employed so that the updating only occurs when there is interaction between client and MDS. Every request sent from the client encodes a version number of its current cached MLT. When the MDS receives the client's request, the version number of the MLT sent from the client is decoded first and is compared with the MDS's version number. If mismatch exists, then return the error code. The client receives the error code and obtains the newest MLT information from the primary MDS. Finally send the metadata request to the correct MDS.

4.2. Metadata Replica Maintenance. To improve the availability of the metadata service and the load balancing of metadata, metadata replication is imported so that if some MDSs are down, the system is still able to supply correct service by visiting the replication.

The design of the distributed system replication will involve a theorem called CAP which includes consistency, availability, and partition tolerance. Because the design of

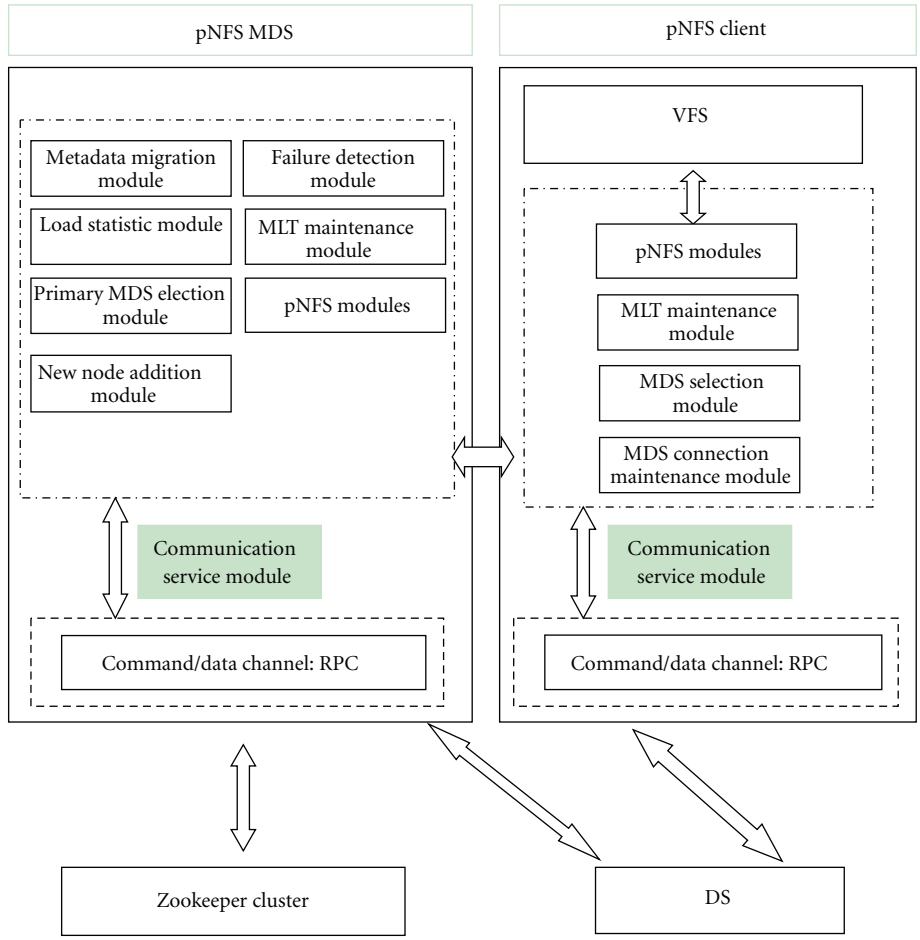

Figure 2: The architecture of metadata server cluster on pNFS.

pNFS file system is general file storage oriented, instead of the final consistency strategy, strong consistency strategy is adopted so that after the client's success writing, the subsequent request from the same client could read the newest value. However, the case that other clients read the replications which have not been updated during maintaining cannot be avoided.

Another issue in the process of maintaining replication consistency is how to coordinate the requests sent from different clients to a same MDS. In dynamo, different clients could operate the different replications for the same metadata in parallel. Despite the large improvement of writing availability, the problem of covering conflict due to the order of writing maybe caused. Dynamo utilizes the vector clock method to implement "last write win" strategy. The primary replication policy is used to solve the writing conflict. The primary replication is locked when sending the write request. The requests that fail to grab the lock of the primary replication keep waiting until the client which grabs the lock finishes its updating and then update their metadata.

The replica is maintained by the client. When modifying or creating metadata, the client obtains the write lock of the metadata's parent directory and updates the replicas in turn. Finally the lock is released. The process is showed in Figure 3.

4.3. New MDS Addition and Failure Recovery. The on-line new MDS addition is supported to enable the scalability of MDS cluster.

Before adding the new node, the IP address of the primary MDS must be got from the cluster and written into the configure file. The new node reads the configure file and interacts with the primary MDS during starting, triggering a subtree migration. The primary MDS selects the appropriate subtree migration and adds it into the new node according to the load information from the cluster and subtrees. Rsync is used as the migration tool.

The write operation in the migration process may lead to the inconsistent. We present a simple solution to handle this problem: Before actual metadata migration, set the status of the migrating subtree as freezing, upgrade the version number of the MLT and then push the updated MLT to all the rest of MDSs. Afterwards, the MDS will wait for a while and trigger the migration request because some corresponding metadata's requests may be still underway or have already bypassed the synchronization point. The metadata migration should keep waiting until the write operation of the metadata is finished to guarantee the integrity of the metadata operation. Similarly, if there is interaction between the client and MDS, the client will

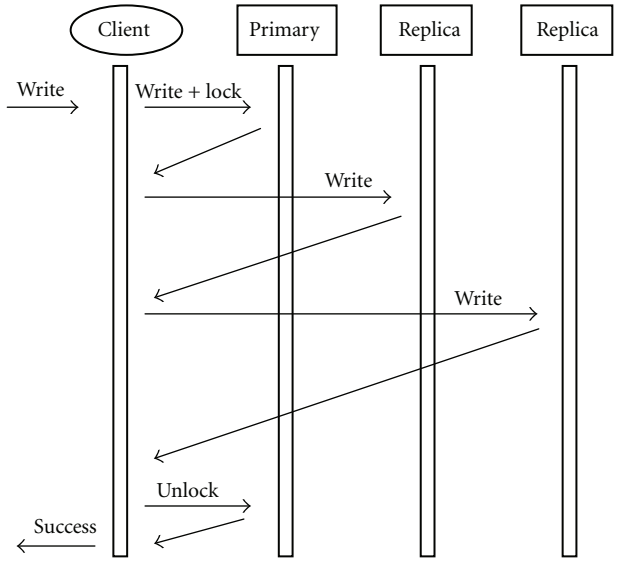

Figure 3: Metadata replica maintenance.

obtain the newest MLT in the cluster. If the current subtree is freezing, the process that sends the request will be suspended in limited time. If the migration is done before timeout, then the client matches the new MLT and sends the request to the correct MDS. Otherwise the client will return the device busying error (-EBUSY) to the upper layer and the error handling policy is determined by upper layer.

How the client obtains the MLT? If the client's metadata operations are all targeting to the freezing subtree, the current client cannot interact with the MDS cluster. Such completion of migration cannot be perceived rapidly. The only communication in the system is the sequence operation in every 90 seconds. Obviously it cannot be afforded by upper layer. As Figure 4 the system employs delay job queue to send the request of asynchronous getting MLT to track the migration change dynamically. Firstly, the timeout time, for example 20 seconds, is calculated. To track the change quickly, the delay jobs are uniformly inserted into the interval between the current time and the max timeout point to grab the change of the MDS cluster rapidly.

The migration program will be triggered after waiting for a certain period of time and the write operation of the freezing subtree will be denied during migrating. Since the migration is completed, the MLT is updated and pushed to all the MDSs including the new MDS. So far the new MDS has been added into the cluster and offers metadata service officially.

Because the scale of the MDS cluster won't be too large (possibly dozens of MDSs), the complex decentralized design is not adopted and the cluster member management has not used the gossip mechanism in dynamo. Instead, there is a central management node (primary MDS) which monitors the cluster members' activities.

All the nonprimary MDSs need to send the heartbeat messages to the primary MDS periodically to refresh its period of alive lease. If the primary MDS does not receive the heartbeat message of the corresponding MDS within the lease time, the status of the current MDS is set to be disabled by the primary MDS. Then the MLT version number is upgraded, and the new MLT is pushed to all the other MDSs. The current MLT is removed from the MDS cluster and the metadata service is stopped.

It is inevitable to handle the center node's failure in the centralized design. New primary MDS must be elected after the previous MDS is down. The most popular leader election algorithm is based on paxos protocol. In order to handle single node failure, we employ the zookeeper distributed consistency cluster to implement the leader election algorithm such that the single node failure and consistency can be handled by the zookeeper.

As shown in Figure 5, all the nodes in the MDS cluster register to the zookeeper (e.g., create the EPHEMERAL SEQUENTIAL node under the zookeeper directory node). Meanwhile, a watch is added to the parent directory node of the creating node so that all the registered nodes could receive the callback message. As the property of the created node is EPHEMERAL, this node is deleted automatically when the corresponding MDS is down. The MDSs which listen to the change of parent directory are able to receive the callback message sent from the zookeeper cluster. Meanwhile, owing to the SEQUENTIAL property of the registered node, the MDS judges whether the node with minimum ID among the rest nodes is created by itself. If so, the current MDS is the new leader elected by the cluster and waits for the connection from the other MDSs. Otherwise the current MDS is a follower and builds connection with the new leader.

5. Performance Evaluation

In this part, we test tree different file operations to evaluate the performance including file creation, file deletion, and file stat.

We conduct a comparison test on the systems including the original pNFS without strategy, the pNFS with the static subtree partitioning strategy, the pNFS with half-dynamic subtree partitioning strategy, and the Ceph with dynamic subtree partitioning strategy.

The testing environment contains one client, four MDSs, and one DS.

All the machines have the same configuration as follows:

CPU: Intel(R) Xeon(R) CPU E5620@ 2.40 GHz,

Memory: 2 GB × 6@ 1066 MHz (Multi-bit ECC, DIMM),

NIC: Intel 82576 Gigabit Network Connection (100 Mb/s),

HDD: Seagate 1T/7200 rpm/32 M,

OS: SUSE Linux Enterprise Server 11 sp1(x86_64),

Kernel version: 2.6.36.

Figure 6 illustrates the network topology of the testing environment. All the six machines are connected to the 100 M switch.

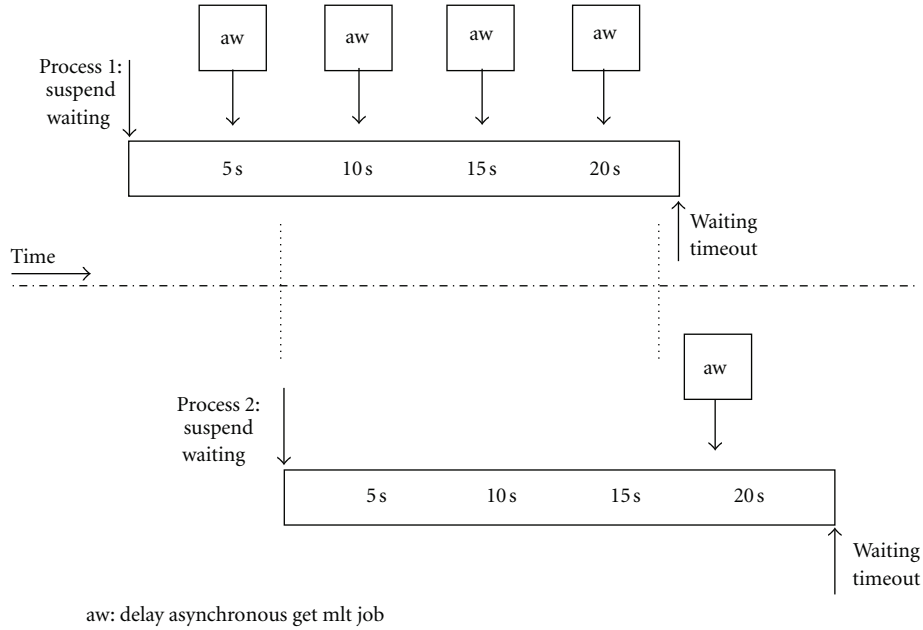

Figure 4: Delay asynchronous mechanism.

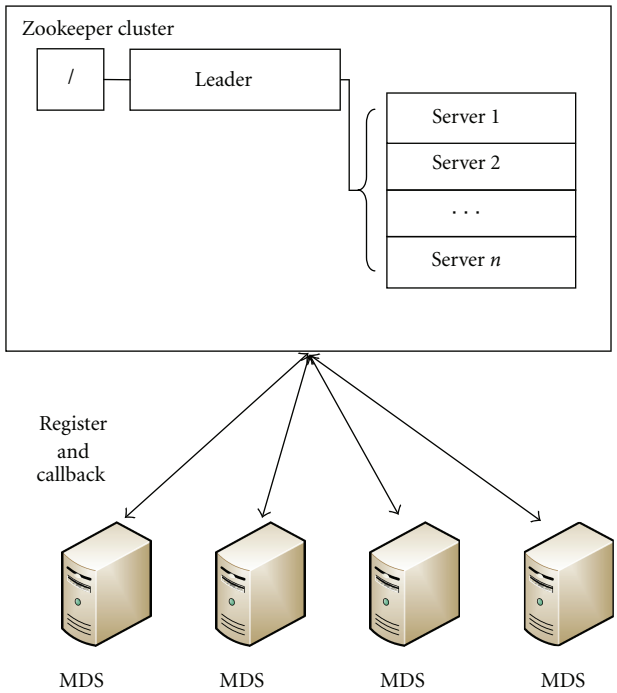

Figure 5: Leader election.

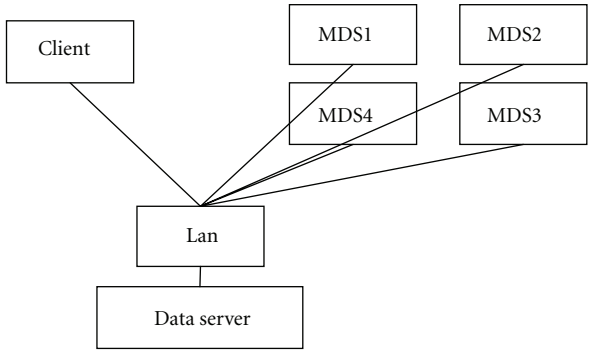

Figure 6: Testing network topology.

5.1. Static Subtree Partitioning Strategy and pNFS. pNFS is a protocol extension of NFS4.1, pNFS moves the metadata server out of the data transfer path, boosting performance by allowing multiple disk drives to serve up data in parallel and setting a dedicated metadata server to provide metadata service. The comparison system is a system which we have implemented based on the pNFS with subtree partitioning strategy.

In the test, there are 1000 files for one process and each condition is repeated 5 times to take the average value. Meanwhile, all the machines use "echo 3>/proc/sys/vm/drop_caches" to clear the cache of the inode, dentry, and page before another test.

5.1.1. File Creation. In pNFS as shown in Figures 7 and 8 processes have reached the peak at about 150 requests/sec. In static subtree partition strategy condition, when the number of the MDSs is two, the performance increases 40% and four MDSs increase 60%.

5.1.2. File Deletion. In pNFS condition, file deletion (as shown in Figure 8) reaches its peak at 700 requests/sec with 8 processes. Two MDSs achieve about 1300 requests/sec with 85% improvement. And four MDSs hit about 2400

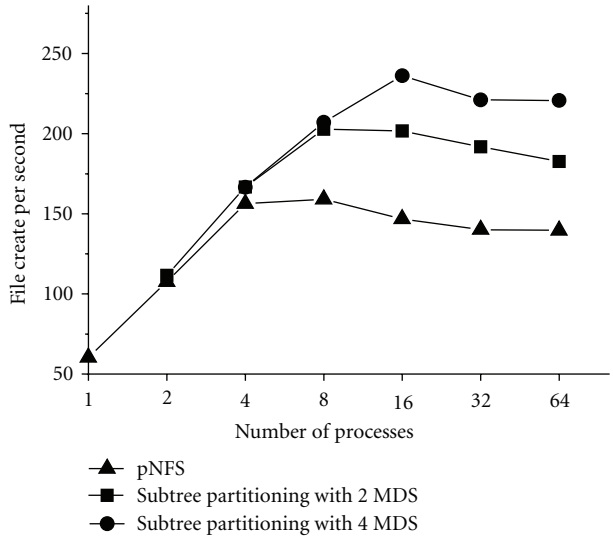

Figure 7: File creation.

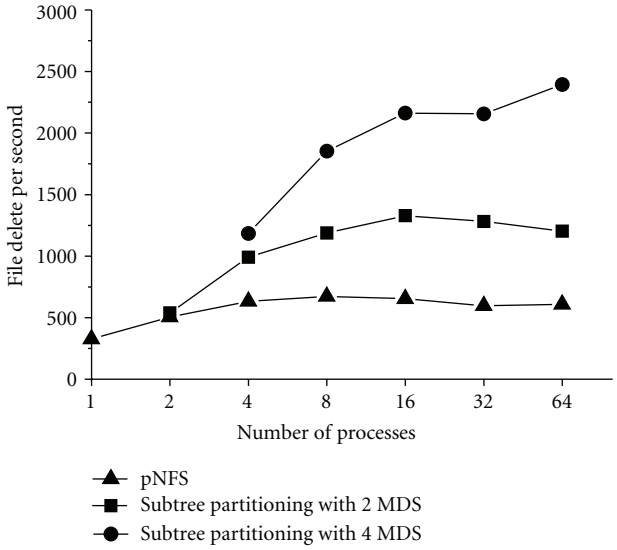

Figure 8: File deletion.

requests/sec with 240% improvement. It can be seen the good scalability of file deletion. But as the result of sharing a single DS with all the MDSs in such configuration, DS becomes the bottleneck. However, the number of DSs is far more than MDSs in real environment.

5.1.3. File Stat. In pNFS condition as shown in Figure 9, the performance of file stat operation reaches the peak at about 25000 requests/sec. And in multiple MDSs condition which uses static subtree partition strategy, the performance of both 2 MDSs and 4 MDSs is almost the same at 27000 requests/sec. There is no significant improvement of the file stat performance in single client multiple MDSs scenario. However, the test has not achieved the anticipated effect that the reading performance of multi MDSs increases linearly. After analysis, we find that the NIC of the client has been saturated and becomes the bottleneck of the system. Afterwards, we conduct equivalent testing between 2 clients and 3 clients when then MDS number is 4. The clients test result shows that the file stat performance can increase almost linearly (as shown in Figure 10).

5.2. Static Subtree Partitioning Strategy with Replicas. This comparison test is between the pNFS with subtree partitioning strategy and the pNFS with the half-dynamic subtree partitioning strategy mainly.

In this test, there are 1000 files for one process and each condition is repeated 5 times to take the average value. Meanwhile, all the machines use "echo 3>/proc/sys/vm/drop_caches" to clear the cache of the inode, dentry, and page before another test.

5.2.1. File Creation. In 4 MDSs and 3 replicas of this situation (as shown in Figure 11), the performance of file creation hits the peak at about 160 requests/sec which is equivalent to the single MDS single replica situation (i.e., the original pNFS situation). The result reveals that it is inevitable to lose the performance when improving the availability of the system. But as the reason of our system has good scalability, we can add more MDSs to increase the overall performance of the system. In the write-intensive application environment, the NWR replica mechanism which is similar to the Amazon dynamo can be employed. For instance, the performance of reading and writing can keep balance in the way of writing 2 and reading 2 to achieve the consistency. However, our system has not utilized NWR mechanism yet.

5.2.2. File Deletion. As shown in Figure 12 the file deletion performance is relatively poor and basically flat with the single MDS single replica condition, this is because the delete operation should delete all the tree copy of the metadata. But as the reason of our system has good scalability, we can also add more MDSs to increase the overall deletion performance of the system.

5.2.3. File Stat. As our system employs the strategy of writing 3 and reading 1, so the read operation is almost the same as the condition of no replications. As a result, all the file stat performances in the situation with replications are basically the same as those with no replications. The file stat performance can almost increase linearly.

5.3. Dynamic Subtree Partitioning Strategy of Ceph. Ceph is the only file system which uses the dynamic subtree partitioning strategy to realize the MDS cluster. All the MDSs provide metadata service in the form of cooperative caching. Each metadata server is responsible for the metadata service of some subtree branches, and load balance manager migrates subtree branches among the MDSs to realize the balance of system. All the metadata is durable stored in the OSD (the DS of ceph), while the metadata in the MDSs are just caching.

The following is the test results of ceph (version 2.0) which is tested under the same condition of the tests above.

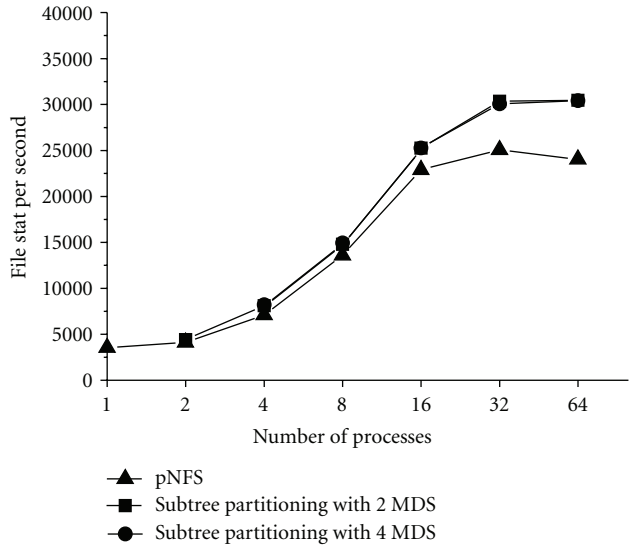

Figure 9: File stat.

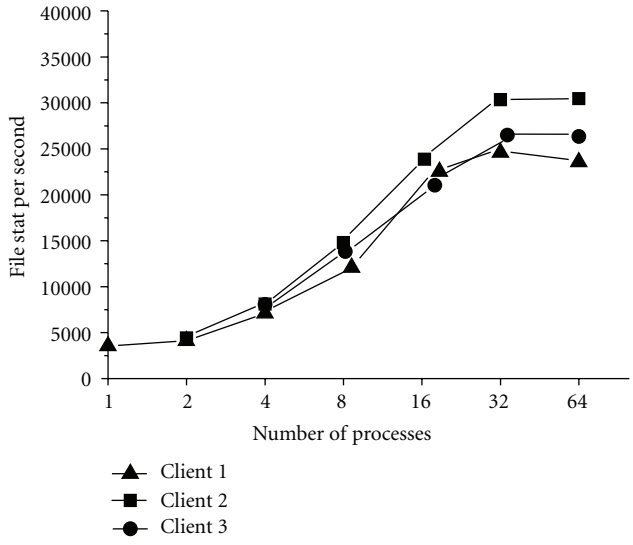

Figure 10: Tree client simultaneously file state.

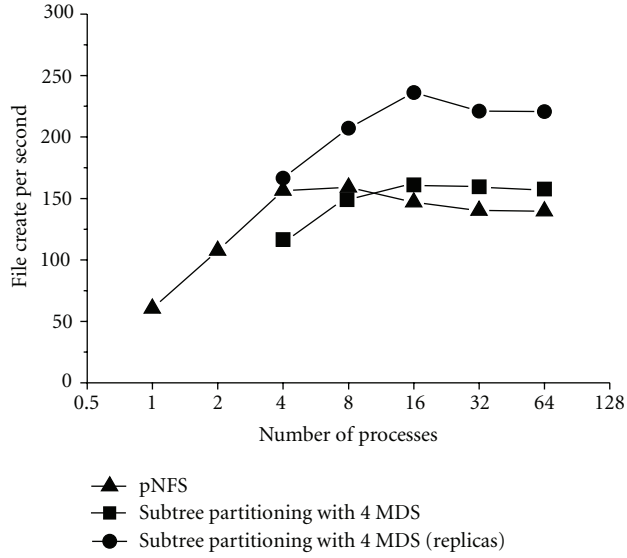

Figure 11: File creation.

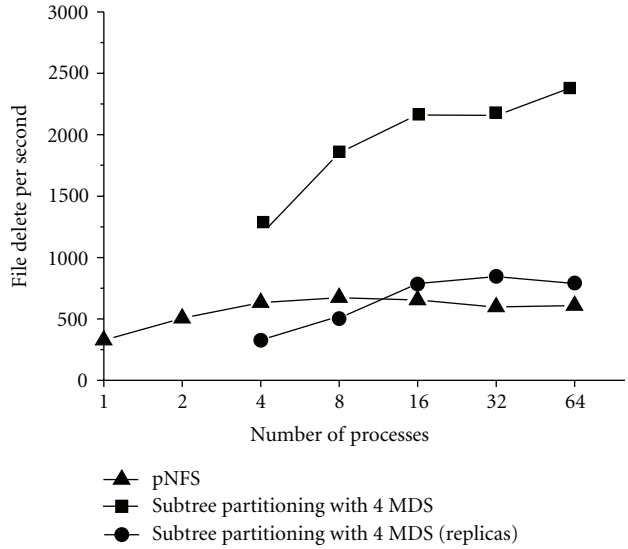

Figure 12: File deletion.

From Figures 13, 14, and 15 we can see that the performance is almost steady, while the MDS is increasing from 1 to 4. This is largely due to the design of the ceph that the MDS just caches metadata but is not responsible for the durable storage of the metadata. The creation operation and deletion operation will add the disk load of OSD, and the only OSD of our test environment would affect the performance of the system. By putting all the metadata in memory, the multi MDS of ceph improves the read performance, as well as the scalability of system. But in order to increase the write performance of the system, the storing of metadata should distribute on all the OSDs.

In contrast of all the tests above, we can see that our system is more simple and also has good scalability and performance. If the pNFS IOPS is S, our system's IOPS can reach about n*S*50% (n is the number of MDS). The reason is that our system directly stores the metadata durably in the MDS. And the Disk bandwidth of all the MDSs let the write performance increase by the adding of MDS. The independent catching of every MDS also increases the overall read performance. But our system also have its disadvantage that the cost of migration is more higher than the ceph when the load of the system is not balance. The reason is that we not only should migrate the metadata into the memory of MDS but also should store it into the disk in advance. The other thing should point out is that also our system has a good performance with the adding of MDS, but the overall performance is just below ceph. This is due to the reason that the current open source pNFS has not been ready for production environment and it has not consider much about the performance. And In short, the key point of our design

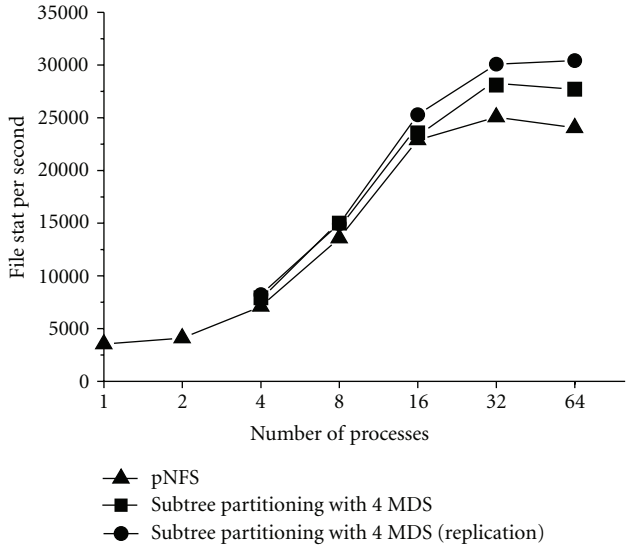

Figure 13: File stat.

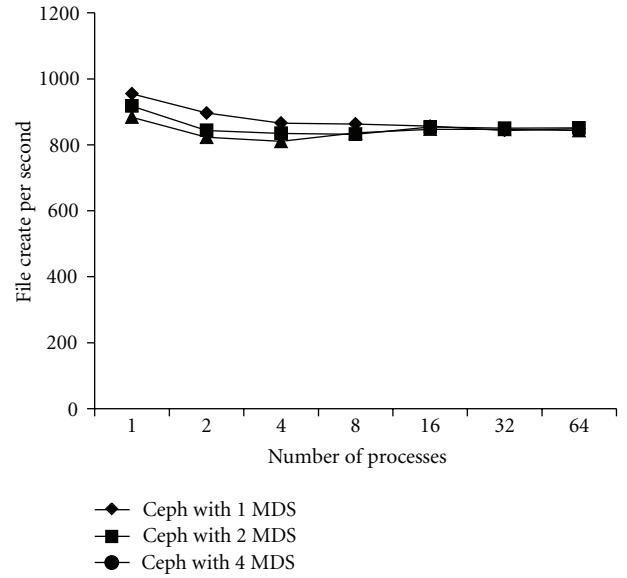

Figure 14: File create.

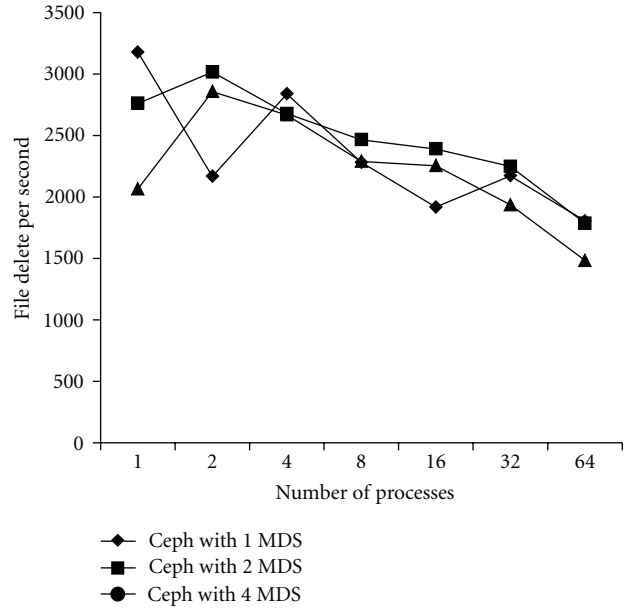

Figure 15: File delete.

is to improve the system's scalability and availability. The test result is desirable.

Compared to system with the traditional static subtree partitioning strategy, our system has combined the metadata replication to increase the availability of the system and also can decrease the unbalance. In contrast to system with the novel dynamic subtree partitioning strategy, the novel system (ceph) has a better ability of scalability, but the realization of this strategy is very difficult, and it is still researched in lab now mainly. Advantage of ours is being simple and mature.

6. Conclusion

The paper presents a half-dynamic subtree strategy in MDS cluster based on the open source pNFS distributed file system to improve the performance, scalability, and availability of the pNFS metadata service.

The experiment indicates that our MDS cluster can improve the read and write performance significantly than the pNFS and provide high scalability and availability at the same time.

We came across a lot of troubles during the study, for example, as the open source pNFS file system utilizes the empty file to store metadata and the inode number is used to file locating and accessing, the metadata inode number will be changed during metadata migration. Meanwhile, the inode number of the same metadata replica is different, leading to the problem that the load balancing of MDSs cannot be processed naturally. The open source pNFS file system puts emphasis on the implementation of the NFS4.1 protocol, regardless of the other issues such as the performance and multiple MDSs. Future work will be performance optimization based on the NFS v4.1 protocol and the redesign of MDS cluster and so on.

Acknowledgments

This work is supported by the National Basic Research 973 Program of China under Grant by National University's Special Research Fee (C2009 m052, 2011QN031, 2012QN099), 863 Project 2009AA01A401, Changjiang innovative group of Education of China no. IRT0725, 973 Project 2011CB302300.

References

[1] J. F. Gantz, C. Chute, A. Manfrediz et al., "The diverse and exploding digital universe: an updated forecast of worldwide information growth through 2011," IDC white paper, sponsored by EMC, pp. 1–25, , 2008.

[2] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller, "Dynamic metadata management for petabyte-scale file systems," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '04)*, pp. 4–15, November 2004.

[3] S. A. Brandt, E. L. Miller, D. D. E. Long, and L. Xue, "Efficient metadata management in large distributed storage systems," in *Proceedings of the 20th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS '03)*, pp. 290–298, 2003.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pp. 29–43, ACM, New York, NY, USA, 2003.

[5] P. J. Braam, The Lustre Storage Architecture. Cluster File Systems, Inc. Whiter Paper, 2003, http://www.clusterfs.com/.

[6] A. Adamson, D. Hildebrand, P. Honeyman, S. McKee, and J. Zhang, "Extending NFSv4 for petascale data management," in *Workshop on Next-Generation Distributed Data Managemen*, Paris, Farnce, June 2006.

[7] S. A. Weil, S. A. Brandt, E. L. Miller et al., "Ceph: a scalable, high-performance distributed file system," in *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06)*, pp. 307–320, USENIX Association, November 2006.

[8] W. Li, W. Xue, J. Shu et al., "Dynamic hashing: adaptive metadata management for petabyte-scale file systems," in *Proceedings of the 23rd IEEE/14th NASA Goddard Conference on Mass Storage System and Technologies*, pp. 93–98, May 2006.

[9] D. Pease, J. Menon, B. Rees, L. Duyanovich, and B. Hillsber, "IBM Storage Tank—a heterogeneous scalable SAN file system," *IBM Systems Journal*, vol. 42, no. 2, pp. 250–267, 2003.

[10] B. Welth, M. Unangst, Z. Abbasi et al., "Scalable performance of the panasas parallel file system," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08)*, pp. 17–33, 2008.