Performance Evaluation of A Load Self-Balancing Method for Heterogeneous Metadata Server Cluster Using Trace-Driven and Synthetic Workload Simulation

Bin Cai[1], Changsheng Xie[1], Guangxi Zhu[2]

[1]Department of Computer Science and Technology, Huazhong University of Science and Technology. Wuhan National Laboratory for Optoelectronics, Wuhan, P.R. China, 430074
{hust_caibin@sohu.com, csxie@263.net}

[2]Department of Electronics and Information Engineering, Huazhong University of Science and Technology. Wuhan National Laboratory for Optoelectronics, Wuhan, P.R. China, 430074
gxzhu@mail.hust.edu.cn

Abstract

In cluster-based storage systems, the metadata server cluster must be able to adaptively distribute responsibility for metadata to maintain high system performance and long-term load balance, due to workload skew and metadata servers' heterogeneity. In this paper, we describe a simple and adaptive metadata load management scheme, called Self-Balancing Uniform (SBU) randomization, to efficiently and continually adapt the metadata distribution to current demands in heterogeneous metadata server cluster. We implement our system within a discrete event driven simulation environment, along with two other systems, simple randomization (SR) and performance aware distribution (PAD) to serve as points of comparison, and evaluate the performance of our SBU algorithms against SR and PAD algorithms by both a trace workload and a synthetic workload. Simulation results verify that our SBU algorithm achieves load self-balance, provides consistent response latencies and resource utilization. Simulation results also indicate that SR cannot cope with skew and heterogeneity and PAD requires a larger shared state to achieve optimal performance.

1. Introduction

Cluster-based storage is a promising alternative to today's traditional distributed file system and monolithic storage systems (*e.g.*, [1], [3], [4], [5], [6]). The concept is that collections of smaller, lower-performance, less-reliable and commodity storage-nodes (sometimes referred to as storage bricks) should be able to provide performance and reliability competitive with today's high-end solutions, but at much lower cost and with greater scalability. As with previous arguments for RAID [9] and cluster computing, the case for cluster-based storage anticipates that high levels of reliability and performance can be obtained by appropriate redundancy and workload distribution across storage nodes [2], [7], [8]. Generally, the architecture for large-scale cluster-based storage consists of many metadata servers (MDS cluster), thousands of storage nodes (storage node cluster) and potentially hundreds of thousands of clients. Applications of such a system will include scientific computing environments, the Internet Archive, and large data centers.

Although the size of metadata are relatively small compared to the overall volume of the system, a study on the file system traces collected in different environments over a course of several months shows that requests targeting at the metadata can account for up to 83% of the total number of I/O requests [12]. As the character of the metadata workload may change over time, the MDS cluster must be able to continually adapt to current demands by dynamically repartitioning workload to maintain high system performance and long-term load balance. Moreover, the increasing heterogeneity of MDS cluster makes this problem more acute. Under such skew and heterogeneity, an efficient metadata load management is important to overcome the severe limitation of the throughput of metadata operations as the number of files or I/O requests increases.

In this paper, we describe an adaptive metadata load management scheme to efficiently distribute responsibility

This research is supported by National 973 Great Research Project of P.R China under the grant No. 2004CB318200, National Natural Science Foundation under grant No. 60273037 and No. 60303031, and Huazhong University of Science and Technology Postdoctoral Special Foundation.

for metadata across a heterogeneous MDS cluster. We utilize a simple dynamic workload partition scheme, called Self-Balancing Uniform (SBU) randomization that is derived from the hashing technique described by Czumaj *et al* [13], to continually adapt the metadata distribution to current demands without prior knowledge about each individual MDS capacity or clients' behavior. SBU randomized processes underlying load balancing is based on the multiple-choice paradigm and uses two independent random hash functions to determine the primary location and alternative location of metadata items in MDS cluster. It is a simple, memoryless, and local search-based algorithm that can evenly distribute the tasks of metadata management onto a group of heterogeneous MDSs and can balance the workload of the MDSs in the system as much as this possible. The non- trivial property of SBU scheme is that it needs only $\Delta \bullet n^{O(1)}$ steps to reach a perfect load distribution with high probability no matter with which load distribution the system starts with the imbalance Δ, and will always converge to a best possible load distribution.

Finally, we evaluate competing load management strategies by simulation on the basis of overall system performance and adaptation to workloads that evolve over time. Simulation results demonstrate that SBU performs comparably to an optimal system and provides consistent performance for clients' metadata workload on any MDS in heterogeneous MDS cluster with an acceptable degree of load migration.

The rest of this paper is organized as follows. In Section2 we summary related work, and Section 3 outlines our system architecture, following which the details of our SBU algorithm are presented in Section 4. Section 5 evaluates trace-driven and synthetic simulation experiments on our system and shows the results, and the conclusion comes at Section 6.

2. Related Work

Traditional network file systems have partitioned workload and storage by statically assigning portions of the directory hierarchy to different file servers; such as NFS [14], AFS [15], Coda [16], Sprite [17]. This static partitioning technology typically requires a system administrator to decide how the file system should be distributed and manually assign subtrees of the hierarchy to individual file servers. However, a statically partitioned cluster can not accommodate file system expansion in a usual fashion, requiring manual redistribution of the hierarchy to accommodate new data or even increased client demand for existing data. Furthermore, if client workload is not evenly distributed across all file data, static partitioning is vulnerable to imbalance as individual servers can be overloaded by "hot spots" of popularity in certain parts of the hierarchy [18].

Randomization is a powerful technique for load management in clusters and distributed systems [19], [20]. Simple hash-based randomized load management schemes balance load effectively in homogeneous environments and incur very small overhead and also provide an efficient addressing scheme, which makes them appealing to traditional clusters and distributed systems. Vesta [21], Intermezzo [22], RAMA [23], zFS [24], PVFS[10], GoogleFS [4] and Lustre [11], [25] all hash the file pathname and/or some other unique identifier to determine the location of metadata and/or data. Because hashing is deterministic and requires no I/O operations, clients can locate and contact the responsible MDS directly and, for average workloads and well-behaved hash functions, requests are evenly distributed across the cluster. Further, hot-spots of activity in the hierarchical directory structure, such as heavy creation of activity in a single directory, do not correlate to individual metadata servers because metadata location has no relation to the directory hierarchy. Although simple randomization performs well in certain environments, it can not support extreme workload skew and server heterogeneity [20].

Many researches have been done in the area of load management in clusters and distributed systems. A number of dynamic load management techniques are designed for parallel systems and homogeneous clusters [33], [34]. Workload is transferred from heavily loaded servers to lightly loaded ones. Another family of techniques [35], [36] takes into account server heterogeneity but require all servers to periodically broadcast load and available capacity.

3. Architecture and Objectives

Figure 1 shows the architecture of a generic cluster-based storage system where a number of storage nodes are connected by a high-bandwidth low-latency switched network.

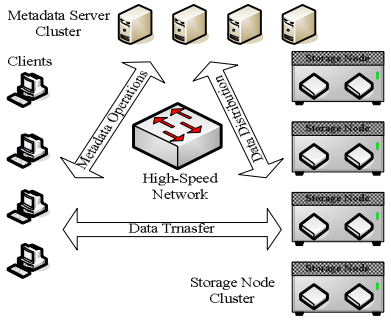


Figure 1. Cluster-based storage architecture

Each node has its own storage devices. There are no functional differences between all cluster nodes. The role of clients, metadata servers, and storage nodes can be carried out by any node and a node may not be dedicated to a specific role. It can act in multiple roles

simultaneously. File data is striped across a large number of storage nodes to maximize I/O throughput and data safety. Unlike data storage in traditional file systems, which typically seek to group related files together, data is distributed to storage nodes based on a deterministic pseudo-random algorithm that guarantees a probabilistically balanced distribution of data throughout the system [26]. Ultimately all metadata must be stored on some sort of permanent disk storage. Metadata for a petabyte file system that may contain more than a billion files might consume a terabyte or more of disk space [18]. This is likely to be too large to reside completely in the collective RAM of the metadata server cluster. Ideally, the MDS memory caches will satisfy most reads, but they will periodically need to go to disk to retrieve requested information, and all updates must be saved to a stable store such as disk.

The objective of this work is to provide a simple and efficient metadata load management scheme for heterogeneous MDS clusters. The scheme aims to evenly distribute the metadata workload and maintain consistent performance for applications without sacrificing overall throughput. It also operates without prior knowledge of heterogeneity, automatically adapts to workload changes, and maintains acceptable load movement during rebalancing due to migration cost.

In this study, we focus on the aspect of metadata load management, and some other important issues, such as management of the striping of file contents, consistency maintenance, concurrency accesses control, file system security and protection enforcement, and incorporation of fault tolerance, are beyond the scope of this paper.

4. SBU Algorithm

The basic idea behind the hashing technique is to use a compact function (the hash function) in order to map some space *U* onto some space *V*. Hash-based randomization is a powerful technique used in clusters and distributed systems for load management. It offers uniform distribution, efficient addressing, little shared state, and scalability.

The study of balls-into-bins games or occupancy problems has a long history. In general, the goal of a balls-and-bins algorithm is to assign a set of independent objects (*e.g.*, task, jobs, and data blocks) to a set of resources (*e.g.*, processor, servers, and disks) so that the load is distributed among the bins as evenly as possible. Previously, in the single-choice paradigm, each ball is placed into a bin chosen independently and uniformly at random. For the case of n bins and m balls ($m \geq n \log n$), it is well known that a simple process that places the balls one by one in the least loaded bin can achieve a maximum load of $m/n + \log\log n + \Theta(1)$ with high probability. A drawback of applying this approach to metadata load balance is that is requires prior and global knowledge about each individual MDS capacity and locations of all metadata, which is far too space consuming if the files stored in the system grow continuously. This also makes the algorithm difficult for implementations in distributed or parallel systems.

We use a technique called Self-Balancing Uniform (SBU) randomization to place and balance load. SBU Randomization is based on the hashing technique described by Czumaj *et al* [13]. SBU randomized processes underlying load balancing is based on the multiple-choice paradigm: m balls have to be placed in n bins, and each ball can be placed into one out of 2 randomly selected bins.

Instead of using a single random hash to hash the unique file identifier (the unique name of a file is specific to clusters, such as an inode number, a pathname or content fingerprint) to determine the location of metadata (*e.g.*, Vesta [21], Intermezzo [22], RAMA [23], zFS [24], PVFS[10], GoogleFS [4] and Lustre [11], [25]), SBU algorithm uses two independent random hash functions *h1* and *h2*. Hash function *h1* determines the primary MDS in cluster where the metadata is stored, and hash function *h2* is used to locate the alternative MDS. Each MDS monitors its performance and produces a performance metric over a chosen time interval. Naturally, we use latency as the performance metric because the metadata workload consists of little data and short-lived transactions. At the end of each interval, each MDS computes its latency in the past interval and reports it to an elected delegate server. The delegate server examines all latencies and comes up with workload distribution. The delegate is designed to be stateless and determines the new load configuration based solely on reported latencies. If the delegate fails, the next elected delegate runs the same protocol with the same information.

Suppose that initially all the metadata has chosen their locations in MDS cluster $\{1, 2, 3, \cdots, n\}$ and each metadata is arbitrarily placed in one of its two locations. For each metadata item *m*, SBU randomization checks the number of metadata items in the MDSs using two hash function *h1(m)* and *h2(m)*, and places *m* in the least loaded of them, and repeats the following Self-Balancing step, described at figure 2.

SBU Algorithm is a simple, memoryless and local search algorithm. Although it is a local search approach, it never arrives at a deadlock situation, in which the balancing may be far away from optimal and no re-balancing progress is possible. The property of SBU algorithm is that no matter with which state (i.e., assignment of metadata items to MDSs) it starts, then in time $O(m) + n^{O(1)}$ it will converge to a state in which the

maximum load of any MDS in cluster is upper bounded by $\lceil m/n \rceil$ with high probability.

Hence, two random hash functions can in principle evenly distribute any tasks of metadata management almost perfectly among the heterogeneous MDS cluster and can balance the workload of the MDSs in the system as much as this possible.. Hash functions that have near-random qualities in practice are, for example, cryptographic hash functions such as SHA-1 [27].

```
suppose that the metadata items arrive one by one;

for each metadata item x, randomly determines primary MDS S1
and alternatvie MDS S2, in other words, S1=h1(x), S2=h2(x);

If (there is a metadata m with h1(m)=S1 and h2(m)=S2)
{
        pick any m;

        place m into the least loaded MDS (among S1 and S2);

        If (there is tie, i.e., S1 without m has the same load as S2)
        {
                place m into a randomly chosen of the two MDSs;
        }
 }
```

Figure 2. Self-balancing step

5. Performance Evaluation

To validate the performance characteristics of our load management system based on SBU randomization, we have implemented our system within a discrete event driven simulation environment, along with two other systems: *simple randomization (SR)* and *performance aware distribution (PAD)* to serve as points of comparison, and evaluate the performance of our SBU algorithms against SR and PAD algorithms by both a trace workload and a synthetic workload.

SR employs a pseudo-random hash function to uniformly assign workload to MDS; and PAD perfectly knows the processing capabilities of each individual MDS and realizes the optimal load balance through identifying the permutation of workload onto MDS that minimizes average response latency of each MDS. For this reason, SR allows us to compare our system with static, offline randomized policies used in heterogeneous clusters, while PAD provides the upper bound of load balancing characteristics.

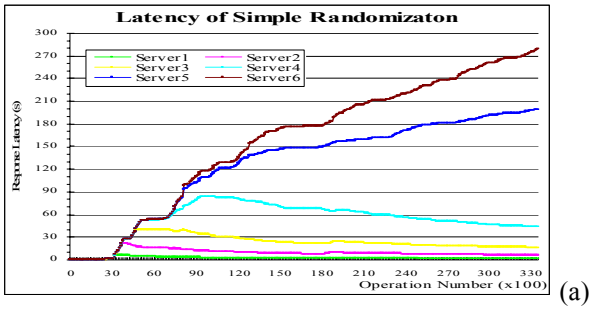
Simulation results verify that our SBU algorithm achieves load balance among heterogeneous MDS cluster, provides consistent response latencies for each client's metadata requests and resource utilization with an acceptable degree of load movement. Simulation results also indicate that SR cannot cope with skew and heterogeneity and PAD requires a larger shared state to achieve optimal performance.

5.1. Simulation Setup and Workload

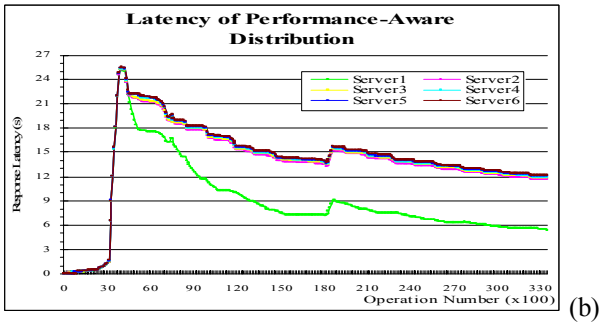
The focus of our simulation efforts is on MDS behavior and workload movement, and not on underlying disk storage behavior. Since a significant body of research has investigated the use of accurate disk simulation for storage system evaluation [29], the distributed metadata management systems we are evaluating can exist on any underlying disk subsystem. For this reason, we simplify the storage simulation to reflect the response latency and resource utilization of each individual MDS, and workload movement during a specified interval of metadata requests only. The simulator models a MDS cluster and uses First-Come-First Served (FCFS) queuing discipline to serve each MDS workload.

We use the DFSTrace [28] to drive our experiments. DFSTrace data were collected on about 30 different workstations running different file systems such as NFS [14], AFS [15], Coda [16], and the local Unix file system. However, DFSTrace data has some shortcomings in driving our simulation, such as being collected on legacy hardware with limited heterogeneity. The data provides limited ability to explore the performance of our system because it represents a fixed hardware configuration. To get around the limitations of DFSTrace, we perform experiments driven by a synthetic workload as well. Synthetic workload explores different workload skew with DFSTrace. This allows us to experiment with an arbitrary amount of heterogeneity and helps to understand our system's characteristics under different hardware/workload configurations deeply. We use DFSTrace workload results for comparison with synthetic workloads to ensure the sanity of our results.

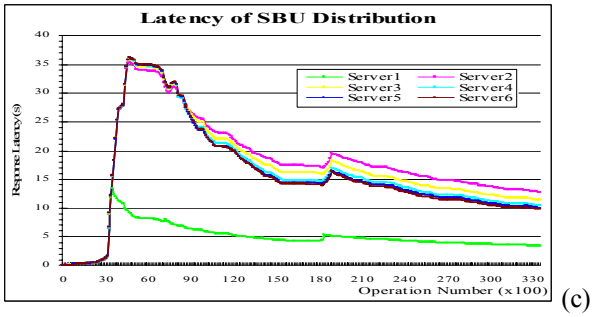
We run simulations based on a 6 hours DFSTrace to test the performance characteristics of our system and the other two systems described above. There are 33,540 metadata requests in total. Based on the number of requests, we choose to simulate a heterogeneous MDS cluster with 6 MDSs. Server 1, 2, 3, 4, 5 and 6 have processing power 11, 9, 7, 5, 3 and 1 respectively. More specifically, for the same workload, if the most powerful server in our simulated MDS cluster (server 1) consumes time T to complete a metadata request, then the least powerful server (server 6) consumes time $9T$.



(a)

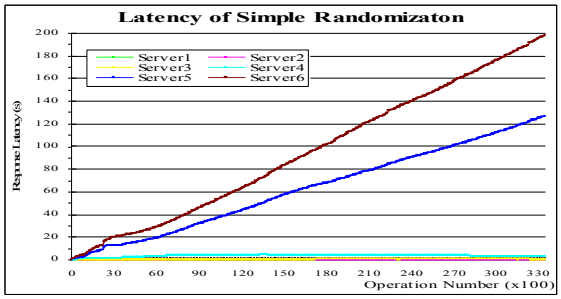


(b)

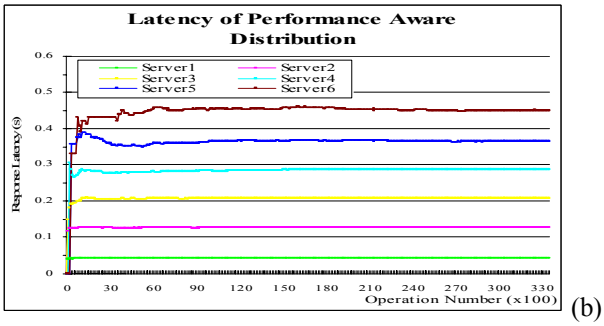


(c)

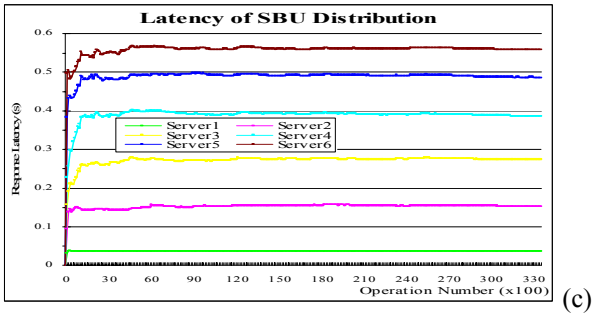
Figure 3. MDS latency for DFSTrace workload



(a)



(b)



(c)

Figure 4. MDS latency for synthetic workload

5.2. Performance Comparison

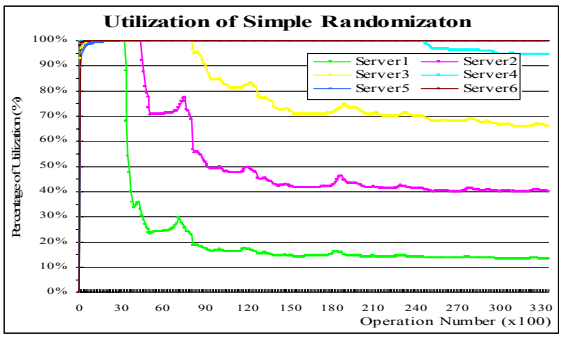
The performance comparison is done in two parts. First, we compare the response latency and resource utilization of each individual MDS in the three load management systems to investigate the performance characteristics of them and to verify that our system only takes some time to discover the heterogeneous configuration in MDS cluster and converge to the balancing workload distribution. Second, we show a close-up comparison of our system and PAD system in some aggregate metrics to understand some load balance performance details, such as average latency and requests distribution, and to illustrate that the performance of our system and PAD system is comparable.

Figure 3 shows the response latency of 6 MDSs that serve metadata requests over DFSTrace with the number of metadata operation increases. SR performs poorly because it is static algorithms. It has no knowledge about MDS or workload heterogeneity and cannot respond to skew when it occurs. Over the simulation, with the number of metadata operation grows, the response latencies of server 4 and 5 increase, even though the more powerful MDSs (server 1, 2, 3 and 4) have unused capacity.

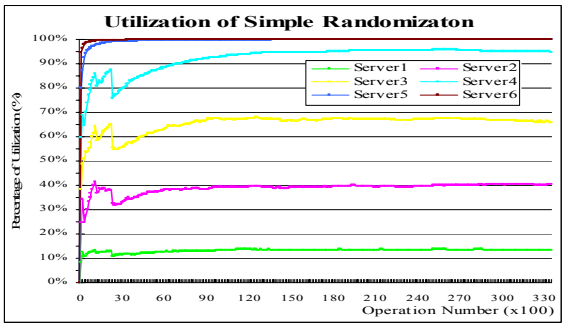
The PAD and SBU algorithms balance load over the course of the experiment. Because of having perfect knowledge about each MDS capacity, the PAD algorithm performs in a load-balanced state at the very beginning, identifies the best load distribution before the workload occurs and configures the MDS to best handle the workload. SBU algorithm has no prior knowledge and, therefore, assumes initially that all workload and all MDSs are uniform. Then, it adapts to workload and heterogeneity of each MDS capacity, and reaches a self-balance state.

Both PAD and SBU policies show increases in latency on the most powerful MDS under heavy load (the first 4000 operations and the operations between 18000-19000). The bursts of workload occur in a few operations and both algorithms map those bursts to the most powerful

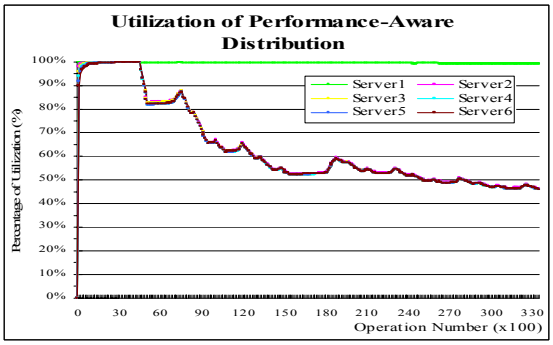
servers. PAD algorithm does so more effectively because it has perfect knowledge of each MDS's capability, and can move operations to any MDS to get the best fit between workload and server capabilities. Even though SBU algorithm has no prior knowledge, it does perform comparably. After experiencing response latencies increase on less powerful servers (server 5 and 6), SUB enlists the next more powerful server (server1, 2, 3 and 4) in the next operation step. As a result, our system takes some time to discover heterogeneous configuration of MDS cluster, and converge to the balanced workload distribution.



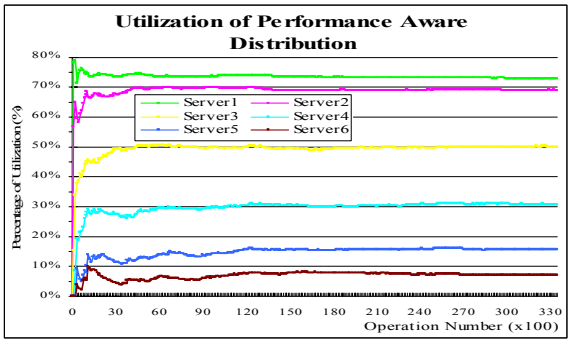
(a)



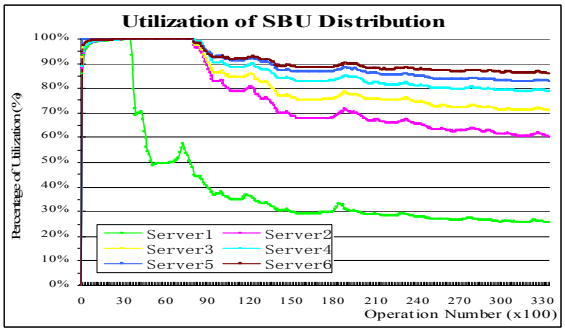
(a)



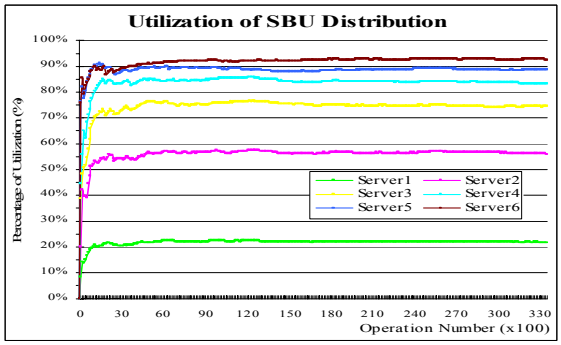
(b)



(b)



(c)



(c)

Figure 5. Utilization for DFSTrace workload

Figure 6. Utilization for synthetic workload

To better understand the impact of our system under extreme workload skew, we conduct experiments using a synthetic workload that is generated using the same parameters as the DFSTrace. In other words, the synthetic workload consists of 33,540 client requests during a period of 6 hours. The request inter-arrival times are governed by a Pareto distribution [20], [30], [31], [32]. As shown in Figure 4, the experiments driven by the synthetic workload follow our results found from DFSTrace simulation. Again, we observe that SR policy cannot cope with workload and MDS heterogeneity, while SBU and PAD are comparable.

The resource utilization of 6 MDSs under DFSTrace and the synthetic workload are illustrated in Figure 5 and Figure 6, respectively. SR system is heavily unbalanced. Its uniform randomization makes more powerful MDS idle and less powerful MDS busy. Hence, it wastes the processing resources and brings bottleneck at the weaker MDS. PAD system balances the resource utilization among MDSs effectively and makes the most powerful

MDS (server 1) always busy, but it needs prior knowledge about each MDS capacity to fit MDS to current workload. Our SBU system makes a tradeoff between these two policies by adaptively assigning workload to more powerful MDS. Although it does not utilize the overall resource of the most powerful MDS, it indeed mitigates the load of less powerful MDS and alleviates the bottleneck.

5.3. Dynamic Workload Migration

It is very costly to move metadata from one server to another. The server that transfers metadata needs to flush its cache, writing all dirty data to disk, while the server that receives metadata must initialize the metadata with a cold cache, which hinders initial performance. Figure 7 shows the number of requests movement during the course of DFSTrace simulation with divided into 3 minutes interval. We observer that most activities of workload movement are upper bounded by 70 and the average number of request movement every 3 minutes interval approximates to 100.

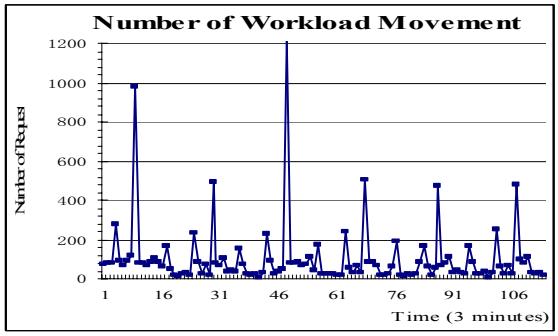


Figure 7. The number of workload movement during DFSTrace simulation

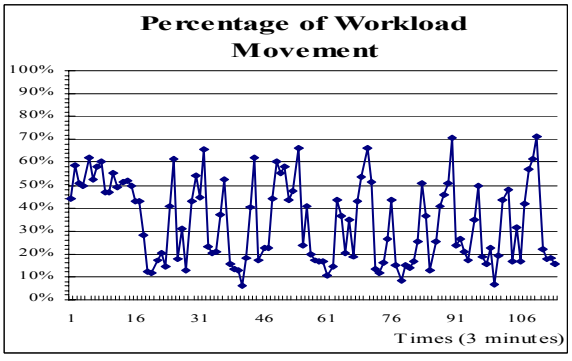


Figure 8. The percentage of workload movement during DFSTrace simulation

Figure 8 illustrates the percentage of total workload that has been moved by SBU algorithm during the same experiment. This result reveals that the percentage of most movement activities is around 20% and the very few peaks of movement curve are about 70%. The average percentage of workload movement every 3 minutes interval is around 34%. These experimental results demonstrate that our system can provide consistent performance with the workload movement falling into an acceptable range.

6. Conclusion

In this study, we focus on the aspect of metadata load management and describe an adaptive metadata load management scheme to efficiently distribute responsibility for metadata across a heterogeneous MDS cluster. We utilize a dynamic workload partition scheme, called Self-Balancing Uniform (SBU) randomization, to continually adapt the metadata distribution to current demands without prior knowledge about each individual MDS capacity or application behavior. SBU is a simple, memoryless, local search algorithm that can evenly distribute the tasks of metadata management onto a group of heterogeneous MDSs and can balance the workload of the MDSs in the system as much as this possible. SBU randomized processes underlying load balancing is based on the multiple-choice paradigm and uses two independent random hash functions to determine the primary location and alternative location of metadata items in MDS cluster. SBU randomization has the non-trivial property that no matter with which state the system starts, it will always converge to a state in which the maximum load is optimally small and does not arrives at a "dead-lock" situation.

Simulation results demonstrate that our load management system based on SBU randomization technique addresses several performance issues in heterogeneous metadata server cluster, deals with heterogeneity in both server and workload, and performs comparably to an optimal system. The results also reveals that SBU randomization maintains performance consistency, comes up with an acceptable degree of load migration, and provides all the load balancing benefits of optimal systems with less shared state.

References

[1] T. E. Anderson, M. D. Dahlin, J. M. Neefe, et al., "Serverless Network File System", ACM Transactions on Computer Systems Special Issue on Operating System Principles, 14(1): 41-79, Feb, 1996.

[2] Y. Saito, S. Frolund, A. Veitch, et al., "FAB: Building Distributed Enterprise Disk Arrays from Commodity Components", ACM SIGARCH Computer Architecture News, 32(5): 48-58, Dec, 2004.

[3] G. R. Ganger, et al., "Self-* Storage: Brick-Based Storage with Automated Administration", Technical Report CMU-CS-03-178, Carnegie Mellon University, Aug, 2003.

[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System", ACM Symposium on Operating

System Principles (SOSP 2003), New York, USA, Oct. 2003.

[5] IBM Almaden Research Center, "Collective Intelligent Bricks", Oct, 2005. http://www.almaden.ibm.com/StorageSystems/autonomic_storage/CIB/index.shtml.

[6] E. K. Lee and C. A. Thekkath, "Petal: Distributed Virtual Disks", Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Cambridge, Massachusetts, USA, pp: 84-92, Oct, 1996.

[7] Z. Zhang, S. D. Lin, Q. Lian, et al., "RepStore: A Self-Managing and Self-Tuning Storage Backend with Smart Bricks", Proceedings of 2004 Autonomic Computing International Conference, May, 2004.

[8] Hong Tang, Aziz Gulbeden, Jingyu Zhou, et al., "Sorrento: A Self-Organizing Storage Cluster for Parallel Data-Intensive Applications", Proceedings of International Conference on High Performance Computing, Networking and Storage, Pittsburgh PA, Nov. 2004.

[9] D. A. Patterson, et al., "A case for redundant arrays of inexpensive disks (RAID)", ACM SIGMOD International Conference on Management of Data, 1988.

[10] P. Carns, W. Ligon III, R. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," In Proc. of 4th Annual Linux Showcase and Conference, 2000.

[11] Cluster File Systems, Inc, "Lustre: A Scalable, High-Performance File System", White Paper, 2003.

[12] J. R. L. Drew Roselli and T. E. Anderson, "A Comparison of File System Workloads", Proceedings of the Annual USENIX Technical Conference, San Diego, California, Jun, 2000.

[13] Artur Czumja, Chris Riley and Christian Scheideler, "Perfectly Balanced Allocation", Proceedings of 7th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM 2003), Aug. 2003.

[14] R. Sandberg, D. Goldberg, S. Klerman, et al., "Design and Implementation of the Sun Network File System", Proceedings of the Summer 1985 USENIX Conference, pp: 119-130, 1985.

[15] J. Howard, M. Kazar, S. Menees, et al., "Scale and Performance in a Distributed File System", ACM Transactions on Computer Systems, 6(1): 51-81, Feb, 1988.

[16] J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System", ACM Transactions on Computer Systems, 10(1): 3-25, Feb, 1992.

[17] J. K. Ousterhout, A. R. Cherenson, F. Douglist, et al., "The Sprite Network Operating System", Computer, 21(2): 23-36, Feb, 1988.

[18] Sage A. Weil, Kristal T. Pollack, Scott A. Brandt, et al., "Dynamic Metadata Management for Petabyte-Scale File Systems", Proceedings of ACM SuperComputing, 2004.

[19] M. Mitzenmacher, "On the Analysis of Randomized Load Balancing Schemes", ACM Symposium on Parallel Algorithms and Architectures, 1997.

[20] Changxun Wu and Randal Burns, "Achieving Performance Consistency in Heterogeneous Clusters", HPDC 2004.

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

[22] P. Braam, M. Callahan and P. Schwan, "The Internezzo File System", Proceedings of the 3rd of the Perl Conference, O'Reilly Open Source Convention, Monterey, CA, USA, Aug, 1999.

[23] E. L. Miller and R. H. Katz, "RAMA: An easy-to-use, High-Performance Parallel File System", Parallel Computing, 23(4): 419-446, 1997.

[24] Rodeh and A. Teperman, "zFS: A Scalable Distributed File System using Object Disks", Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, Apr, 2003.

[25] P. J. Braam, "The Lustre storage architecture", Technical report, Cluster File Systems, Inc., 2002. http://www.lustre.org/docs/lustre.pdf

[26] R. J. Honicky and E. L. Miller, "Replication under Scalable Hahsing: A Family of Algorithms for Scalable Decentralized Data Distribution", Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), Santa Fe, NM, Apr, 2004.

[27] M. J. R. Robshaw, "MD2, MD4, MD5, SHA and Other Hash Functions", RSA Labs., Vol. 4.0, Technical Report, TR-101, 1995.

[28] L. Mummert and M. Satyanarayanan, "Long Term Distributed File Reference Tracing: Implementation and Experience", Technical Report CMU-CS-94-213, Carnegie Mellon University, Nov, 1994.

[29] J. Wilkes, "The pantheon storage-System Simulator", Technical Report HPL-SSP-95-14, Storage Systems Program, Computer Systems Laboratory, Hewlett-Packard Laboratory, Palo Alto, CA, May, 1996.

[30] P. Barford, A. Bestavros, A. Bradley, et al., "Change in Web Client Access Patterns: Characteristics and Caching Implication", World Wide Web, 1999.

[31] P. Barford and M. E. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", Proceedings of the 1998 ACM SIGMETRICS, Jul, 1998.

[32] M. E. Crovella, R. Frangioso and M. Harchol-Balter, "Connection Scheduling in Web Servers", Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems, Boulder, Colorado, USA. Oct, 1999.

[33] D. L. Eager, E. D. Lazowska and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems", IEEE Transactions on Software Engineering, 12(5), 1986.

[34] J. Watts and S. Taylor, "A Practical Approach to Dynamic Load Balancing", IEEE Transactions on Parallel and Distributed Systems, 9(3), 1998.

[35] H. Zhu, T. Yang, Q. Zheng, et al. Adaptive Load Sharing for Clustered Digital Library Servers. International Journal on Digital Libraries, 2(4), pp 25-235, May. 2000.

[36] J. Watts, M. Rieffle and S. Taylor, "Dynamic Management of Heterogeneous Resources", Proceedings of the High Performance Computing Conference: Grand Challenges in Computer Simulation, Apr, 1998.