A Dynamic Metadata Equipotent Subtree Partition Policy for Mass Storage System

Gongye Zhou[1, 2],Qiuju Lan[1], Jincai Chen[1, 2]
[1]*College of Computer Science and Technology, Huazhong University of Science and Technology*
[2]*Division of Data Storage System, Wuhan National Laboratory for Optoelectronics*
Wuhan 430074, P.R.China
E-mail: lansabrina@163.com

Abstract

Metadata plays an important role in mass storage system. How to distribute and balance the metadata of the metadata server cluster determines the overall performance of a cluster. Two popular metadata distribution policies are the dynamic subtree policy and hashing policy, while the dynamic subtree partition is vulnerable to the imbalance workload and hashing partition has a random distribution which will incur a burst of network overhead when updating metadata.

We present a novel approach for metadata management. It combines hash and subtree partitioning policies together to partition directory hierarchy tree into equipotent subtrees with a certain granularity and employs value of hashing subtree to distribute subtrees across the metadata servers. It also employs a balance strategy to adjust the metadata distribution dynamically. After adjustment, we present a hot spots elimination strategy to detect and reclaim hot spots in the file system efficiently. We also demonstrate a design using this strategy to achieve more efficient performance than the other policies using hashing partitioning and subtree partitioning purely.

1. Introduction

Metadata is critical in the large distributed object-based storage systems (OBS)[1]. Traditional metadata server (MDS) cluster suffers from frequent metadata access and metadata movement within the cluster [2]. Although the size of metadata is relatively small compared with the overall capacity of the storage system, according to research, 50% of all file system operations are metadata operations[3].So carefully designed metadata management strategy is needed .

At present, the prevailing system architecture for mass storage systems is Object-based Storage architecture [4,5].To handle the workload requested by tens of thousands of clients, metadata should be set properly to take full advantage of the MDS cluster [6]. Nevertheless, the workload can dynamically change in every MDS. Furthermore, thousands of clients may access a same file simultaneously or over a short period of time [6,7]. The access to the metadata of this file will become the bottleneck and the MDS which storage the corresponding metadata will be overhead [7]. And the bottleneck metadata is called the hot-spot. The hot-spots will result the system in long response and even some client request will be dropped. So a well designed metadata partition strategy and balance strategy and a hot-spots elimination strategy is needed.

2. Related Work

Nowadays, the metadata distribution policies can be concluded into three groups: static subtree partitioning policies and dynamic subtree partitioning policies [8,9]and hashing distribution policies [10,11].

Static subtree partitioning policies partition the namespace into subtrees and all sub files and subdirectories of a directory is managed by one metadata server [9]. Both the namespace partitioning and the subtree assignment are performed manually beforehand. The major disadvantage is that the workload may not be evenly distributed among metadata servers.

Like static subtree partitioning policies, dynamic subtree partitioning assigns subtree of the global namespace to metadata servers. The smallest unit of metadata transfer is directory (subtree).DCFS [12]

which employs this policy will automatically choose a metadata server when creating an item under the root directory. While creating an item under other directory instead of the root directory, it will store the item in the metadata server managing the item's parent directory. However, this policy can't keep the relative balance of metadata distribution dynamically.

Hashing partitioning policies uses random policy to hash pathname or inode number. It's the simplest policy to distribute metadata evenly while eliminating all hierarchical relationship of the file systems. Due to this reason, to rename or modify a directory, there will be severe performance decline. And Brandt etc [10] presented a policy called LazyHybrid to address these problems.

3. Dynamic Metadata equipotent subtree partitioning strategy

The rest of the paper is organized as follows. First, there's introduction about the system architecture in 3.1. In section 3.2, we will discuss how to distribute metadata subtree with a certain granularity. In section 3.3, we will discuss the method of mapping between metadata and metadata servers. How to adjust the workload of metadata servers will be discussed in section 3.4.Fianlly, we give the strategy to eliminate the metadata hotspots.

3.1. System architecture

The system architecture we use here is object-based storage architecture. It composed of a number of client nodes and shared IP-SAN storage devices and a cluster of metadata servers as figure 1. The storage devices are made of a number of object-based storage devices which separate metadata management from file read/write operations.

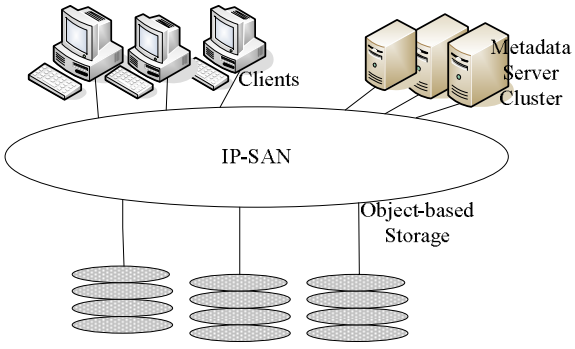

Figure 1. The Object-based Storage System

3.2Dynamic subtree partitioning

Hashing partitioning policies can be looked as distribute a subtree with one node among the metadata servers while the subtree partitioning policies as the distribution level is 1 whose directories under the first level are all mounted to one metadata server. It is too large to keep the balance of metadata distribution. In order to meet the balance of metadata of distribution and keep the most of hierarchy of directory, we present a medium-granularity subtree partitioning policy.

The subtree we used is a part of directory hierarch which is specified with certain depth and leaf nodes. The metadata is partitioned according to the defined granularity.NodeNum stands for the maximum child nodes of a node and TreeDepth represents the maximum depth of a subtree. Assume the subtree's TreeDepth is 3 and NodeNum is 2, the following figure 2 illustrates the subtree.

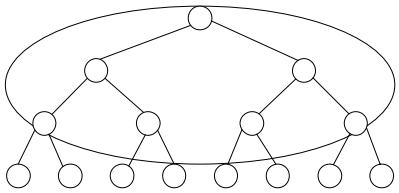

Figure 2. The subtree with TreeDepth =3, NodeNum =2

This policy uses the directory hierarchy to partition the whole tree into numbers of equipotent subtrees. Then it employs hashing policy to distribute these subtrees across the metadata servers.

```
If (X is a file)
      X is stored in the P.MDSi;
Else {
      If (P.Nodenum+1<=NodeNum) {
         If (P.Depth+1<=TreeDepth) {
           P.Nodenum++;
           X.MDSi=P.MDSi;
           X.Depth++;
         }
      }
      Else {
         MDSi=hash (X's directory);
         X.Depth=1;
         X.Nodenum=0;
         X.MDSi=MDSi;
      }
 }
```

Figure 3.The algorithm of distribution metadata subtree

First, we have to define the const values of NodeNum and TreeDepth. We also create a structure to store the information of tree node.

Depth: the depth of the node.

Nodenum: the number of node.

MDSi: the metadata server No that stores the node.

Supposing a subtree is G, we create an item X in the directory P. If X is file, its entry should be stored in the MDS where its parent directory is managed. If X is a directory and both node number and depth of its parent directory P are not reached to the maximum, P's Nodenum and Depth is added one and X's MDSi is P's MDSi. Else, we will create another subtree with parent directory is X. To accomplish this, figure 3 illustrates the algorithm of partitioning metadata.

3.3. Metadata Mapping

Metadata look-up table provides the clients with the mapping between subtree and MDS. When a file is accessed by a client, the metadata server first finds the subtree which stores the file. Then, using the hashing value of the subtree as an index to the MTL, and the corresponding metadata server id found in the entry indicates the metadata server. Finally, the client contacts the selected metadata server to obtain file's metadata, file-to-object metadata and security information.

In order to avoid the hashing value changing suddenly when updating the directory. We choose the directories of every subtree's root directory that hierarchy above the current subtree as the parameter of hashing function. Figure 4 indicates how to define the hashing parameter. The subtree1's root directory is /home1 and the subtree2's root directory is home2. We use the value of hashing /home1 to distribute subtree1 and hash /home1/home2 to distribute subtree2 while not considering the other hierarchy directories above the home2.So, when updating the directory in the subtree; it won't change the hashing values.

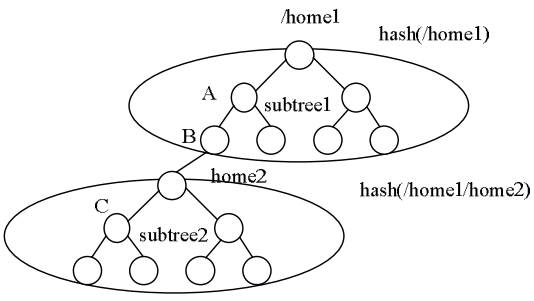

Figure 4. Hash policy of distributing two subtrees with TreeDepth =3, NodeNum =2

Every client and every metadata server maintain a copy of the MLT and a directory tree. When a MLT is updated in a server, the update message is broadcasted to every metadata servers.

3.4. Metadata Load Balancing

The subtree is distributed in the MDS cluster according the hash value of the subtree's the directory. However, the workload changes all the time. In order to adapt to the changing workload, we adopt a metadata load balancing strategy.

1. Calculate every metadata server access counter using the bellow equation
   $MDS_i = \sum_{subtree \in MDS_i} ACCESS_{counter}$
2. Calculate the following equations to get the average access counter of every metadata server.
   $x_i = \frac{w_i}{\sum w_i} \sum MDS_i$
3. For i =1 to n
   $p_i = x_i - MDS_i$
4. Busy=$\{ i \mid p_i < 0 \}$, Free=$\{ s \mid p_s > 0 \}$
5. For ($i \in$ Busy && i != NULL) {
   $p_i = -p_i$
   For ($s \in$ Free && s ! = NULL) {
   If ($| p_i - p_s | > 2\% * x_i$ && $p_i - p_s < 0$) {
   Find next s in Free;
   Break;
   }
   Else {
   Find a subset of subtrees in MDS_s .sum of $ACCESS_{counter}$ is approximate to p_s ;
   Move metadata of these subtrees from MDS_s to MDS_i ;
   If ($p_i - p_s | \leq 2\% * x_i$)
   Break;
   $p_i = p_i - p_s$;
   Move s away from Free;
   Find next s in Free;
   }
   }
   Move i away from Busy
   Find next i in Busy;
   }

Figure 5. The algorithm of adjusting the distribution across metadata servers

Supposing the update period is P, each MDS maintains the access counter $ACCESS_{counter}$ for every subtree. The absolute workload of every metadata MDS_i is the sum of $ACCESS_{counter}$ of its

subtrees. Each MDS is assigned a weight w_i according to the performance of the server (such as CPU frequency, memory capacity, I/O). The above Figure5 illustrates the algorithm of how to redistribute the metadata. First, we calculate every metadata server access counter MDS_i and average counter x_i in terms of w_i assigned. The adjustment is executed every period of T and whole algorithm's complexity is determined by the number of subtrees which is much fewer than single metadata. We implemented the algorithm in normal PC (Celeron 2.0G, 512M memory). It consumes no more than 2ms when subtree number is 1024 and 5 metadata servers.

3.5 Hotspots Elimination

Because the clients are unaware in system, some metadata may be accessed simultaneously or over quite a period of time, which will lead thousands of quests to being delivered to a single MDS. Thus, certain MDS will need a lot of time to response to the clients, some metadata servers even will drop the requests of the clients.

In order to detect the popular metadata, we use an access information structure to record the active metadata which stores in the cache of the MDS. The following table 1 illustrates the structure[13].

Table1. Access information structure

$Access_{total}$	$Number_{replica}$	T	$Access_{average}$

Where $Access_{total}$ stands for total access counter to the metadata. $Number_{replica}$ stores active replica number. T field stores the exact time when $Access_{total}$ is flushed to zero.

When metadata is accessed, it begins to check the corresponding table. If the interval time is bigger than the given interval T_p, $Access_{total}$ is updated to zero and the $Access_{average}$ will be set as follows.

$$Access_{average} = Access_{total} / Num_{replica}$$

The strategy algorithm is as follows. With increasement of the access frequency, if the average access frequency of the metadata exceeds the threshold of average access $P_{threshold}$, a new metadata should be replicated. Replicas are distributed randomly among MDS cluster except the one maintaining the metadata. When accessing the hot-spots,clients can choose metadata server randomly from the list which we maintains for the hot-spots metadata. If the $Access_{average}$ is smaller than the threshold of average access, one replica should be marked to be unavailable and dropped from the replicas list.

4. Evaluation

Performance tests in a fixed cluster with 5 metadata servers and public NFS file system. This file system has 4,564direcories and 26,500files.The granularity of the subtree with TreeDepth is 4 and NodeNum is 2.

Three policies including Subtree policy, LazyHybrid policy and policy we proposed are tested in the cluster system. The following figure 6 described the distribution of metadata with different distribution policy.

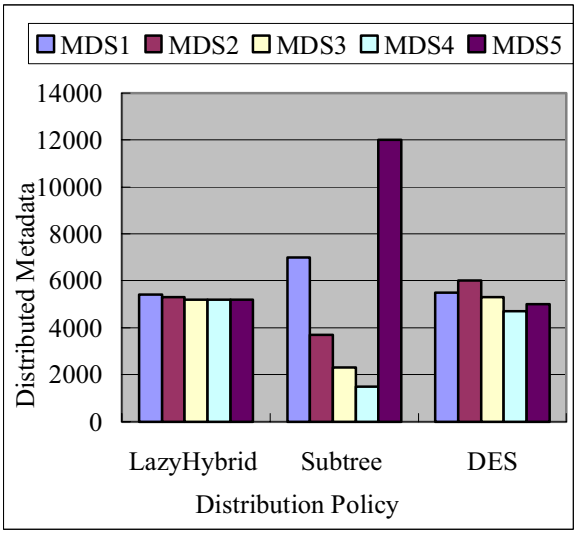

Figure 6. Metadata Balancing with Different Distribution Policy

DES represents the dynamic equipotent subtree policy. From the figure 6, LazyHybrid policy can balance the metadata workload evenly, while subtree can't adjust the imbalance of the metadata servers and there are obvious distribution differences among the metadata servers. The dynamic equipotent subtree policy which uses hashing to distribute subtree with equipotent granularity can balance the metadata distribution dynamically while not as evenly as the LazyHybrid. So, hashing policy can balance the workload evenly while subtree policy is vulnerable to the imbalance of metadata servers.

In the following test, we update a file or directory randomly with a rate between 1%~5%, the figure 6

describes the rate of metadata migration with different policy. It only needs to update the directory when using subtree policy. So, there's no metadata migration as the figure 7 shows. While using LazyHybrid, the rate of metadata migration increases sharply with the growth of the rate of directory update. Because updating the directory, lots of hashing value will change and lots of metadata have to redistribute. If using dynamic equipotent subtree policy, some subtrees will be migrated. If the directory updated is not the root directory of a subtree, no metadata need to be removed. So the migration metadata is much fewer than employing LazyHybrid policy.

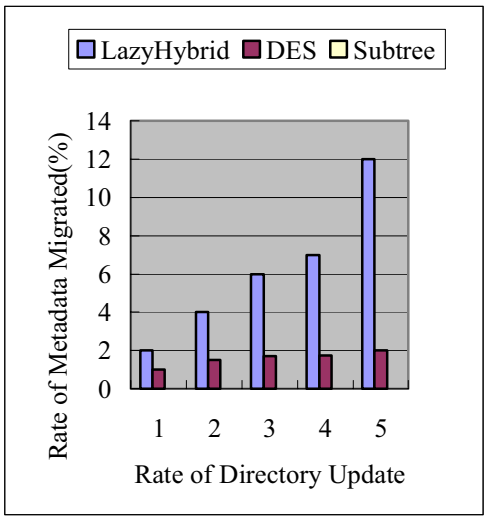

Figure 7. Rate of metadata migrated vs. Rate of directory update

The following figure 8 indicates the throughputs of operations per second while increasing the nodes of clients with different distribution policies. The throughputs of subtree policy are much lower than the other two. The throughput of LazyHybrid policy is lower than that of dynamic equipotent subtree policy and the gap between the two policies tends to widen when increasing the clients. These results indicates that the dynamic equipotent subtree policy is potentially more efficient that the other two policies.

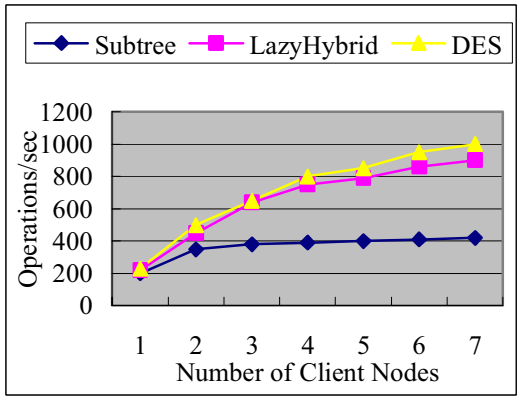

Figure 8. The operations /sec with the growth of number of client nodes

5. Conclusion

In a cluster file system that consisting of thousands of nodes and thousands of online users and running tasks, not only the data path to object-based storage devices should be addressable but also multiple metadata paths to a group of metadata servers. How to distribute, mapping and balance the metadata across the metadata servers is an important issue that determines the holistic metadata processing performance of all metadata servers.

This paper presents a novel policy named dynamic equipotent policy to combine subtree policy and hashing policy together. It partitions the directory tree to lots of medium-granularity subtrees and uses the value of hashing of subtree to distribute subtree across the metadata servers. When the workload changes dynamically, dynamic equipotent policy can adjust the metadata distribution to get high performance. After the adjustment, we present a strategy to manage the whole life cycle for all hot spots in the file system.

Our performance results show that this policy is potentially more efficient than the other policies using subtree partitioning or hashing partitioning purely. We will future work on how to define the granularity of subtree in clusters with different scales.

Acknowledgments

This work was supported by the National 973 Program of China (2004CB318201) and the Natural Science Foundation of Hubei Province of China (2005ABA257).

References

[1] Weijia Li,Wei Xue,Jiwu Shu and Weimin Zheng, "Dynamic Hashing: Adaptive Metadata Management for Petabyte-scale File Systems", In *Proceedings of the 23 st IEEE/14 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST2006).*

[2] Jie Yuan, Yao-Long Zhu, Hui Xiong, Renuga Kanagavelu, Feng Zhou and So LihWeon, "A Design of Metadata Server Cluster In Large Distributed Object-based Storage", In *Proceedings of the 23 st IEEE/14 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST2006).*

[3] D.Roselli, J.Lorch, and T.Anderson, "A Comparison of File System Workloads", *Proceedings of the 2000 USENIX Annual Technical Conference, page 41-54, Jun 2000.*

[4] R.O. Weber, *Information technology-SCSI object-based storage device command (OSD)*, Technical Council Proposal Document T10/1335-D, Technical Committee T10, Au 2002.

[5] Thomas M. Ruwart, "OSD: A Tutorial on Object Storage Devices", *Proceedings of the 19 th IEEE/10 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST2002).*

[6] S. A. Brandt, L.Xue, E. L. Miller and D. D. E. Long, "Efficient Metadata Management in Large Distributed File Systems", *Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies,* Apr 2003,pp. 290-298.

[7] Jin Xiong, Rongfeng Tang, Sining Wu, Dan Meng, Ninghui Sun, "An Efficient Metadata Distribution Policy for Cluster File Systems", In *Cluter Computing 2005.* Sept 2005, pp.1-10,

[8] P. F. Corbett and D. G. Feitelso, "The Vesta Parallel File System", *ACM Transactions on Computer Systems*, 1996,pp.14(3):225-264.

[9] S. A. Weil, K. T. Pollack, S. A. Brandt and E. L. Miller, "Dynamic Metadata Management for Petabyte-scale File Systems", *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC'04), Nov2004.*

[10] Scott A.Brandt, Ethan L.Miller, Darrell D.E.Long and Lan Xue, "Efficient Metadata Management in Large Distributed Storage Systems", In *Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2003).*April 2003,pp.290-298.

[11] E. Levy and A. Silberschatz. "Distributed File Systems: Concepts and Examples", *ACM Computing Surveys*,Dec 1990,pp.22(4).

[12] Jin Xiong, Sining Wu, Dan Meng, NinghuiSun, Guojie Li. "Design and Performance of the Dawning Cluster File File System",In *2003 IEEE International on Conference Cluster Computing (Cluster2003)*, Hong Kong,Dec.1-4,2003, pp.232-239.

[13] Qingsong Wei, Wujuan Lin and Yong Khai Leong. "Adaptive Replica Management for Large-scale Object-based Storage Devices",In *Proceedings of the 23 st IEEE/14 th NASA Goddard Conference on Mass Storage Systems and Technologies(MSST2006).*