Design and Implementation of a Non-Shared Metadata Server Cluster for Large Distributed File Systems

Jong-Hyeon Yun, Yong-Hun Park, Seok-Jae Lee, Su-Min Jang, Jae-Soo Yoo
Dept. of Computer&Communication Engineering Chungbuk National University
{jhyun, yhpark, sjlee, smjang}@netdb.cbnu.ac.kr, yjs@cbnu.ac.kr

Abstract

Most of large-scale distributed file systems decouple metadata operation from reading or writing operation for a file. In other words, certain servers named metadata server (MDS) is responsible for maintaining the metadata information of file systems. But, mostly the existing file systems are used a restrictive metadata management technique, because those systems mostly designed to focus a distributed management of data and an input/output performance rather than the metadata.

In this paper, we present a new non-shared metadata management technique in order to provide flexible metadata throughput and scalability. First, we introduce a new metadata distribution technique called dictionary partitions (DP). Then, we introduce load distribution technique based DP. In addition, we demonstrate the superiority of our technique with shared metadata management technique.

1. Introduction

Recently, the various researches for large-scale distributed file system and related works are actively underway. Mostly the existing distributed file systems have a feature that is decoupling metadata operation from reading or writing operations for file. In the distributed file systems, certain servers named metadata server (MDS) maintains metadata information in file system such as file system namespace. Other servers, named storage server, are responsible for maintaining contents of a file. In such a system, a client will get the metadata information from MDSs to access to file contents. Then, the client verifies permission to access to the files and gets information specifying the location of storage servers having the file contents.

Although the size of metadata is relatively small compared to the overall size of the system, metadata operations may make up over 50% of all file system operations[1] because it is necessary to access the metadata of MDS before reading or writing file in system. In addition, the requests which does not need to access to the file contents are exist, such as traverse of directories, change of permission or owner of files, etc. Therefore, in such a system, the metadata operation is relatively frequent compared to the file operation, so that the performance of the MDS is critical for the entire system.

In this paper, we describe a new metadata management method for a cluster of MDSs to efficiently distribute metadata in large-scale file system. We propose a dictionary partitioning (DP) method to adapt dynamically changing the workload, as well as continually preserving and exploiting the locality of file system namespace for providing better search of the metadata. Finally, we demonstrate the superiority of our technique with shared metadata management technique.

The rest of this paper organized as follows. Section 2 describes the related works. Section 3 proposes the DP method and section 4 describes the load balancing in the method. Section 5 shows the performance evaluation. Finally, section 6 describes the conclusion of the paper.

2. Related works

As the appearance of large-scale file system that separates metadata operation from file read or write operations, metadata management has become an interesting area of research. When designing a metadata server, it is necessary to design method of partitioning metadata among MDSs for maintaining the performance of MDS efficiently and the workload among MDSs uniformly.

Usually, networked file systems have partitioned metadata in file system by statically assigning portions of the directory hierarchy to different file server. It is called the "static subtree partitioning"[2][3]. *Static subtree partitioning* is easy to maintain the metadata

978-0-7695-3428-2/08 $25.00 © 2008 IEEE
DOI 10.1109/CSA.2008.52

but its performance is influenced by initially partitioning the metadata.

In order to provide good load balancing among MDSs, the several system make use of *the hash* to distribute metadata across MDSs uniformly[2][3]. The unique file identifier, such as the inode number or path name, is used as the input value of hash function. *The hash* provides good load balancing across MDSs. However, the hash needs the additional traversal of the metadata scattered on multiple MDS to access to a file or directory because it does not consider the locality of the file or directory.

In addition, dynamic subtree partitioning has been proposed for improving on statically subtree partitioning[3]. However dynamic subtree partitioning is more difficult to maintain a good partition because it is not easy to adapt to changing workload dynamically and growth of the file system.

3. Dictionary Partitioning Method (DP)

To partition metadata of file system namespace, it needs to sort the metadata in numeral and alphabetical order like a dictionary. The metadata, which has hierarchical structure, is presented as a linear structure. Figure 1 shows the transformation of the directories formed a hierarchical structure into a linear structure.

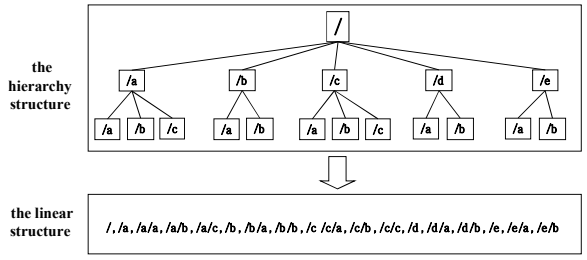


Figure 1. The transformation the hierarchical directory structure to the linear directory structure

The sorted metadata is divided into the number of MDSs in consideration of amount of the metadata and it makes the number of portions of the metadata. The portions are assigned to the MDSs one by one. Naturally, the MDSs has the sequence according to the sequence of the portion assigned them respectively. Figure 2 shows the example of the division of metadata into three and the assignment of each portion of the metadata to a MDS one by one.

Division-position is a set of file paths produced by partitioning metadata. They present the edges between the portions of the metadata. The division-position includes a version to distinguish the latest division-position. If a division-position is modified, the version is updated incrementally. Every MDS maintain the same division-position. The maintenance of division-position hardly affect to the performance of MDSs because the size of it is very small and it is never updated before the edges of the portions of metadata are moved.

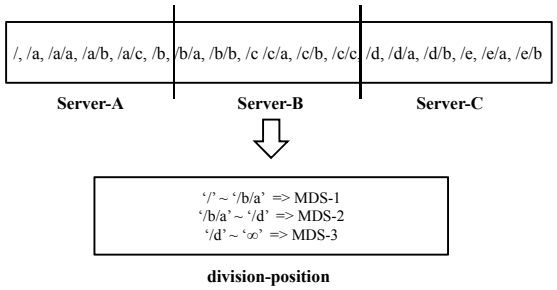


Figure 2. A example of the dictionary partitioning method(DP)

When a client accesses to a MDS, if the client does not have a division-position, the client receives division-position from the MDS. Since then the client can directly access to the MDS which have the metadata the client requests while other existing metadata management methods need to access to MDSs several times for it. If a client already has had the division-position with the same version as the MDS, the client does not receive them. However, if a client has different version from MDS's later, the client receives them again.

When a search operation occurs, a client chooses a MDS to request the metadata of files or directories though the division-position. And then the client sent the request to the MDS. If the MDS does not have the metadata requested from the client because of a change of the division-position, the comparison of the version of it between the MDS and the client are performed. If they have different version each other, the client updates the own division-position and retry the search operation.

4. Load balancing

DP divides the metadata into the number of MDSs and assigns the divided metadata to MDSs. Therefore, each MDS is responsible for maintaining the assigned metadata. Every client that requests the metadata needs to access to the MDS having the requested metadata because the MDSs do not share the metadata.

In the applications using a large scale file system, the access to several files may occur frequently such as interesting UCC contents, hot news in web sites, etc. In the case that the frequent accesses to a certain contents occur, the metadata of the files related to the contents is also frequently accessed. When the operation related

to the metadata of certain MDSs extraordinarily increase, the throughput of the entire file system is highly decreases because of the limit capability of a MDS although the system has enough capability to process them. Therefore, in order to keep each MDS workload equally, this problem must be solved.

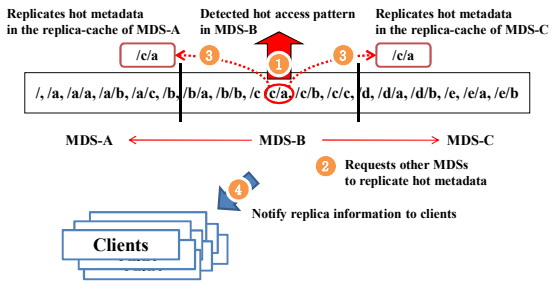


Figure 3. The steps of the load balancing using a replica-cache

In the proposed method DP, a replica-cache is used to solve the problem. The replica-cache maintains the frequently requested metadata at all MDSs redundantly. Figure 3 shows how to use the replica-cache. First, each MDS manages the request of client at a queue to analyze the access pattern. The request records in the queue are analyzed periodically. If hot access pattern are detected by the analysis, the MDS which is responsible for the hot metadata requests other MDSs to replicate them. After the replication, if clients request the hot metadata, the MDS accessed by the clients notify them that the metadata are replicated to all MDSs. The MDS which request the replication notify all other MDSs the metadata in the replica-cache are invalid anymore when the workload of the metadata does not have hot access pattern.

Initially, the metadata are assigned to each MDS equally. However, the amount of the metadata which each MDS has increases and decreases over time. Specially, according as a single directory becomes extraordinarily large or movement of directories occurs, it makes the amount of the metadata of MDSs unbalanced. If this state is continued long time, it may be undesirable of inefficient for it to reside on a single. Therefore, it needs the methods to keep the amount of metadata equally at each MDS.

If the unbalance occurs between MDSs, the adjustment of the division-positions is needed in the file system. The adjustment makes the amount of the metadata and the request balanced over MDSs. However the adjustment requires large computation and communication costs such as scanning whole metadata, analyzing the access pattern, deciding the new division-positions, etc. Consequently the administrator of the system must establish the appropriate threshold to decide whether or not the adjustment performs.

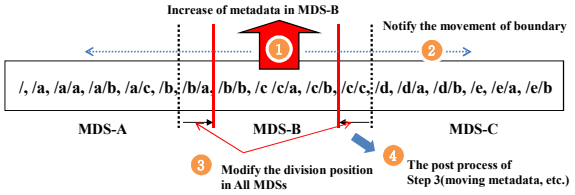


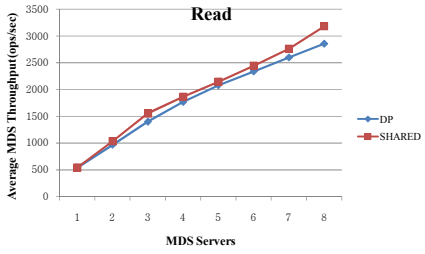
Figure 4. The steps of the adjustment of the division-positions

Figure 4 shows the procedure of the adjustment for the new division-positions when the amount of the metadata of a MDS extraordinarily increases. First, MDS B computes the difference of metadata statistics, which include the amount of metadata and access pattern, from the adjacent MDSs. Next, MDS B computes the new division-positions and sends them to all other MDSs. In addition, parts of the metadata can be sent to the adjacent MDSs according to the new division-positions.

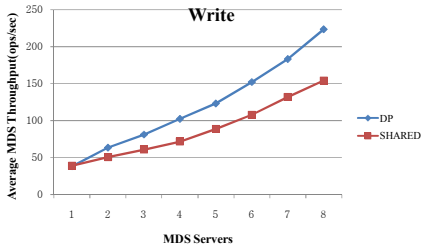
5. Evaluation

Our experiments were performed on *IBM xserise 3600* with *2G RAM* running *Redhat Linux Fedora Core 6* as MDSs and Clients. *BekeleyDB 4.6* is used to manage the metadata in each MDS. The programs to control the database and the file operations are implemented by C language. The TCP protocol is used to connect all MDSs each other. We evaluate *DP* with the shared based strategy (SHARED), which the entire metadata is replicated to all MDSs. In the case of SHARED, the clients can process metadata operations by accessing to any MDS because every MDS has the same metadata.

Figure 5 show the affection of the total throughput according to the number of MDSs. A thousand of clients execute total 100K times the read and write operation. The directory hierarchy is maximum 5 and the files are uniformly distributed to all the directories in the file system. The files and directories in the namespace are randomly chosen for the operations. Both the strategies show a similar throughput for the read operations, but DP relatively improves 10~40% throughput of the write operation as compared with SHARED. That is because SHARED need additional cost for the write operation to apply it to all other MDSs, unlike DP.



(a)



(b)

Figure 5. Performance comparison with varying the number of MDSs. (a) read (b) write

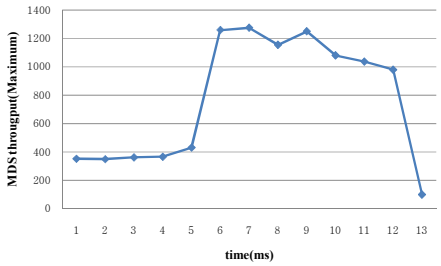


Figure 6. MDS Throughput with the replication policy

When the operations related to the metadata of a certain MDS extraordinarily increase, the throughput of the file system is highly decreased even though the system has enough capability to process them. DP immediately replicates the metadata in all other MDSs and all MDSs collaborate with the operations related to the metadata. So that, the operations skewed to a MDS are distributed to all MDSs equally through the replication policy.

Figure 6 shows the affection of the throughput according to use of the replication policy. The 8 MDSs are used and clients request 10,000 read-operations to a MDS in the simulation.

In the result of the simulation as shown in Figure 9, after processing the 2000 requests, the throughput of the request is improved. During processing the 2000 requests, the replication is not operated and the average throughput per 1ms is about 400 requests. After that, DP detects the hot access pattern and the replication is operated with the replication policy. Consequently, the throughput is immediately increased and it is about 4 times better than before.

6. Conclusion

In this paper, we designed and implemented the DP for large scale metadata. DP sorts the entire metadata, computed the division-positions dividing them into the number of MDSs. The division-positions are moved according to the change of the number of metadata and the amount of workload. DP uses the replication policy to avoid decreasing the throughput of the system when flash-crowd occurs. We compared the performance of DP to SHARED and proved the superiority of DP. In future, we will research and develop the large scale distributed file system adopting DP.

Acknowledgements

This research was supported by Energy of the Korean Government and the Korea Research Foundation Grant funded by the Korean Government (MOEHRD)(The Regional Research Universities Program/Chungbuk BIT Research-Oriented University Consortium) as well, the Program for the Training of Graduate Students in Regional Innovation which was conducted by the Ministry of Commerce Industry.

Reference

[1] D. Roselli, J. Lorch, and T. Anderson. “A comparison of file system workloads”, *In Proceedings of the 2000 USENIX Annual Technical Conference*, San Diego, CA, USENIX Association, June 2000, pages 41–54.

[2] Sage Weil, Kristal Pollack, Scott A. Brandt, Ethan L. Miller, “Dynamic Metadata Management for Petabyte-Scale File Systems”, *In Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC '04)*, ACM, November 2004.

[3] S. A. Brandt, L. Xue, E. L. Miller, and D. D. E. Long, “Efficient metadata management in large distributed file systems”, *In Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2003, pages 290–29