WPAR: A Weight-based Metadata Management Strategy for Petabyte-scale Object Storage Systems

Wujuan Lin
Hitachi Asia Ltd., Singapore
wjlin@has.hitachi.com.sg

Qingsong Wei
Data Storage Institute, Singapore
wei_qingsong@dsi.a-star.edu.sg

Bharadwaj Veeravalli
National Univ. of Singapore
elebv@nus.edu.sg

Abstract

In an Object-based network storage system, metadata access is decoupled from the data transferring path to improve system performance, scalability, and management. Designing an efficient metadata management scheme in such a system is critically important to the overall system performance and poses lots of challenges to the system designers. Traditional metadata management schemes, either partitioning the metadata statically, or using pure hashing methods, are not able to adapt to the dynamic metadata workload, and hence suffer from scalability and hotspot problems. In this paper, we propose a new metadata management strategy, referred to as Weighted Partitioning and Adaptive Replication (WPAR) scheme, taking into account the weight (CPU power, memory capacity, network bandwidth, etc.) of each metadata server. Our objective is to efficiently distribute and dynamically replicate the metadata, based on the weights, to balance the workload and reduce hotspots within the metadata server cluster.

1. Introduction

One of the most critical issues today in network storage is the difficulty and the high cost of managing both the data and storage devices. Recent studies by Gartner and IDC show that the purchase price of storage devices comprises only 5% to 10% of the total cost of storage ownership, while the majority goes to the cost of administration, device management tasks, and backup/recovery procedures [13]. The problem is further hastened by the huge growth of information (e.g., the rapid growth of Internet and the proliferation of data-intensive network services, such as satellite or medical image processing and multimedia applications) and the extended data lifecycle, which translate to more storage devices and data to manage. On the other hand, enterprises desire their large datasets can be made globally available to all the clients across multiple platforms simultaneously, while the systems also have to be reliable, secure, and managed by a small number of staff. Industries and academic researchers are thus driven to look for more efficient solutions for data storage, retrieval, and management.

Object-based network storage systems, which leverage the strengths of DAS (Direct Attached Storage), NAS (Network Attached Storage), and SAN (Storage Area Network) into a single framework, have the potential to provide high system performance, scalability, robust data security, and easy data sharing across multiple platforms [2, 12, 21, 22]. By decoupling the metadata access from data access and enabling the object-based storage devices (OSDs) more intelligence (e.g., self-management, self-optimization, and QoS provision), the object-base network storage system is emerging as a new network storage paradigm.

Rather than simply putting data on tracks and sectors, data in an OSD are organized as objects that encapsulate user data and a set of attributes. The user-accessible attributes describe characteristics of the object, and hence allow the OSD to make decisions regarding data layout, QoS, or Security on a per-object basis [1, 8, 10, 11]. The offloading of these functionalities from a traditional file system into OSD devices also simplifies the data sharing across multiple platforms, by presenting a standard object interface (SNIA/T10 OSD-SCSI standard [16]). Furthermore, the Fixed Content Aware Storage technical work group (FCAS TWG) of SNIA, is currently working on a new specification, called eXtensible Access Method (XAM), to provide an object-oriented API between application and storage system [28]. The concepts of OSD and XAM are a natural match and will supplement each other.

As now the OSD devices can intelligently manage their own on-disk objects and ensure security, files are usually striped into objects across multiple OSDs, and the clients (or applications servers) are allowed to access the objects directly and in parallel to maximize the system performance. A dedicated metadata server (MDS) cluster is used as a global resource to maintain the file system namespace, find out the locations of objects, and ensure secure access to these objects. The clients will first consult the MDS clus-

0-7695-3097-4/08 $25.00 © 2008 IEEE
DOI 10.1109/SNAPI.2007.14

ter for the object locations and access credential, and the subsequent object read/write operations are carried out directly between the clients and OSDs without the interruptions from MDSs. By decoupling the metadata access from file read and write operations, OSD systems provide an out-of-band metadata management and thus remove the file server bottlenecks found in today's NAS systems [12, 13].

Although the size of metadata is relatively small when compared to that of application data, it has been reported in [14, 19, 27, 29] that metadata operations may make up over 50% of all the system file operations, making the behavior of metadata server (MDS) cluster critically important to the overall system performance. One of the key challenges in the MDS cluster design lies in how to partition the metadata among the MDSs to maintain a balanced or near-balanced workload and avoid "hotspots" in the MDS cluster. Different metadata partitioning strategies may cause different hotspot problems. For example, directory subtree partitioning (DSP) technique suffers hotspot problem when a single file or directory becomes popular, and a hashing method based on filename leads to a bottleneck when large parallel accesses go to different files with the same name in different directories [5, 27]. To minimize the hotspot problem, more sophisticated metadata partitioning strategies together with adaptive metadata replication/migration schemes are required for the changing metadata workload [20, 21, 25, 26]. In turn, when a metadata can be replicated/cached in more than one MDS in the cluster, metadata consistency has to be taken care for metadata update. Data serialization mechanisms (e.g., *time-stamps* [18], *locking* [4], or leasing [23] mechanisms) are required to update or invalidate the redundant copies.

In this paper, we focus on the design of a petabyte-scale object-based storage system where there are 1000s clients, 10s MDSs, and 100s OSDs in the system. The metadata management in such a system becomes more complicated when compared to a small-scale system. We have to consider a heterogeneous environment where each MDS may have different memory capacity, CPU power, and network bandwidth. Evenly distributing metadata to the servers in the cluster may not maximize the overall performance because different servers may be bound by different resource constraints. For example, servers managing widely but not frequently accessed metadata may be limited by memory capacity while servers managing busy "hotspots" may be bound by CPU or network bandwidth but not the memory capacity [27]. Therefore, in this paper, we propose a new metadata management strategy, referred to as Weighted Partitioning and Adaptive Replication (WPAR) scheme, taking into account the weight (CPU power, memory capacity, network bandwidth, etc.) of each metadata server. Our objective is to efficiently distribute and dynamically replicate the metadata, based on the weights, to balance the workload and reduce hotspots within the metadata server cluster.

The remainder of this paper is organized as follows. In Section 2, we give a brief study of metadata management methods found in the literature. In Section 3, we present our weight-based metadata partitioning mechanism and study its performance in terms of workload balancing. In Section 4, we introduce our adaptive metadata replication scheme that can further balance the workload and minimize hotspots. In Section 5, we conclude this paper with a summary and discuss several directions for further research.

2. Related Work

There have been a number of research efforts in recent years to address the metadata management issues in an OSD system.

DSP and Hashing are two popular approaches used in traditional and some modern distributed file systems [5, 24, 27]. By partitioning the metadata of portions of the entire directory hierarchy to different MDSs, DSP method (used in NFS [17], Sprite [15], and others) maintains the directory semantic and has relatively good performance, together with caching and prefetching techniques. However, DSP suffers from bottleneck problems when a single file, or a directory sub-tree, becomes popular. Furthermore, DSP method cannot handle well when the file system expands in an irregular manner, resulting in a very coarse granularity workload distribution among MDSs. Hashing strategy (implemented in Lustre [3], zFS [20], and others) distributes the metadata across MDSs based on some unique file identifier, such as file name or path name, providing a more evenly distributed workload in the MDS cluster. Hashing strategy also provides a one-step metadata lookup, without searching a metadata lookup table as required in DSP method, because a client can locate and contact the MDS that manages the requested metadata directly. However, similar "hotspots" problem exists in hashing strategy when there are a large amount of parallel accesses to the files with same hashing output. Moreover, partitioning the metadata by hashing eliminates the directory hierarchical locality and is therefore not able to take full advantage of caching.

Lazy Hybrid (LH) metadata management method [5] intends to combine together the DSP and hashing method. Instead of using hash value directly to indicate which metadata server to access, it uses the hash value as an index to the global metadata look-up table (MLT). The purpose is to reduce the metadata movement due to MDS cluster resizing, in which case both DSP and hashing method encounter difficulties. Nevertheless, this requires two-step indirect metadata access, slowing down the metadata lookup

process, and makes the MLT itself become a bottleneck resource.

Dynamic directory subtree partitioning method (DDSP) [27] is an extension to DSP method, integrating with an collaborative caching scheme to adapt to the file system changing workload. DDSP has a number of advantages in terms of locality of reference, cache management, and flexibility. To minimize the hotspots problem, DDSP attempts to limit the number of clients who know where to get the requested metadta. Requests will then be randomly sent and forwarded within the MDS cluster, leading to unexpected long request response time and inefficient resource (e.g., network bandwidth, CPU, et.) utilization.

3. Metadata Partitioning Strategy

We should now present the design of our metadata management strategy. As shown in Figure 1, in the MDS cluster, we use a global storage pool shared access by all the MDSs as the permanent storage for the metadata. The global storage pool is organized into LUNs (Logical Unit Numbers). Each MDS can mount and exclusively access any of the LUNs.

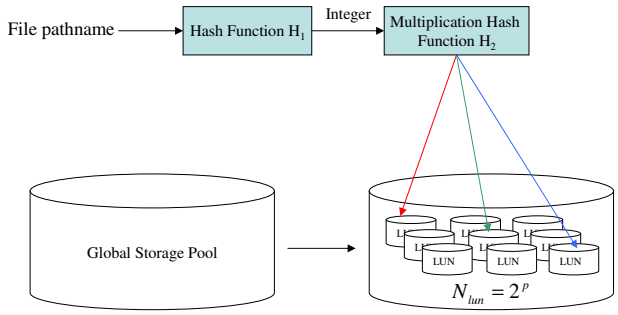


Figure 1. File full pathname is hashed to the global storage pool

Given a file, the file full pathname will first be hashed to one of the LUNs for metadata allocation and lookup. As the path name is a character string, we first use a hash function H1 (r5_hash in our experiments, which is a default hash function in Reiserfs to hash a file name to integer) to convert the path name into an integer. The integer will then become the input key of a multiplication hash function H2, as follows, to determine the LUN that will store the metadata,

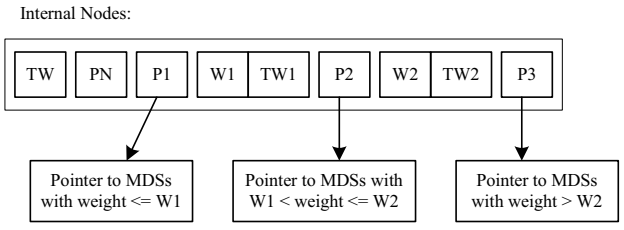
$$H_2(k) = \lfloor N_{lun}(kA \mod 1) \rfloor \quad (1)$$

Where $A \approx (\sqrt{5}-1)/2$ is a constant and $N_{lun} = 2^p$ is the total number of LUNs (p is an integer). Under these conditions, the multiplication hash function can have a better output [9]. Without loss of generality, the number of LUNs should be greater than the number of metadata servers. Only then it makes sense to assign different number of LUNs to the metadata servers with different weights. The purpose we use a global storage pool instead of individual storage attached to each server is to minimize the metadata movement during adding or removing metadata servers, by simply reassigning the LUNs, instead of physically moving the metadata.

3.1. Weight-based Metadata Partitioning

As mentioned in Section 2, both DSP and LH methods maintain a metadata lookup table (MLT) for metadata allocation and lookup. One of the disadvantages of this lookup table is that when the number of metadata servers becomes enormous, the table will become very large, requiring $O(N_{mds})$ to get the MDS that manages the requested metadata, where N_{mds} is the number of MDSs in the system. Further, when the file system expands in an irregular manner, or when a metadata server is added or removed from the cluster, the system administrators have to manually reconfigure the MLT to maintain a balanced workload within the cluster. Therefore, instead of lookup table, we use a B+-tree data structure in our design for metadata allocation and lookup.

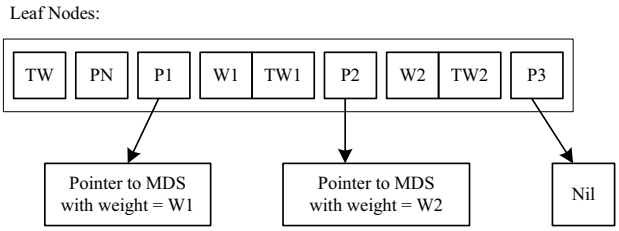
A B+-tree is called a *balanced tree* as any path from the root node to a leaf node has the same length, requiring only $O(m \cdot log_{(\lceil \frac{m}{2} \rceil + 1)} N_{mds})$ for metadata lookup, where m is the order[1] of the B+-tree. In this paper, we carefully design a variant B+-tree, incorporating with a universal hashing method [6], to partition the metadata based on the weight (CPU power, memory capacity, network bandwidth, etc.) of each MDS. Figures 2 and 3 show the data structures of internal nodes and leaf nodes of our variant B+-tree, respectively.



W1/W2: the key in an internal node, whose value is equal to the weight of a MDS
TW1: the total weight of MDSs in the subtree pointed by P1
TW2: the total weight of MDSs in the subtrees pointed by P1 and P2
TW: the total weight of MDSs in the subtrees pointed by P1, P2, and P3
PN: A prime number greater than N_{lun} and TW

Figure 2. Data structure of Internal Nodes

[1] A B+-tree with order m ($m \geq 2$) contains at most m keys and $m+1$ pointers at each node.



W1/W2: the key in a leaf node, whose value is equal to the weight of a MDS
TW1/TW2: TW1=W1, TW2=W2
TW: the total weight of MDS1 and MDS2, i.e., TW=W1+W2
PN: A prime number greater than N_{lun} and TW

Figure 3. Data structure of Leaf Nodes

The B+-tree will first be constructed based on the weight of each MDS. The values of TW, PN, and TW1/TW2 (defined in Figures 2 and 3) can then be calculated in a reverse manner from the leaf nodes to the root node.

As mentioned earlier, a file name is first hashed to a LUN in our global storage pool. In order to find the MDS that manage a given LUN, we need to search the B+-tree, which can be constructed at any node in the system by exchanging heartbeat messages. The searching process in our design is novel and different from that in a traditional B+-tree. We design a universal class of hash functions that will be used during the searching process as follows,

$$H(LUN,TW1,TW2) = ((TW1*LUN+TW2) \bmod PN) \bmod TW \quad (2)$$

It should be noted that for different nodes, the values of TW1, TW2, PN, and TW may be different as well. Following the similar steps in [6], we can prove that the class of hash functions defined by Equation (2) is indeed universal. The main idea behind the universal hashing method is to provide a class of carefully designed hash functions that are independent of the keys to be stored, and can be chosen randomly to yield provably good performance on average.

By using the variant B+-tree and universal hashing method, our objective is to partition the metadata based on the weight of each MDS, so that a MDS with higher weight will get higher probability to manage more metadata. Figure 4 illustrates the B+-tree searching process for metadata partitioning and lookup.

Given a LUN, when searching the B+-tree, a universal hashing function will be called to determine which subtree need to search further for an internal node, or to find out the corresponding MDS that manages the LUN for a leaf node. It should be noted that although the tree order $m = 2$ in our current design, the extension to the case where $m > 2$ is straightforward.

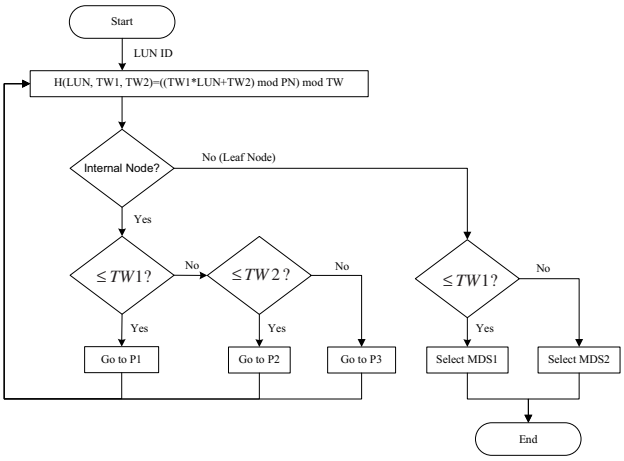


Figure 4. Metadata partition and lookup

3.2. Experimental Results

In this section, we compare the performance of our metadata partitioning method with DSP, Hashing, and LH methods, in terms of workload balancing.

We used the AUSPEX file system trace [7] as the workload trace for our performance evaluation. The trace follows the NFS activity of 237 clients serviced by an Auspex file server over the period of 1 week, and was gathered by snooping Ethernet packets on four subnets in University of California, Berkeley. There are $31,752,708$ requests in the trace, where $11,939,762$ of them are metadata requests (about 40%). This further verifies that the design of MDS cluster is crucial to the OSD system, due to the large amount of metadata requests.

In our experiment, we consider a 8-nodes MDS cluster, numbering from [1,2,...,8], with weight of [10,20,...,80], respectively. The global storage pool shared by the MDS cluster is organized into 1024 LUNs. Figure 5 shows how the metadata of $152,346$ different files in the trace are partitioned to the MDS cluster. The Hashing method, as expected, randomly assigns the metadata to the MDS cluster, leading to an unpredictable result. DSP and LH methods perform similarly to our proposed weight-based metadata partitioning method. This is because we know the entire set of metadata in advance and carefully design the MLT for both DSP and LH methods.

Even so, our proposed metadata partitioning method can still perform the best in terms of workload balancing. As shown in Figure 6, the workload distribution of WPAR is the closest to the theoretically optimal workload distribution (grey dash line in the figure), due to the fact that the weights of MDSs are inherent in the design of our WPAR scheme. The experimental results also indicate that our design is more adaptive by given an arbitrary workload trace.

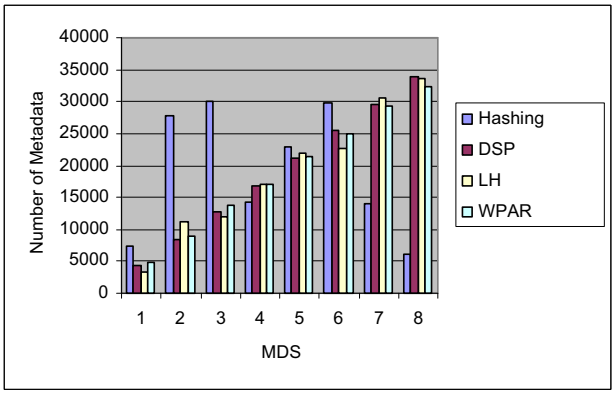


Figure 5. Metadata partitioning in a 8-nodes MDS cluster

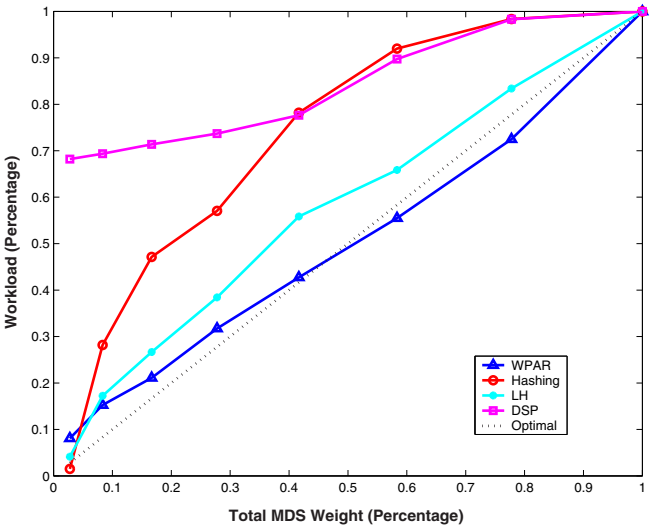


Figure 6. Workload distribution in a 8-node MDS cluster

3.3. B+-tree Reconstruction

In a large-scale Object-based network storage system, it is very likely that some of the metadata servers may be failed or old enough to be retired, and some new servers may be added into the cluster to satisfy new system requirements. In this paper, we use a global storage pool to minimize the metadata movement during adding or removing MDSs. However, when MDS cluster resizing happens, some existing keys may be deleted and some new keys may be inserted into our variant B+-tree, resulting in B+-tree reconstruction. After the B+-tree reconstruction, the universal hashing function for each node may change as well, due to the weight re-distribution. A given LUN may therefore be assigned to another MDS. In a real system, this means that a LUN will be un-mounted from one MDS and mounted to another. Realize that due to the large amount of LUNs, it may cause a "mount storm" in the system, as every server is busy in mounting or un-mounting LUNs. We believe that the Lazy policy or Deferred Update algorithm proposed in [5, 29] can solve this problem, providing that there is some signature information in each LUN to indicate which MDS that currently manages it. As a result, a LUN will not be shifted from one MDS to another during the B+-tree reconstruction, until the LUN is requested.

It is worth to note that by exchanging heartbeat messages, the MDS cluster resizing can be propagated to the entire system. The B+-tree reconstruction process can therefore be carried out in parallel and automatically, unlike LH and DSP that require manually reconfiguring the MLT.

4. Adaptive Metadata Replication Scheme

Without adaptive replication schemes, no matter how the matadata are partitioned in the MDS cluster, hotspots can not be avoided if a large number of requests for the same metadata arrive simultaneously and all the clients know the location of the metadata. In this section, we present our window-based adaptive metadata replication scheme that can dynamically create and delete metadata replicas when a changing workload arrives at the MDS cluster. Our objective is to further balance the workload on top of our weight-based metadata partition strategy (in Section 3) and minimize the hotspots.

Each metadata will be associated with a FIFO type request window with size of k bits, and a list of MDSs that have replications of the metadata, as additional attributes to the corresponding file. A "0" in a bit will be inserted into the request window when a metadata read request (e.g., GET ATTRIBUTES request) arriving and a "1" represents a metadata update request (e.g., SET ATTRIBUTES request), as illustrated in Figure 7. The initial values of the k bits are all set to "1".

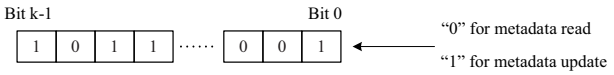


Figure 7. Illustration of a request window

Each request window is associated with two thresholds, T_r and T_d. When the number of read requests in the window reaches T_r, the corresponding MDS (or primary MDS) will then choose another MDS, which is not in the replication

list and has the highest available weight[2], to replicate (or cache) the metadata. After replication, the number of read requests in the window will be reset to $\frac{N_{read} \cdot N_R}{N_R+1}$, where N_{read} is the number of read requests in the window and N_R is the total number of replications (including the one in the primary MDS), before the new replica is created. It should be noticed that only the primary MDS will make the decision on whether a new replica should be created or not.

On the other hand, when the number of read requests in the window falls below T_d, the primary MDS will then delete the replica in the MDS which is in the replication list (excluding the primary MDS) and has the lowest available weight. After deletion, the number of read requests in the window will be reset to $\frac{N_{read} \cdot N_R}{N_R-1}$, to reflect the workload redistribution. Further, a replica may also be deleted by some MDS in the replication list due to its own cache replacement scheme. In this case, message will be passed back to the primary MDS to update the replication list and reset the window accordingly.

Figure 8 illustrates how the clients will now access the metadata with our adaptive metadata replication scheme.

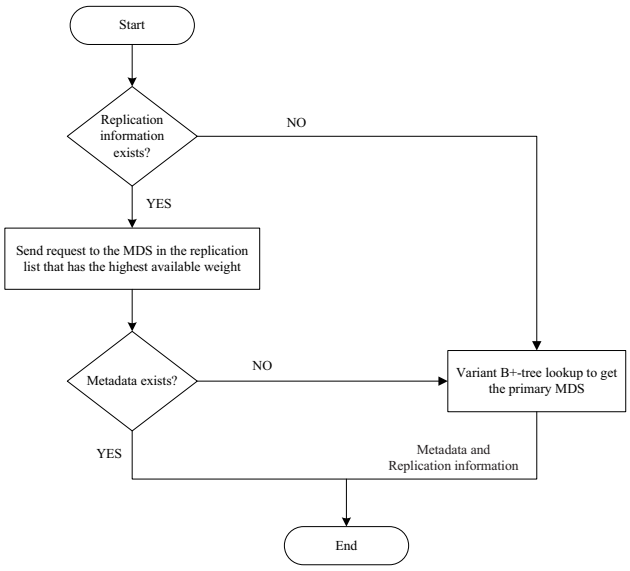


Figure 8. Metadata lookup with replications

In order to access a metadata, if there is no corresponding replication information available in the local cache, a client will follow the same metadata lookup process as described in Figure 4, by looking up the variant B+-tree and sending request to the primary MDS. The primary MDS will then send back the requested metadata together with the replication information. On the other hand, if the corresponding metadata replication information is available in its local cache since previous metadata access, the client can now select a MDS in the replication list that has the highest available weight to access the metadata. However, the requested metadata may not exist in the selected MDS due to replica deletion (deleted by the primary MDS or local cache replacement scheme), in which case the client will again go back to lookup the variant B+-tree to get the metadata and updated replication information.

In addition, we adopt a *read-one-write-all* mechanism, i.e., a metadata can be read from any of the replications, whereas a metadata update has to be propagated to all the replications in the MDS cluster, for strong data consistency. Therefore, when servicing a metadata update request, the selected MDS will take the responsibility to propagate the request to all the MDSs in the replication list.

[2]In terms of current free cache size, network bandwidth, and CPU power, which can be updated periodly through the heartbeat message

5. Conclusions

In this paper, we have proposed a new metadata management strategy for petabyte-scale object storage systems, called WPAR (Weighted Partitioning and Adaptive Replication), that can efficiently distribute the metadata within the MDS cluster based on the weight of each MDS, and dynamically generate or delete metadata replications within the MDS cluster to minimize the hotspot problems. Experimental results show that our weight-based metadata partitioning method is more adaptive and can outperform several existing metadata partitioning methods in terms of workload balancing in a heterogeneous environment.

Currently, we are implementing the WPAR design in our iSCSI-based OSD prototype to further explore several issues that are not addressed in this paper. Firstly, more experiments are required to determine the values of k, t_r, and t_d in order to optimize the performance of our adaptive metadata replication scheme. Secondly, we would like to further evaluate the performance of our WPAR design, as a whole, in terms of throughput, number of requests per metadata access (or metadata request response time), etc. Lastly, competitive analysis [26] will always create an interesting research direction to analyze the performance of our adaptive metadata replication scheme from theoretical angle.

Acknowledgment

This work was done when the first author worked at Data Storage Institute, Singapore. The authors would like to acknowledge the support under the project “Design, Implementation and Management of a High Performance Object-Based Storage and Retrieval System for Large Scale Multimedia Applications”, grant R-263-000-345-305 by A*STAR (Agency for Science, Technology and Research), Singapore.

References

[1] A.Azagury, R.Canetti, M.Factor, S.Halevi, E.Henis, D.Naor, N.Rinetzky, O.Rodeh, and J.Satran, "A Two Layered Approach for Securing an Object Store Network", in the Proceedings of the *First International IEEE Security in Storage Workshop*, pp.10-23, 2002.

[2] A.Azagury, V.Dreizin, M.Factor, E.Henis, D.Naor, N.Rinetzky, O.Rodeh, J.Satran, A.Tavory, and L.Yerushalmi, "Towards an object store", *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pp.165-176, 2003.

[3] P.J.Braam, "The Lustre Storage Architecture", White paper, Cluster File Systems, Inc., 2004.

[4] P. Bernstein, V. Hadzilacos, and N. Goodman, "Concurrency Control and Recovery in Database Systems", *Addison Wesley, Massachusetts*, 1987.

[5] Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Lan Xue, "Efficient Metadata Management in large Distributed Storage Systems", *20th IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems and Technologies*, 2003.

[6] T.H.Cormen, C.E.Leiserson, R.L.Rivest and C.Stein, "Introduction to Algorithms", *MIT Press*, September 2001.

[7] M.D.Dahlin, C.J.Mather, R.Y.Yang, T.E.Anderson, and D.A.Patterson, "A Quantitative Analysis of Cache Policies for Scalable Network File Systems", *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp.150-160, 1994.

[8] M.Factor, D.Nagle, D.Naor, E.Riedel, J.Satran, "The OSD Security Protocol", In the Proceedings of the *Third IEEE International Security in Storage Workshop*, pp.29-39, 2005.

[9] D.E.Knuth, "Sorting and Searching", V(3), *The Art of Computer Programming, Addison-Wesley*, 1973.

[10] K.KleinOsowski, T.Ruwart, D.J.Lilja, "Communicating Quality of Service Requirements to an Object-Based Storage Device", in the Proceedings of the *22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies*, pp.224-231, 2005.

[11] Y.Lu, D.H.C.Du, T.Ruwart, "Qos Provisioning Framework for an OSD-based Storage System", In the Proceedings of *22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies*, pp.28-35, 2005.

[12] M.Mesnier, G.R.Ganger, E.Riedel, "Object-Based Storage", *IEEE Communications Magazine*, 41(8), pp.84-90, 2003.

[13] J.Menon, D.A. Pease, R.Rees, L.Duyanovich, and B.Hillsberg, "IBM Storage Tank - A Heterogeneous Scalable SAN File System", *IBM Systems Journal*, 42(2), 2003.

[14] J.K.Ousterhout, H.D.Costa, D.Harrison, J.A.kunze, M.Kupfer, and J.G.Thompson, "A Trace-driven Analysis of the Unix 4.2 BSD File System", *10th ACM Symposium of Operating Systems Principle*, pp.15-24, 1985.

[15] J.K.Ousterhout, A.R.Cherenson, F.Douglis, M.N.Nelson, and B.B.Welch, "The Sprite Network Operating System", *IEEE Computer*, 21(2), pp.23C36, Feb. 1988.

[16] "SCSI Object-based Storage Device Commands-2 (OSD-2)", T10 Committee, Storage Networking Industry Association (SNIA), Jan. 2007.

[17] B.Pawlowski, C.Juszczak, P.Staubach, C.Smith, D.Lebel, and D. Hitz, "NFS version 3: Design and Implementation", In Proceedings of the *Summer 1994 USENIX Technical Conference*, pp.137C151, 1994.

[18] D.P.Reed, "Implementing Atomic Actions on Decentralized Data", *ACM Trans. on Computer Systems*, 1(1), pp.3-23, 1983.

[19] D. Roselli, J. Lorch, and T.Anderson, "A Comparison of File System Workloads", *USENIX Annual Technical Conference*, pp.41-54, 2000.

[20] O. Rodeh and A. Teperman, "zFS - A Scalable Distributed File System using Object Disks", *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pp.207-218, 2003.

[21] J. Satran and A. Teperman, "Object Store Based SAN File Systems", *International Symposium of Santa Caterina on Challenges in the Internet and Interdisciplinary Research*, 2004.

[22] Thomas M. Ruwart, "OSD: A Tutorial on Object Storage Devices", *Advanced Concepts Ciprico*, 2002.

[23] A. Teperman, and A.weit, "Improving Performance of Distributed File System Using OSDs and Cooperative Cache", IBM, 2004

[24] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Kristal T. Pollack, "Intelligent Metadata Management for a Petabyte-Scale File System", *2nd Intelligent Storage Workshop*, University of Minnesota, May 2004.

[25] L.Wujuan and V.Bharadwaj, “An Adaptive Object Allocation and Replication Algorithm in Distributed Databases”, In the Proceedings of the *23rd IEEE International Conference on Distributed Computing Systems Workshops*, pp.132-137, 2003.

[26] L.Wujuan and V.Bharadwaj, “Object Management in Distributed Database Systems for Stationary and Mobile Computing Environments: A Competitive Approach”, *Kluwer Academic Publishers*, 2003.

[27] Sage A. Weil, Kristal T. Pllack, Scott A. Brandt, and Ethan, L. Miller, “Dynamic Metadata Management for Petabyte-scale File Systems”, Proceedings of the *ACM/IEEE Conference on Supercomputing*,2004.

[28] “XAM specification v1.2”, Fixed Content Aware Storage, Storage Networking Industry Association (SNIA), 2005.

[29] J.Yan, Y.Zhu, H.Xiong, R.Kanagavelu, F.Zhou, and S.L.Weon, “A Design of Metadata Server Cluster in Large Distributed Object-based Storage”, *21th IEEE Conference on Mass Storage Systems and Technologies*, pp.100-106, 2004.