A DESIGN OF METADATA SERVER CLUSTER IN LARGE DISTRIBUTED OBJECT-BASED STORAGE

Jie Yan, Yao-Long Zhu, Hui Xiong, Renuga Kanagavelu, Feng Zhou, So LihWeon
Data Storage Institute, DSI building, 5 Engineering Drive 1, Singapore 117608
{Yan_jie, Zhu_Yaolong}@dsi.a-star.edu.sg
tel +65-68748085

Abstract
In large distributed Object-based Storage Systems, the performance, availability and scalability of the Metadata Server (MDS) cluster are critical. Traditional MDS cluster suffers from frequent metadata access and metadata movement within the cluster. In this paper, we present a new method called Hashing Partition (HAP) for MDS cluster design to avoid these overheads. We also demonstrate a design using HAP to achieve good performance of MDS cluster load balancing, failover and scalability.

1. Introduction

Unlike traditional file storage systems with metadata and data managed by the same machine and stored on the same device [1], the object-based storage system separates the data and metadata management. An Object-based Storage Device (OSD) [2] cluster manages low-level storage tasks such as object-to-block mapping and request scheduling, and presents an object access interface instead of block-level interface [3]. A separate cluster of MDS manages metadata and file-to-object mapping, as shown in Figure 1. The goal of such storage system with specialized metadata management is to efficiently manage metadata and improve the overall system performance. In this paper, we mainly address performance, availability and scalability issues for the design of MDS cluster in Object-based Storage Systems.

Two key concerns about MDS cluster are the request load of metadata and load balancing within the cluster. In our preliminary OSD prototype, which adopts the traditional directory sub-tree to manage metadata, we find that more than 70 percent of all file system access requests are for metadata when using Postmark [4] to access 0.5k files, as shown in Figure 2. Although

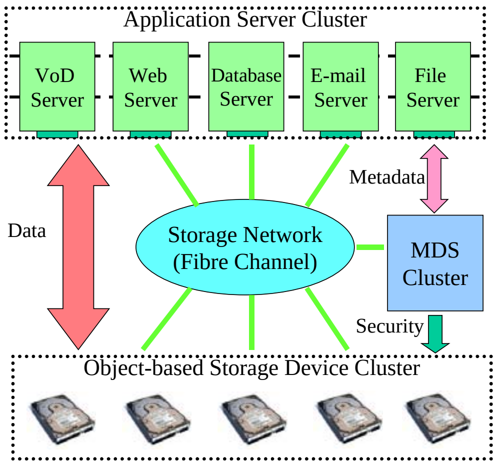
Application Server Cluster
VoD Server
Web Server
Database Server
E-mail Server
File Server
Data
Metadata
Storage Network (Fibre Channel)
MDS Cluster
Security
Object-based Storage Device Cluster

Figure 1. Object-based Storage System

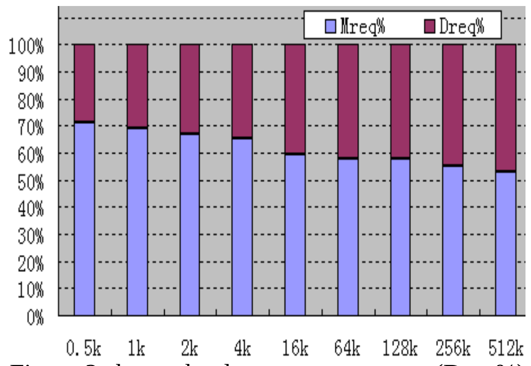

Figure 2 shows the data request percent (Dreq%) and the metadata request percent (Mreq%) of the total requests. This test is based on our OSD prototype (one client, one MDS and one OSD) connected by Fibre Channel, using Postmark (1000 files, 10 subdirectories, random access, 500 transactions).

the size of the metadata is generally small compared to the overall storage capacity, the traffic volume of such metadata access degrades the system performance. The large number of metadata requests can be attributed to the use of directory sub-tree metadata management.

Apart from metadata requests, an uneven load distribution within a MDS cluster would also raise severe bottleneck. Based on traditional cluster architecture, the performance of the load balancing, failover and scalability in the MDS cluster is limited, because most of these operations lead to the inevitable massive metadata movement within cluster. The Lazy Hybrid metadata management method [5] presented a hashing metadata management with the hierarchical directory support, which dramatically reduced the total number of metadata requests, but Lazy Hybrid did not deal with reducing metadata movement between MDSs for load balancing, failover and scalability.

This paper presents the new method called Hashing Partition (HAP) for MDS cluster design. HAP herein also adopts the hashing method, but focuses on reducing the cross MDS metadata movement in a clustered design, in order to achieve high performance of load balancing, failover and scalability.

The rest of the paper is organized as follows. The next section details the design of HAP and section 3 demonstrates our solutions of MDS Cluster load balancing, failover and scalability. Section 4 discusses MDS Cluster Rebuild. Finally, the conclusion of the paper is drawn in section 5.

2. Hashing Partition

Hashing Partition (HAP) provides a total solution for the file hashing, metadata partitioning, and metadata storage. There are three logical modules in the HAP: file

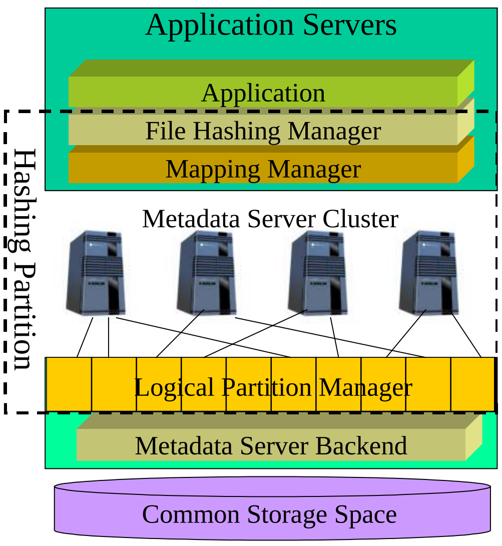

Figure 3. Hashing Partition

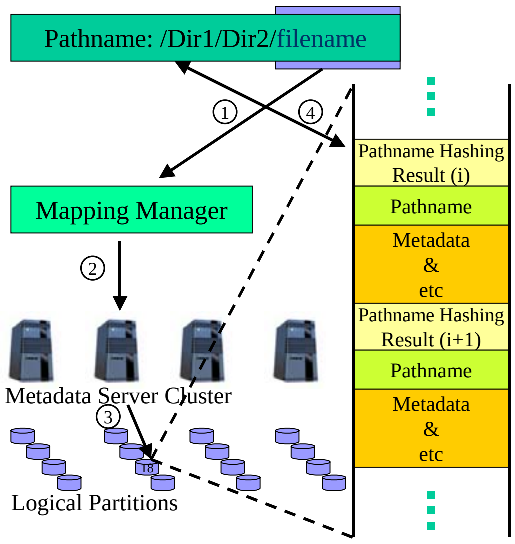

Figure 4. Metadata Access Pattern
①.Filename hashing, ②.Selecting MDS through Mapping Manager, ③.Accessing metadata by pathname hashing result, ④.Returning metadata to application server.

hashing manager, mapping manager, and logical partition manager, as shown in Figure 3.

In addition, HAP employs an independent common storage space for all MDSs to store metadata, and this space is divided into multiple logical partitions. Each logical partition contains part of global metadata table. Each MDS can mount and then exclusively access logical partitions allocated to it. Thus as a whole, MDS cluster can access a unique global metadata table.

The procedure of metadata access is described as follows. Firstly, file hashing manager hashes a filename to an integer, which can be mapped to the partition that stores the metadata of the file. Secondly, mapping manager can figure out the identity number of MDS that currently mounts that partition. Then client can send metadata request with the hash value of pathname to the MDS. Finally, logical partition manager located in MDS side accesses metadata on the logical partition in the common storage space. Figure 4 describes this efficient metadata access procedure. Normally, only a single message to a single metadata server is required to access a file's metadata.

2.1. File Hashing Manager

File hashing manager performs two kinds of hashing: filename hashing for partitioning metadata in MDS cluster, and pathname hashing for the metadata allocation in MDS. To access metadata of a file in MDS cluster, client needs to know two facts: which MDS manages the metadata and where the metadata is located in the logical partition. Filename hashing answers the first question and pathname hashing solves the second one. For example, if the client needs to access the file, "/a/b/filec", client uses the hashing result of "filec" to select MDS that manages the metadata. Then instead of accessing directory "a" and "b" to know where is the metadata of "filec", a hash result of "/a/b/filec", directly indicates where to retrieve the metadata.

But the filename hashing may introduce a potential bottleneck when a large parallel access to different files with the same name in different directories. Fortunately, the different hash values of various popular filenames, such as *readme* and *makefile*, make all these "hot points" distributed among MDS cluster and reduce the possibility of the potential bottleneck. In addition, even if certain MDS is over-loaded, our dynamic load balancing policy (section 3.1) can effectively handle this scenario and shift the "hot points" from overloaded MDS to the less-loaded MDSs.

2.2. Logical Partition Manager

Logical partition manager manages all logical partitions in the common storage space. It performs many logical partition management tasks, e.g. *mount*/*un-mount*, *backup* and *Journal recovery*. For instance, logical partition manager can periodically backup logical partitions to a remote backup server.

2.3. Mapping Manager

Mapping manager performs two kinds of mapping tasks: hashing result to logical partition mapping and logical partition to MDS mapping. Equation 1 describes these two mapping functions.

$$\begin{aligned}
&Pi = f(H(filename)) \\
&MDSi = ML(Pi, PWi, MWi) \\
&Pi \in \{0, Pn\}; H(filename) \in \{0, Hn\}; MDSi \in \{0, Mn\} \\
&(Hn \geq Pn \geq Mn > 0)
\end{aligned} \tag{1}$$

Where, *H* represents a filename hashing function; *f* stands for the mapping function that transfers hashing result to partition number (*Pi*); *ML* represents the function that figures out MDS number (*MDSi*) from partition number and related parameters (*PW* and *MW* will be explained in section 3.1); *Pn* is the total number of partitions; *Hn* is the maximum hashing value and *Mn* is the total number of MDSs.

When *PW* and *MW* are set, mapping manager simplifies the mapping function *ML* to a mapping table MLT, which describes the current mapping between MDS and logical partition. It is noted that one MDS can mount multiple partitions, but one partition can only be mounted to one MDS. To access metadata, mapping manager can indicate the logical partition that stores the metadata of a file based on the hash result of the filename. Then through MLT, mapping manager knows which MDS mounts that partition and manages the metadata of the file. Finally the client contacts the selected metadata server to obtain the file's metadata, file-to-object mapping and security information. Table 1 gives an example of MLT. Based on this table, in order to access metadata on logical partition 18, client needs to send request to *MDS1*.

Table 1. Example of MLT

Logical partition Number	**MDS ID**	**MDS Weight**
0~15	0	300
16~31	1	300
32~47	2	300
48~63	3	300

3. Load Balancing, Failover and Scalability

3.1. MDS Cluster Load Balancing Design

We propose a simple Dynamic Weight algorithm to dynamically balance the load of MDSs. HAP assigns a MDS Weight (*MW*) to each MDS according to its CPU power, memory size and bandwidth, and uses a Partition Weight (*PW*) to reflect the access frequency of each partition. *MW* is a stable value if the hardware configuration of the MDS cluster does not change, and *PW* can be dynamically adjusted according to the access rate and pattern of partitions. In order to balance the load between MDSs, mapping manager allocates partitions to MDS based on Equation 2.

$$\frac{\sum PWi}{MWi} = \frac{\sum_{a=0}^{Pn} PWa}{\sum_{a=0}^{Mn} MWa} \tag{2}$$

Where, ΣPWi presents the sum of *PW* of all partitions mounted by *MDSi*; *Pn* stands for the total number of partitions; *Mn* presents the total number of MDSs.

In addition, each MDS needs to maintain load information about itself and all partitions mounted on it, and periodically uses Equation 3 to calculate new values.

$$MDSLOAD(i+1) = MDSLOAD(i) \times \alpha\% + MDSCURLOAD \times (1-\alpha\%) \qquad (3)$$
$$PLOAD(i+1) = PLOAD(i) \times \beta\% + PCURLOAD \times (1-\beta\%)$$

Where, *MDSCURLOAD* is the current load of the MDS; *PCURLOAD* is the current load of the logical partition; *MDSLOAD(i)* represents the load status of a MDS at time *i*; *PLOAD(i)* stands for the load status of a logical partition at time *i*; α and β are constant used to balance the effects of old value and new value.

However, MDSs don't report their load information to the master node, e.g. one particular MDS, until a MDS alarms in its overloaded situation, such as the MDSLOAD exceeding the preset maximum load of the MDS. After receiving load information from all MDSs, the master node sets the *PW* using new *PLOAD* values. Then according to new *PW* and Equation 2, HAP shifts the control of certain partitions from the over-loaded MDS to some less-loaded MDSs and modifies MLT accordingly. This adjustment does not involve any physical metadata movement between MDSs.

3.2. MDS Cluster Failover Design

Typically, a conventional failover design adopts a standby server to take over all services of the failed server. In our design, the failover strategy relies on the clustered approach. In the case of a MDS failure, mapping manager assigns other MDSs to take over the work of the failed MDS based on Equation 2. Then the logical partition manager allocates the logical partitions managed by the failed MDS to its successors, as shown in Figure 5. So application servers can still access metadata on the same logical partition in the common storage space through the successors.

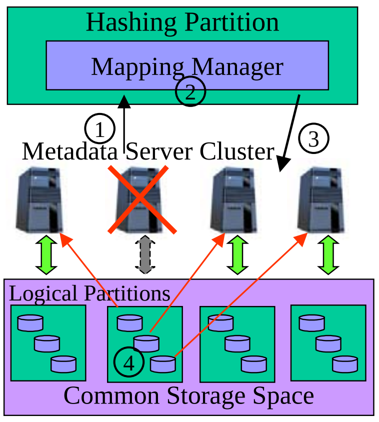

Figure 5. MDS cluster failover procedure ①.Detecting the MDS failure, ②.Recalculating MW and adjusting MLT, ③.Other MDSs take over logical partitions of the failure one, ④.Journal recovery

3.3. MDS Cluster Scalability Design

HAP significantly simplifies the procedure to scale the metadata servers. If the current MDS cluster cannot handle metadata request effectively due to the heavy load, new MDSs can be dynamically set up to release the overhead of others. HAP method allows the addition of MDS by adjusting *MWs* and thus generating a new MLT based on *ML*. This process doesn't touch the mapping relationship between filename and logical partition, because the number of logical partitions is unchanged. Following the new MLT, logical partition manager un-mounts certain partitions from existing MDSs and mounts them to the new MDS. This procedure also doesn't introduce any physical metadata movement within MDS cluster.

4. MDS Cluster Rebuild

Although HAP method can dramatically simplify the operation of MDS addition and removal, HAP actually has a scalability limitation, called Scalability Capability. The preset number of logical partitions limits Scalability Capability, since one partition can only be mounted and accessed by one MDS at a time. For instance 64 logical partitions can only support up to 64 MDSs without rebuild. In order to improve Scalability Capability, we can add storage hardware to create new logical partitions and redistribute metadata among the entire cluster. This metadata redistribution introduces multi-MDS communication because the change in the number of logical partitions requires a new mapping function *f* in Equation 1, and affects the metadata location of the existing files in logical partitions. For example, after Scalability Capability is improved from 64 to 256, the metadata of a file may need to move from logical partition 18 to logical partition 74. The procedure that redistributes all metadata based on new mapping policy and improves Scalability Capability, is called MDS Cluster Rebuild.

In order to reduce the response time of MDS cluster rebuild, HAP adopts Deferred Update algorithm, which defers metadata movement and distributes its overhead. After receiving the cluster rebuild request, HAP saves a copy of the mapping function *f*, creates a new *f* based on the new number of logical partitions, and generates a new MLT. Then logical partition manager mounts all logical partitions including both the old and new according to the new MLT. After that, HAP responses immediately to the rebuild request and changes MDS cluster to a rebuild mode. Thus the initial operation for this entire process is very fast.

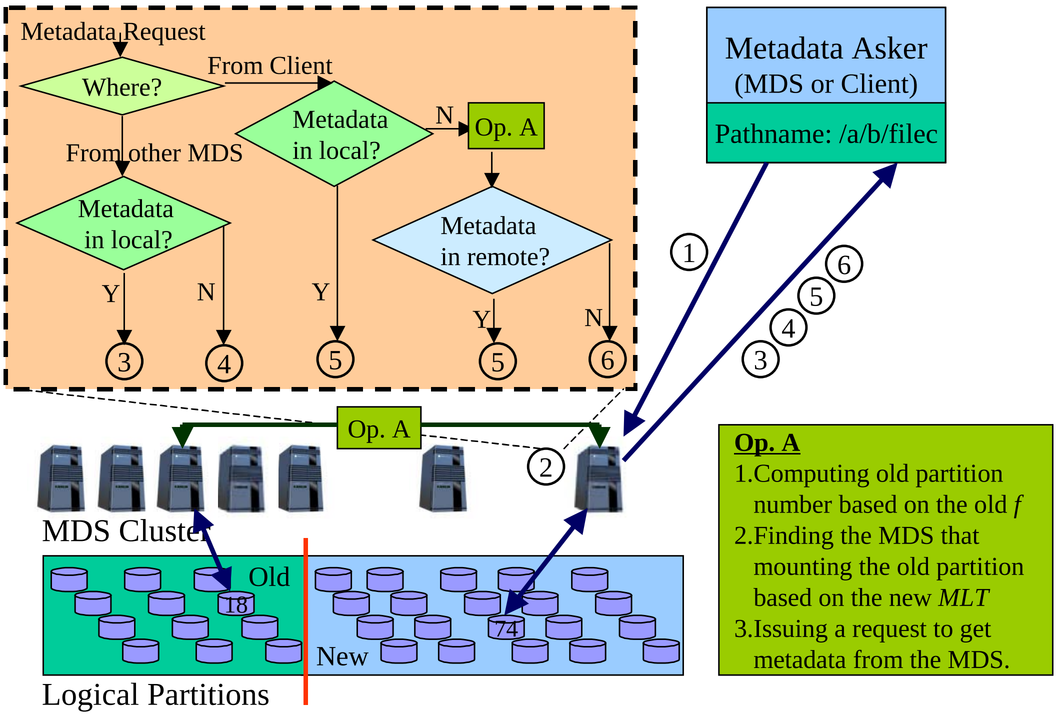
Metadata Request
Where?
From Client
From other MDS
Metadata in local?
N
Op. A
Metadata in local?
Metadata in remote?
Y
N
Y
Y
N
3
4
5
5
6
Metadata Asker (MDS or Client)
Pathname: /a/b/filec
1
6
5
4
3
Op. A
2
MDS Cluster
Old
18
74
New
Logical Partitions
Op. A
1.Computing old partition number based on the old f
2.Finding the MDS that mounting the old partition based on the new MLT
3.Issuing a request to get metadata from the MDS.

Figure 6. MDS Cluster Rebuild

①.Sending request to MDS based on new mapping result, ②.Searching for metadata and making judgment (the rectangle on the left shows the internal logic and Op. A is explained in the bottom rectangle), ③.Returning metadata and deleting it in local, ④.Reporting Error, ⑤.Returning metadata, ⑥.Wrong filename

During the rebuild, the behavior of the system is as if all the metadata had been moved to the right logical partitions. Actually, HAP updates or moves the metadata upon the first access. If a MDS receives a metadata request, and the metadata hasn't been moved to the logical partition that is mounted by it, the MDS needs to use the old mapping function *f* to calculate the original logical partition number based on the filename. Then through the new MLT, the MDS can find the MDS that currently mounts the original logical partition and send a metadata request to it. So the MDS can retrieve the metadata and complete the metadata movement. Figure 6 describes this procedure in detail. In addition, in order to accelerate the metadata movement progress, HAP can also adopt an independent thread to travel the metadata database and move the affected metadata only during the spare time of system.

5. Conclusion

We present a new method of Hashing Partition to manage metadata server cluster in large distributed object-based storage system. We use hashing method to avoid the numerous metadata accesses, and use filename hashing policy to remove the overhead of multiple MDS communication. Furthermore, based on the concept of logical partitions in the common storage space, HAP method significantly simplifies the implementation of the MDS cluster and provides efficient solutions for load balancing, failover and scalability.

The design described in this paper is part of our BrainStor project that targets to provide the full object-based storage solution. Currently we are implementing the Hashing Partition management for MDS Cluster in the BrainStor prototype. We also plan to explore the application of BrainStor technologies in Grid storage.

References

[1] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, and F. D. Smith. "Andrew: A distributed personal computing environment", *Communications of the ACM*, 29(3):184–201, Mar. 1986.
[2] R. O. Weber. "Information technology—SCSI object-based storage device commands (OSD)", Technical Council Proposal Document T10/1355-D, Technical Committee T10, Aug. 2003.
[3] Thomas M. Ruwart, "OSD: A Tutorial on Object Storage Devices", *19th IEEE Symposium on Mass Storage Systems and Technologies*, University of Maryland, Maryland, USA, April 2002.
[4] KATCHER, J. "Postmark: A new file system benchmark", Tech. Rep. TR3022 (Oct. 1997). Network Appliance.
[5] Scott A. Brandt, Lan Xue, Ethan L. Miller, and Darrell D. E. Long. "Efficient metadata management in large distributed file systems", *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 290–298, April 2003.