Efficient storage management for distributed storage system

Myat Pwint Phyu, Ni Lar Thein
University of Computer Studies, Yangon
myatpwint.ucsy@gmail.com, nilarthein@gmail.com

ABSTRACT

As storage systems grow larger and more complex, the traditional block-based file systems cannot satisfy the large workload. More recent distributed file systems have adopted architectures based on object-based storage. This paper presents a framework of efficient storage management for distributed storage system. In object storage side, low-level storage tasks and data distribution must be managed and in metadata server side, we will manage how to scale the metadata. Due to the high space efficiency and fast query response, bloom filters have been widely utilized in recent storage systems. So, we will also utilize bloom-filter based approach to manage metadata by taking the advantages of bloom-filter and the semantic-based scheme will also be used to narrow the managed workload. In this paper, we will neglect the data distribution of object-based storage side.

Keywords: Object-based storage, metadata management, bloom-filter, unstructured data.

1. INTRODUCTION

Today's file systems are not well suited to the long-term storage of massive amounts of unstructured data. File-based storage provides only very basic metadata, limiting management capabilities. Object-based storage is designed to overcome these limitations. Object-based storage offers an innovative approach to storing and managing vast amounts of unstructured data, from medical images to e-mail. Object-based storage allows access to data by means of a unique identifier that helps avoid the need to know the specific location of a data object. Data can be stored with a much richer set of metadata in an object-based model than in a file-based model. Information stored with the object can include the application of retention and deletion policies.

Object-based storage systems take a new approach to storing data. The file data is stored as an object with the application that stores and retrieves data defining objects with object-based storage. This creates new capabilities in dealing with objects that can be exploited by applications and management software. By dealing with objects and not the specific physical placement requirements of block storage systems, the object-based storage system should have some self-management capabilities regarding data placement and access, relieving storage administrators from that task.

The metadata kept about objects is really the key to enabling new capabilities for object-based storage systems. The content of the metadata is both information (attributes) that the storage system adds, such as size, date, access, etc., as well as information that the application includes for use by applications. The metadata server cluster in a system should efficiently maintain file system directory and permission semantics for a variety of workloads. Although the size of metadata is relatively small compared to the size of the system, metadata operations may make up over half of all file system operations [6]. So, the role of metadata management is challenging. As our knowledge, bloom filter is a fast and space-efficient data structure to represent a set. Due to this features, it have been widely utilized in recent storage systems. So, we will also utilized bloom-filter based approach together with a semantic filter to manage metadata in this system.

The rest of this paper is organized as follows. Section 2 discusses the various approaches of metadata storage. Section 3 explains the bloom filter and Section 4 introduces the proposed system. Then we conclude the paper with future work in Section 5.

2. RELATED WORK

In this section, we will survey the metadata storage strategies. A distributed metadata server (MDS) cluster requires that the workload be partitioned among some set of hosts such that the size of the cluster can be scaled to handle

increased client transactions. By using the cluster of MDSs, the system can efficiently cope with extreme workloads and scale well.

According to whether metadata can be dynamically redistributed, previous approaches fall into table-based strategy, static strategy, dynamic strategy and bloom filter-based strategy. Although a fine-grained table allows flexibility in metadata placement, the memory space requirement for this approach makes it unattractive for large-scale storage systems. To reduce the memory space overhead, a coarse-grained table maps a group of files to an MS. The table-based mapping method does not require any metadata migration, but it fails to balance the load.

Sub-tree Partitioning divides the global namespace into sub-trees and gives them to metadata servers. The major disadvantage is that the workload may not be evenly distributed among metadata servers. Hashing provides a better balanced load across metadata servers and eliminates hot-spots consist of popular directories. Metadata update operations in hashing may incur a burst of network overhead because it is a random distribution. To address the metadata update problem, LH uses Lazy Policies to defer and distribute update cost. Performing the update and metadata movement at a later time avoids a sudden burst of network activities between metadata servers [1]. Because of the metadata movement in LH, a new method of Hashing Partition focuses on reducing the cross MDS metadata movement in a clustered design [9]. These static strategies cannot dynamically redistribute the global namespace to get high throughput in the case of a changing workload. When new MDSs are added or existing MDSs are removed, none of these strategies address how to determine the optimal amount of metadata that should be moved. Finally, they can do nothing for hot-spots which are caused by popular individual files.

To cope with the changing workload, Dynamic Sub-tree Partitioning [7] leverages the dynamic load balancing mechanism to dynamically redistribute metadata among metadata servers. It can eliminate bottlenecks caused by hot-spots consisting of individual files by replication. But it cannot reclaim replicas for file metadata which are not popular any more. These cumulate replicas will consume storage space and incur maintenance overhead. So, Dynamic Hashing (DH) [4] is proposed by combining its own three strategies and Lazy Policies borrowed from the Lazy Hybrid (LH) metadata management together to form an adaptive, high-performance and scalable metadata management technique. It introduces strategy to adjust the metadata distribution when the workload changes dynamically, strategy to support the MDS cluster changes and strategy to find hot spots in the file system efficiently and reclaim replicas for these hot spots when necessary.

Hierarchical Bloom Filter Arrays (HBA) [10] maintains two levels of BF arrays, with the one at the top level representing the metadata location of most recently visited files on each metadata server and the one at the lower level maintaining metadata distribution information of all files with lower accuracy in favor of memory efficiency. To incur smaller memory overheads and provide stronger scalability and adaptability, G-HBA [3] organizes MDSs into multiple logic groups and utilizes grouped Bloom filter arrays to efficiently direct a metadata request to its target MDS. In this paper, we will enhance bloom-filter based approach to manage metadata by taking the advantages of bloom-filter and by adding a semantic-based scheme to narrow the managed workload.

3. BLOOM FILTER

Bloom Filters provide space-efficient storage of sets at the cost of a probability of false positives on membership queries. A Bloom filter is traditionally implemented by a single array of M bits, where M is the filter size. On filter creation all bits are reset to zeroes. A filter is also parameterized by a constant k that defines the number of hash functions used to activate and test bits on the filter. Each hash function should output one index in M. When inserting an element e on the filter, the bits in the k indexes $h_1(e), h_2(e), ..., h_k(e)$ are set.

As shown in Figure 1 [8], Bloom filter is a bit array of M bits for representing a set $S = \{a_1, a_2, ..., a_n\}$ of n items. All bits in the array are initially set to 0. Then, a Bloom filter uses q independent hash functions $\{h_1, ..., h_q\}$ to map the set to the bit address space $[1, ..., M]$. For each item a, the bits of $h_i(a)$ are set to 1. To check whether an item a is a member of S, we need to check whether all $h_i(a)$ are set to 1. If not, a is not in the set S. If so, a is regarded as a member of S with a false positive probability, which suggests that set S contains an item a although it in fact does

not. Generally, the false positive is acceptable if it is sufficiently small. The time complexity of a standard bloom filter is a fixed constant $O(k)$, completely independent of the number of items in the set. Use of Bloom filters have a strong space advantage over other data structures for representing sets, such as self-balancing binary search trees, tries, hash tables or simple arrays or linked listed of the entries. Most of these require storing at least the data item themselves, which can require anywhere from a small number of bits to arbitrary number of bits.

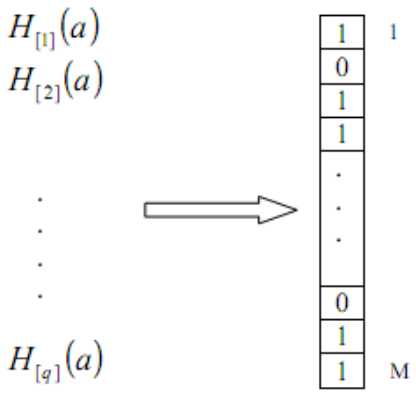

Figure 1. Standard bloom-filter structure

4. THE PROPOSED SYSTEM

In object-based storage system, a client contacts an MDS first to acquire access permission and obtain metadata of the desired file. Then the client directly accesses the respective data store to get the content. We won't consider the responsibility of object-based data store in this paper. In this section, we present a novel approach called two-stage bloom filter to gain efficient metadata management and fast metadata lookup. We use bloom filter array (BFA) on each MDS to manage metadata of multiple MDSs. A client randomly chooses an MDS to perform its request. Each MDS includes LRU-BF for providing access locality and semantic-based BF to map the workloads to the same concept space. Figure 2 demonstrates the block architecture of the proposed system.

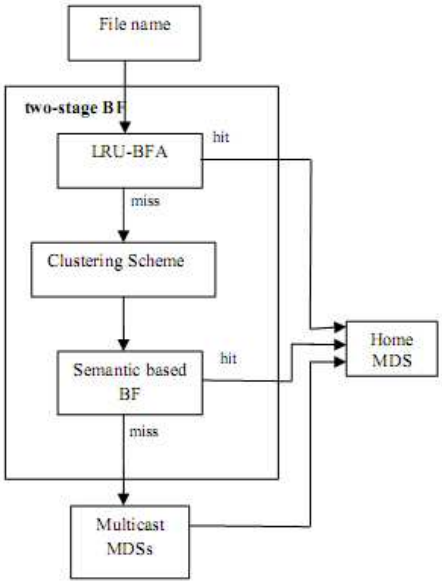

Figure 2. Block diagram of the proposed system

4.1 Two Stage BF Scheme

Figure 3 shows the structure of two-stage BF scheme. When a request comes, LRU-BF of the selected MDS starts firstly to return the hit/ miss response. The LRU-BF array maintains all the files cached in LRU (Least Recently Used) list of the corresponding MDS. The BFA returns a hit when exactly one filter gives a positive response. A miss takes place when zero hit is found in the array. If a miss takes place at LRU-BFA, the MDS calculates grouping scheme to determine the specific group. Then the request is forwarded to semantic-based BF in which the probability of hit is high because it maintains the grouped information of all MDSs. If a lookup fail, the request is multicast among all MDSs which store the information of files by grouping. In this scheme, we will use MD5 approach for hashing because of its available fast implementation.

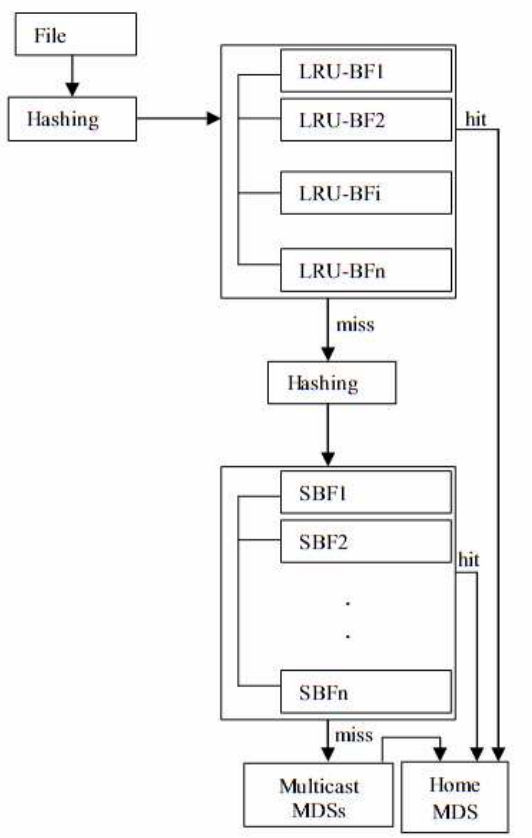

Figure 3. Cross-correlation method for image matching

4.2 Clustering Scheme

Although there are other methods for grouping, we choose LSI because of its high efficiency and easy implementation. Latent Semantic Indexing (LSI) [2, 5] generates semantically correlated groups. Semantic correlation can be exploited to optimize system performance. LSI is a technique based on the Singular Value Decomposition (SVD) to measure semantic correlation. SVD reduces a high-dimensional vector into a low-dimensional one by projecting the large vector into a semantic subspace. Specifically, SVD decomposes an attribute-file matrix A, whose rank is r, into the product of three matrices, i.e., $U\sum V^T$, where $U=(u_1,...,u_r)\in R^{t\times r}$ (t be the number of attributes) and $V=(v_1,...,v_r)\in R^{d\times r}$ (d be the number of records) are orthogonal, $\sum = diag(\sigma_1,...,\sigma_r)\in R^{r\times r}$ is diagonal, and σ_i is the i[th] singular value of A. V^T is the transpose of matrix V. LSI utilizes an approximate solution by representing A with a rank-p matrix to delete all but p largest singular values, i.e., $A_p = U_p V_p^T$.

4.3 The properties of the proposed system

To the best of our knowledge, the proposed semantic-based bloom filter is the novel one. There are some properties that are offered by the proposed system.

1. We present a scalable metadata management for distributed storage system. The proposed two-stage BF scheme can store large amount of metadata and support fast and accurate lookup with MDSs.
2. We use LRU list of accessed file so we can get the temporal locality of the system.
3. We apply the grouping scheme to collect the spatial locality of the system and to reduce the workloads for the sake of memory efficiency.
4. The client's request can start from any MDS and as a result the system can offer the balancing of the request workload.

5. CONCLUSION

In this paper, we present the efficient way to assist the storage management of distributed storage system. We emphasized on the metadata management side. The two-stage BF scheme and grouping scheme are introduced to improve temporal and spatial locality and fast lookup. Later, we intend to get dynamic balancing when a new MDS is added. On the other hand, data distribution in object-based storage will be implemented. We will evaluate our system with various kinds of workloads and show that the system can provide high throughput and better performance for distributed storage system. At this time, we cannot demonstrate the system with experimental results. Our system is still in progress and we will present the system with rich experimental results at future.

REFERENCES

[1] Brandt S. A., Xue L., Miller E. L. and Long D. D. E., "Efficient metadata management in large distributed file systems," Proceedings of the 20th IEEE /11th NASA Goddard Conference on Mass Storage Systems and Technologies, 290–298 (2003).

[2] Deerwester S., Dumas S., Furnas G., Landauer T. and Harsman R., "Indexing by latent semantic analysis," JASIST, 391–407 (1990).

[3] Hua Y., Zhu Y., Jiang H., Feng D. and Tian L., "Scalable and Adaptive Metadata Management in Ultra Large-scale File Systems", University of Nebraska–Lincoln, Computer Science and Engineering, Technical Report TR-UNL-CSE-2007-0025, Issued Nov. 20, 2007.

[4] Li W., Xue W., Shu J. and Zheng W., "Dynamic Hashing: Adaptive Metadata Management for Petabyte-scale File Systems", Proc. 23rd IEEE Conference on Mass Storage Systems and Technologies (MSST 2006), Maryland, USA, 2006.

[5] Papadimitriou C., Raghavan P., Tamaki H. and Vempala S., "Latent Semantic Indexing: A Probabilistic Analysis", JCSS, vol. 61, no. 2, 217–235 (2000).

[6] Roselli D., Lorch J. and Anderson T., "A comparison of file system workloads", In Proceedings of the 2000 USENIX Annual Technical Conference, 41–54 (2000).

[7] Weil S. A., Pollack K. T., Brandt S. A. and Miller E. L., "Dynamic metadata management for petabyte-scale file systems", In Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC '04), Pittsburgh, PA, November 2004.

[8] Xiao B., Hau Y., "Using Parallel Bloom Filters for Multiattribute Representation on Network Services", IEEE Transactions on Parallel and Distributed Systems, Vol 21, No. 1, Jan 2010.

[9] Yan J., Zhu Y., Xiong H., Kanagavelu R., Zhou F. and Lihweon S., "A Design of Metadata Server Cluster in Large Distributed Object-based Storage", In Proceedings of the 21st IEEE /12th NASA Goddard Conference on Mass Storage Systems and Technologies, 2004

[10] Zhu Y., Jiang H., Wang J. and Xian F., "HBA: Distributed Metadata Management for Large Cluster-based Storage Systems", IEEE Transactions on Parallel and Distributed Systems, Vol 19, No. 41, April 2008.