A Framework of Efficient Storage Management for Distributed Storage System

Myat Pwint Phyu
University of Computer Studies, Yangon
myatpwint.ucsy@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 which manage disk block allocation internally by object-based storage devices (OSDs), 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, we will utilize bloom-filter based approach to manage metadata and add semantic-based scheme to narrow the managed workload. Then we will optimize the data distribution in OSDs using chord mechanism.

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.

The most familiar storage system implementation is block storage, wherein blocks of data are sent to the storage system from the host over an interface, and the identity of the data is based upon the volume and the logical block address. File-based storage systems are really remote file systems, where a storage system stores data as files. In reality, file server turns it into block storage.

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 that metadata is both information 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.

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, we will also utilized bloom-filter based approach together

with a semantic filter to manage metadata in this system. On the other hand, object placement is also an importance issue in Object-based Storage System. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data item pair at the node to which the key maps. So, we will use the chord mechanism for data distribution in OSDs.

The rest of this paper is organized as follows. Section 2 shows the related works of the system. Section 3 explains some theory backgrounds and Section 4 introduces the proposed system. Then we conclude the paper with future work in Section 5.

2. Related Work

Typical algorithms for decentralized data distribution work best in a system that is fully built before it first used; adding or removing components results in either extensive reorganization of data or load imbalance in the system. Large scale persistent storage systems such as Farsite [1] and OceanStore [5] provide more file system-like semantics. Objects placed in the file system are guaranteed, within some probability of failure, to remain in the file system until they are explicitly removed. The inefficiencies that are introduced by the peer-to-peer and wide area storage systems address security, reliability in the face of highly unstable nodes, and client mobility (among other things). However, these features introduce far too much overhead for a tightly coupled mass object storage system. RUSH [3] (Replication Under Scalable Hashing) maps replicated objects to a scalable collection of storage servers or disks. It guarantees that replicas of a particular object are not placed on the same server, and allows servers to have different "weights," distributing more objects to servers with higher weights. The algorithm is very fast, and scales with the number of server groups added to the system. Because there is no central directory, clients can compute data locations in parallel, allowing thousands of clients to access objects on thousands of servers simultaneously. CRUSH [7] (Controlled Replication under Scalable Hashing), most closely resembles the RUSH. A number of issues make RUSH an insufficient solution in practice. CRUSH fully generalizes the useful elements of $RUSH_P$ and $RUSH_T$ while resolving previously unaddressed reliability and replication issues, and offering improved performance and flexibility. CRUSH a pseudo-random data distribution algorithm that efficiently and robustly distributes object replicas across a heterogeneous, structured storage cluster. The primitive rule structure currently used by CRUSH is just complex enough to support the data distribution.

In GoogleFS and HDFS [12], there is no metadata partition. Several copy of namespace metadata is maintained in multiple metadata servers to ensure reliability. Metadata is kept in memory to improve the performance. Weak scalability at both scale and performance. In static sub-tree partition like AFS, partition is based on hash value of sub-tree or static assignment and don't support sub-tree migration. In dynamic partition such as ceph [8] or farsite [1], partition is based on hash value of sub-tree or dynamic assignment and support sub-tree migration based on workload on each metadata server. The namespace of Sub-dir partition like GPFS [6] and giga+ [4] is partitioned by fixed size of directory partition. Directory partition number increased with the extension of directory. Multiple directory partitions of the same dir can process requests in parallel. As for very large directory, large directory partition has low memory utilization and low efficiency of location. In static file partition such as xFS [13], metadata partition based on file's name hash. Relation between file and server is fixed, and is not support scalable. Files distributed on MDS with no skew. On the other hand, file does not map to server directly, additional information is needed in dynamic file partition such as HBA [10] and GHBA [2]. Relation between file and server can change.

3. Theory Background

In this section, we will discuss chord mechanism and bloom filter as theory background.

3.1. Chord

Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data item pair at the node to which the key maps. Chord adapts efficiently as nodes join and leave the system, and can answer queries even if the system is continuously changing. Chord provides fast distributed computation of a hash function mapping keys to nodes responsible for them. It uses consistent hashing [11], which has several good properties. With high probability the hash function balances load. Also with high probability, when an N^{th} node joins (or leaves) the network, only an $O(1/N)$ fraction of the keys are moved to a different location - this is clearly the minimum necessary to maintain a balanced load. Chord improves the scalability of consistent hashing by avoiding the requirement that every node know about every other node.

3.2. Bloom Filter

Bloom Filters (BFs) 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.

Bloom Filter [9] 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.

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. In this section, we present the mechanisms for MDS management and data distribution in OSDs. Figure 1 shows the architecture of object-based storage system.

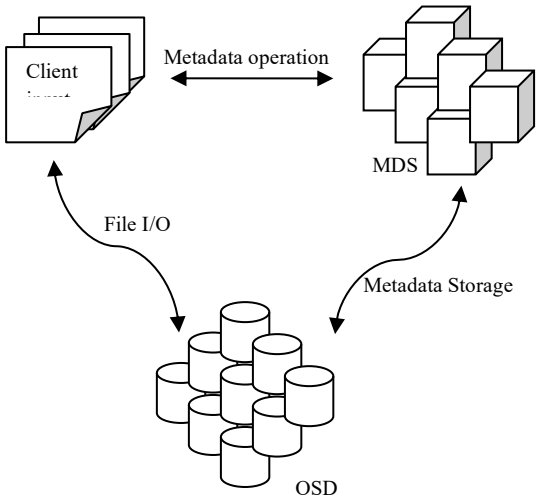

Figure 1. Architecture of object-based storage system

4.1. Metadata Server Management

We present an approach called two-stage bloom filter to gain efficient metadata management and fast metadata lookup. We use bloom filter array (BFA) on each Metadata Server (MDS) to manage metadata of multiple MDSs. A client randomly chooses an MDS to perform its request. In each MDS, it is organized including Least Recently Used-BF (LRU-BF) for providing access locality and semantic-based BF (SBF) to map the workloads to the same concept space. For SBF, Latent Semantic Indexing (LSI) is used to generate semantically correlated groups because of its high efficiency and easy implementation.

Figure 2 shows the structure of the 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 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.

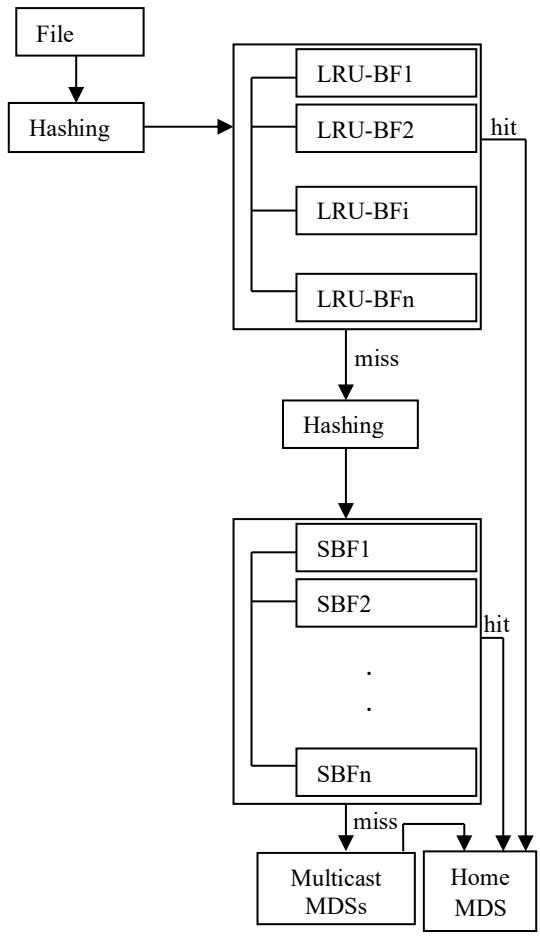

Figure 2. Structure of two-stage BF

4.2. Data Distribution Management

Object-based storage devices (OSDs) manage disk block allocation internally, exposing an interface that allows others to read and write to variably-sized, named objects. In such a system, each file's data is typically striped across a relatively small number of named objects

distributed throughout the storage cluster. Objects are replicated across multiple devices.

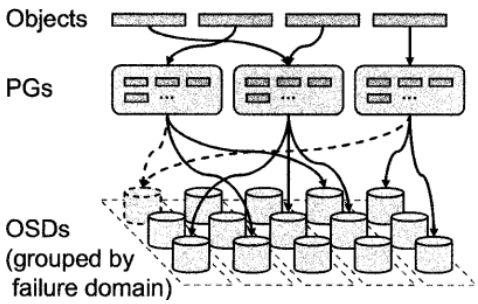

Figure 3. Block diagram of object placement mechanism

Firstly, each object's placement group (PG) is determined by a hash of the object name o, the desired level of replication r and a bit mask m that controls the total number of placement groups in the system. That is, $pgid = (r, hash(o) \,\&\, m)$, where & is a bit-wise AND and the mask $m = 2^k - 1$ constraining the number of PGs by a power of two. This is derived from ceph [8] and can be seen in figure 3. Then, the chord mechanism is used to distributed data objects.

4.3. Properties of the proposed system

To the best of our knowledge, the proposed semantic-based bloom filter is the simple 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.

Moreover, using chord mechanism to take the placement of objects also optimizes the data distribution. By applying the consistent hashing approach, the OSDs can maintain load balancing well.

5. Conclusions and Future Work

In this paper, we present the efficient way to assist the storage management of distributed storage system. The two-stage BF approach which includes grouping scheme is introduced to improve temporal and spatial locality and fast lookup for metadata management. 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 showed that the system can provide high throughput and high storage utilization for object-based 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] A. Adya, W. J. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. Wattenhofer, "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment", In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), Boston, MA, Dec. 2002. USENIX.

[2] Y. Hua, Y. Zhu, H. Jiang, D. Feng, L. Tian, "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.

[3] R. J. Honicky and E. Miller, "Replication Under Scalable Hashing: A family of algorithms for scalable decentralized data distribution", 18th International Parallel and Distributed Processing

Symposium (IPDPS 2004), April 2004, Santa Fe, New Mexico.
[4] S. Patil, G. Gibson, S. Lang, M. Polte, "GIGA+: Scalable Directories for Shared File Systems", In Proceedings of the 2nd International Petascale Data Storage Workshop (PDSW 2007).
[5] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: the OceanStore prototype", In Proceedings of the 2003 Conference on File and Storage Technologies (FAST), Mar. 2003, pp. 1–14.
[6] F. Schmuck, R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters", FAST'02 - 1st USENIX Conference on File and Storage Technologies, January 2002.
[7] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "CRUSH: Controlled, scalable, decentralized placement of replicated data", In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC '06), Tampa, FL, November 2006. ACM.
[8] S. A.Weil, S. A. Brandt, E. L.Miller, D. D. E. Long, and C. Maltzahn. "Ceph: A scalable, high-performance distributed file system", In Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI), Seattle, WA, November 2006. USENIX.
[9] B. Xiao, Y. Hau, "Using Parallel Bloom Filters for Multiattribute Representation on Network Services", IEEE Transactions on Parallel and Distributed Systems, Vol 21, No. 1, Jan 2010.
[10] Y. Zhu, H. Jiang, J. Wang, F. Xian, " HBA: Distributed Metadata Management for Large Cluster-based Storage Systems", IEEE Transactions on Parallel and Distributed Systems, Vol 19, No. 41, April 2008.
[11] http://en.wikipedia.org/wiki/Consistent_hashing
[12] http://wiki.apache.org/hadoop/HDFS
[13] http://en.wikipedia.org/wiki/XFS