# Clustering Metadata Servers for Extremely Scalable and Reliable Services

Keun-Tae Park, Myung-Hoon Cha, Young-soo Min, Young-Kyun Kim, and Han Namgoong
Storage System Research Team
Internet Platform Research Department
S/W Content Research Laboratory
Electronics and Telecommunications Research Institute
{ktpark, mhcha, minys, kimyoung, nghan}@etri.re.kr

***Abstract* — LakeFS[1] is a cluster file system designed for high scalable and reliable storage service. It provides excellent scalability and availability on data I/O operations. However, it lacks the ability to scale up the metadata operations, which makes the metadata server (MDS) as a single point of bottleneck especially when full scan of metadata is needed. In this paper, we present a way to cluster MDSs to solve the bottleneck problem. We also change the file system utilities to exploit the cluster fully. Experimental results show that the throughput of metadata operation is enhanced linearly as the number of MDS participating in the cluster increases.**

***Keywords* — Cluster file system, extreme scalability, high reliability, metadata cluster.**

## 1. Introduction

In our previous research [1], we propose a cluster file system, called LakeFS, which is designed for high scalable and reliable storage service. Two key features of the LakeFS are replication based data redundancy for data reliability and separation of metadata operation from data operation for I/O scalability.

As a file is replicated over multiple servers in LakeFS, it can endure multiple concurrent data corruption or server failures until all the servers storing the file are down. Additionally, LakeFS automatically recovers the partially corrupted or lost file, which means some of, but not all replicas are corrupted or lost, by replicating it to another server.

Separation of metadata server (MDS) from normal data I/O can make data I/O nearly independent of each other and scalable to the number of data servers (DS). For a normal file operations, every client contacts metadata server only once to retrieve all the file metadata. Then, it can directly communicate with the data server that stores a replica of the file and provides the highest throughput. As metadata server is fully responsible for metadata operations, data servers can concentrate on the efficient data I/O.

Experimental results on a small cluster environment with 128 nodes under the Internet streaming service traffic pattern showed that LakeFS is scalable to the number of servers and sufficiently reliable under server failures. However, under intensive workload where more than 100 million files are stored and accessed, we found that MDS became the bottleneck point of the system.

The main reason why the MDS becomes the bottleneck is the metadata eviction from the memory cache. Almost all metadata is accessed from the MDS disk, which leads one hundredth performance degradation from the cache based operation. With more than 100s of million files, cache thrashing occurs easily even considering the popularity variation of files. Moreover, automatic file recovery of LakeFS requires sequential scan of the entire metadata to find the target of replication, which aggravates the cache trashing more severely.

In this paper, we present MDS clustering mechanism to solve the MDS bottleneck. All MDSs in the cluster share the burden of metadata operations, which drops the processing load in each MDS and solves the bottleneck if exists. Moreover, it also reduces working set size of each MDS, which even enables all the metadata cached in the memory and greatly enhances the processing time of each metadata operation.

Clustered MDS structure in LakeFS has following characteristics. First, it provides efficient metadata distribution among MDSs, which also guarantees the availability of metadata under MDS crash. Second, client-side MDS selection mechanism intelligently distributes client requests based on the context of the request in order to limit the working set of each MDS, which enhance the cache locality of metadata. Last, file system utilities that access the metadata, especially file recovery utility, are also enhanced to exploit the scalability of MDS cluster.

This paper is consisted as follows. Previous works on scaling up the metadata operation is presented in Section 2. Overall structure and operation of clustered MDS in LakeFS are explained in Section 3. We verified the scalability and availability enhancement of metadata operations through the experiment in Section 4. Section 5 concludes the paper.

## 2. Related Works

Lots of previous works try clustering of servers to alleviate the burden of a single server. Even in the case of metadata storage, there exist several previous researches.

This work was supported by the IT R&D program of MIC/IITA [2007-S-016-01, A Development of Cost Effective and Large Scale Global Internet Service Solution]

Many researches [2][3][4] divide the metadata space into several partitions and make each MDS take charge of a partition. Based on the dividing policy, they are classified into two types; sub-tree partitioning [2][3] and hashing distribution [4]. Sub-tree partitioning schemes assign all the metadata of each sub-directory and its files to an individual MDS. Hence, there exists no overlapped metadata between any two MDSs. According to their assignment policy, sub-tree partitioning is more classified into static [2] and dynamic [3].

Sub-tree partitioning needs global mapping table that indicates which MDS takes charge of each sub-directory. Hashing distribution [4] reduces the size of global table into a hash function. Every metadata is distributed over each MDS based on its hashing value. As a tradeoff of simplified global information, however, it damages the spatial locality of metadata in each MDS.

Basically, LakeFS replicates all the metadata into all MDSs for reliability. For only metadata cache, which allows loss of data, partitioning issues are considered. Since the target of LakeFS is extremely scalable service, sub-tree partitioning induces the need of too large global mapping table. Hence, LakeFS applies modified hashing distribution scheme in its metadata cache management, which also maintains spatial locality. Detailed mechanism is explained in Section 3.2.

LakeFS MDS cluster relies on MySQL DB [5] for metadata storage. It provides replication scheme that copies the selected contents of a DB (master) to another DB (slave). Hence, all the MDS nodes have the same metadata image through the replication. As the replication is asynchronous, however, it takes some time to complete replication after a DB update. LakeFS considers the possibility of late update at slave DB in its design, so that it contacts the master DB again whenever it detects old version of metadata in a slave DB.

## 3. Cluster MDS

### 3.1. Basic structure

As a scalable way to solve the memory shortage in the MDS, we try to cluster multiple nodes as MDS. Basic structure of LakeFS MDS cluster is shown in the Figure 1.

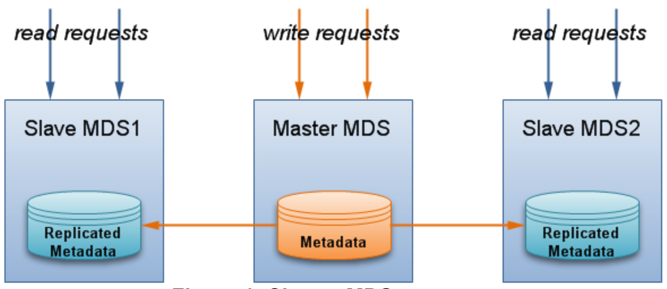

**Figure 1. Cluster MDS structure**

A MDS cluster consists of one master MDS and multiple slave MDSs; all the metadata update operation should be directed to the master, but metadata read can be serviced by any slave MDS. Metadata stored in the master is propagated to the entire slave MDSs asynchronously.

With this simple cluster structure, LakeFS can achieve following advantages.

- First, read requests, majority of the Internet requests, are distributed over multiple MDSs and each MDS experiences smaller workload. For more efficient use of memory in each MDS, we also propose a way to distribute read requests well in Section 3.2.
- Second, metadata consistency and concurrency control point is still centralized at the master MDS. Hence, there is little possibility of metadata conflict between MDSs, and metadata integrity check and fix is simple and easy.
- Last, each slave MDS can work as a stand-by node of the master. When the master fails, any slave can take charge of the master MDS role. Until all the master and slave MDSs fail, LakeFS can continue file service without interruption.

### 3.2. Request distribution

LakeFS uses client-side request distribution based on its requesting contents. When a client mounts a LakeFS, it gains the list of MDSs that participates in the MDS cluster. As explained in Section 3.1, all the write requests should be directed to the master MDS only. However, client can contact any MDS for metadata read request. This section explains the way for client to select a MDS to maximize the cache efficiency of each MDS.

LakeFS client uses FUSE [6] to support POSIX API compatibility. Hence, every client request has full path of the requesting file as an argument.

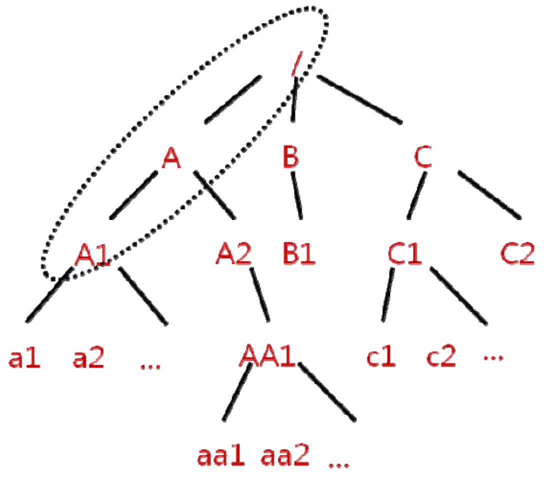

**Figure 2. Hashing on a directory tree**

Each LakeFS client has hash function to get the corresponding slave number of the given file. Hashing depth of the given full path file name is fixed as N level. For example, hash in Figure 2 works on a directory depth of two (/A/A1/a1 → Hash(/A/A1), /A/A2/AA1/aa1 → Hash(/A/A2)). If the full path has less then N level directory depth, client randomly selects slave MDS.

Hash based request distribution of LakeFS three advantages as follows.

- First, with a fixed hash function, multiple requests of the same file are directed to the same slave MDS. It greatly enhances the cache efficiency of each MDS.
- Second, metadata of files in the same directory tends to be served by the same MDS. Many applications and even some file system utilities like *ls* require information of multiple files in a directory; hence, spatial locality of metadata is very important.
- Last, upper level directories in the directory hierarchy are cached in the entire slave MDSs. As directories near the root directory tends to be requested frequently, caching of such directories in all MDSs avoids metadata hot spot problem.

## 3.3. Parallel recovery

Sequential scan of the entire metadata in automatic recovery is major source of cache eviction. We partition the range of the metadata scan and assign each non-overlapping partial range to a different slave. Figure 3 shows the operation flow of the file system check and the corrupted file recovery.

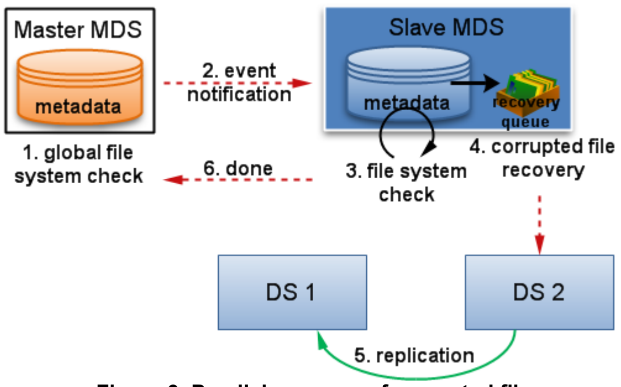

**Figure 3. Parallel recovery of corrupted file**

There are two options in partitioning the metadata scanning range. First approach tries to maximize the utilization of metadata cached in the memory. It reconstructs the request pattern by clients using the hash function, and then picks up metadata that matches to the pattern since it will be cached already by previous requests from clients. Figure 4-a shows the sequence of the first approach;

a. Construct N-level directory tree
b. Apply hash function to all the possible paths
c. Traverse inode entries under the directories that has the hash number assigned to the slave MDS

Even though the first approach maximizes the cache utilization, tree traverse to find the pattern matched metadata induces non-negligible search overhead. Hence, the second approach simply splits the inode range by the number of slave MDSs and each piece becomes the scanning range of a different slave MDS as shown in Figure 4-b. As the metadata is stored in the order of the inode number, there is little overhead in searching the target metadata to scan. However, it also damages the cache efficiency since scanned metadata may evict the cached metadata.

Two approaches have their own merits and there is no optimal solution for all the cases. For a metadata storage that has workload pattern of high memory usage or has much overhead on fetching the metadata from the storage to the memory, the first one is better approach. However, LakeFS uses DB as metadata storage and DB query overhead overwhelms the benefits from the metadata cache hit. Hence, current implementation of LakeFS applies the second approach.

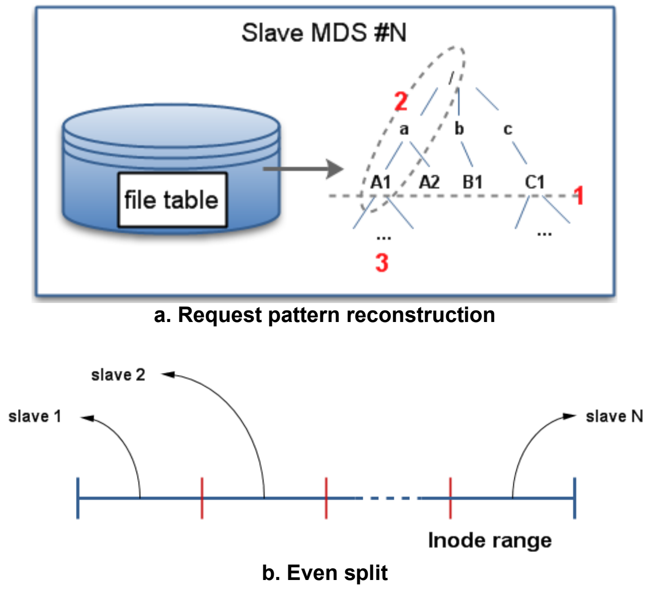

**Figure 4. Two approaches to partition the range**

As file recovery requires update of the file metadata, slave MDS should inform recovered file list to the master MDS (step 6 in Figure 3). Then, the master solves the possible update conflict, like file update by other clients during recovery, by removing the replicated file that becomes stale, and finalize the update of the metadata.

## 4. Experimental result

## 4.1. Experimental setup

We evaluate the performance of LakeFS on a small test-bed that consists of 18 identical nodes as shown in Figure 5. Each node has 1GB memory, an 100GB SATA hard disk, an 1 Gbit Ethernet interface, and Linux OS, whose kernel version is 2.6.18. All nodes are on the same rack and interconnected through an Ethernet switch.

Experimental setup consists of 10 LakeFS clients, 3 LakeFS DS nodes, and 1~5 LakeFS MDS nodes. The number of

LakeFS MDS nodes is varied to check the effect of MDS clustering.

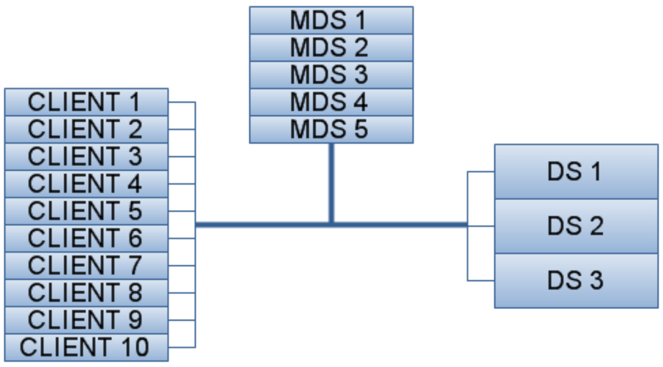

**Figure 5. Experimental setup**

## 4.2. Metadata read operation

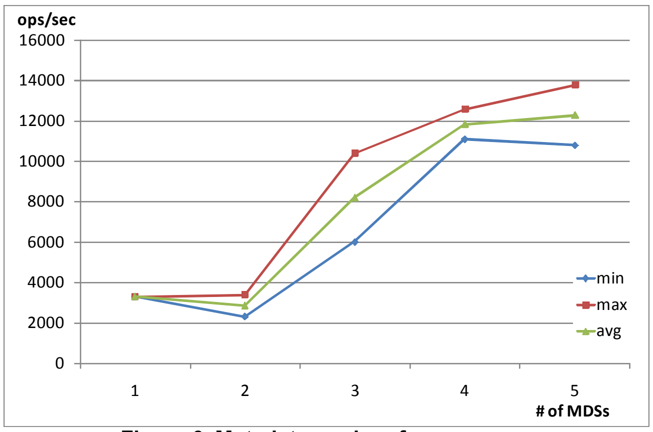

**Figure 6. Metadata read performance**

We first check the scalability of the MDS cluster by increasing the number of MDSs in LakeFS as shown in Figure 6. To simulate the metadata read operations, one client executes all possible combination of file operations that include both read and write operations, and other 9 clients executes 30 processes that infinitely invoke '*ls –alR; find*' that has only metadata read operations.

The result shows that the metadata performance increases as the number of MDS increases. However, the increase is not linear since the clustering has non-negligible overhead due to metadata replication. With two MDSs, the performance of LakeFS metadata operation even decreases. As the replication overhead increases as the number of MDS increases, the slope of performance increase decreases (5 MDSs case in the figure).

## 4.3. Linear scan performance

Next, we check the performance of linear metadata scan, which is the major source of MDS load. We check the performance of file system check utility as the number of MDS increases and compare it with that of single MDS case. 50000 files of 1MB are created and checked by the file system check utility. For each case, we check the files system 5 times and get minimum, maximum, and average performed operations per second.

The metadata scan performance increases linearly as the number of MDS increases as shown in Figure 7. For one slave case, clustered MDS has 12% performance degradation due to communication overhead between the master and the slave MDS. However, if the number of slaves is greater than one, clustered MDS provides outstanding performance boost over the single MDS case.

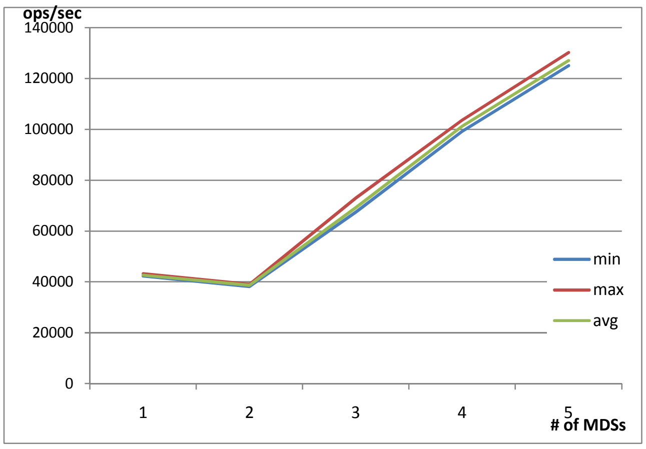

**Figure 7. Metadata linear scan performance**

To check the effect of communication cost between the master MDS and slaves, we calculate the performance gain of 4 node MDS cluster over single MDS as the number of files to check increases as shown in Figure 8.

Due to the communication overhead, cluster MDS becomes more profitable as the number of files to check increase. Since LakeFS targets on the extremely scalable environment that has billions of files, cluster MDS has more impact on the real environments.

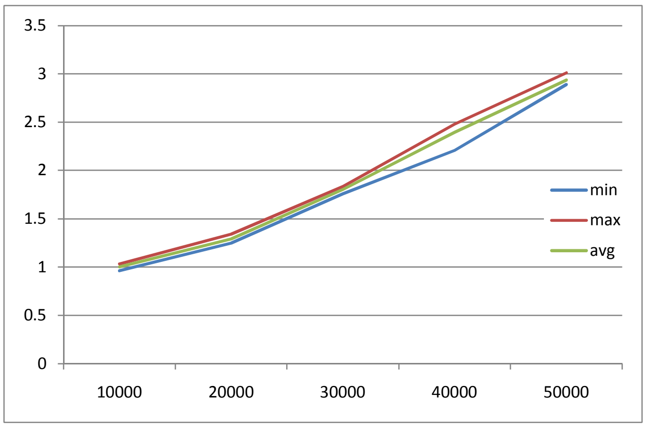

**Figure 8. Performance gain according to the number of files**

## 5. Conclusion

In extremely scalable environments, single MDS of LakeFS became a performance and reliability bottleneck. In this paper, we present simple and efficient way to scale up LakeFS MDS by clustering multiple nodes. It focuses on the scalability of the metadata read operations, which are most of metadata operations, and the consistency of the write operations. Client-side request distribution mechanism properly shapes the workload on each MDS, so that metadata cache utilization is maximized and the load of each MDS is evenly distributed.

Still, LakeFS needs excessive field test to prove its reliability and scalability. However, preliminary results from the test-bed show that we are on the right direction. Now, we start to apply LakeFS on commercial streaming service, which will give us more accurate evaluation of LakeFS under real circumstances.

## References

[1] K. Park, H. Kim, Y. Kim, S. Lee, Y. Kim, and M. Kim, "Lake: Towards highly manageable cluster storage for extremely scalable services", *The 2008 International Conference on Computational Science and Applications*, Perugia (Italy), 2008, pp. 122-131.

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

[3] 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.

[4] 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 the IEEE/11 the NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2003)*,April 2003,pp.290-298.

[5] MySQL, http://www.mysql.com

[6] File system in userspace, http://fuse.sourceforge.net