Intelligent Data Management in Object-based Storage Device

Ohnmar Aung
University of Computer Studies, Yangon.
ohnmaraung2008@gmail.com

Abstract

The performance of the whole file system depends on how managing the system's components well. Consequently, designing intelligent object will catch the fancy of most researchers. Furthermore, the intelligence of OSD (Object-based Storage Devices) depends on the object intelligence. Therefore, space allocation in OSD is strongly connected with this special object.

As a nature of OSD, objects are allocated on disk continuously. But, whenever, insufficient space is in current OSD, objects also are needed to place in scattering manner among the OSD Cluster. In this paper, specific intelligent object and component object are combined to work for efficient data allocation which satisfies one of the features: intelligent space management in the storage layer on OSD. And the proposed object work well for space allocation scheme using simple mathematical logic and the result is stored using doubly linked list. The goal is to manage system free resources in predefined style for efficient space collection and fast allocation for the incoming object.

1. Introduction

The performance of the storage system depends on the way objects can be managed by using appropriate method. Thus, object attributes are very important to the OSD. The more attributes operation requests to the OSD, the more frequent of the disk I/O and the performance of the file system descend more quickly [1].

In object-based storage system, objects encapsulate user data (a file) and attributes of that data. This makes decisions on data layout or quality of service on a per-file basis, improving flexibility and manageability. OSD enables: Intelligent space management in the storage layer and Data-aware pre-fetching and caching.

An OSD is analogous to a logical unit. Unlike a traditional block-oriented device providing access to data organized as an array of unrelated blocks, an object store allows access to data by means of storage objects. A storage object is a virtual entity that groups data together that has been determined by the user to be logically related. Space for a storage object is allocated internally by the OSD itself instead of by a host-based file system. OSDs manage all necessary low-level storage, space management, and security functions. Because there is no host-based metadata for an object (such as inode information), the only way for an application to retrieve an object is by using its object identifier (OID). The collection of objects in an OSD forms a flat space of unique OIDs. According to OSD-2 draft, four types of objects are defined, which are root object, partition object, collection object and user object. User object is the container of data which lots of additional user-defined attributes can be appended. By making full use of these attributes, such as network QoS and the combination of network QoS and storage QoS. As partial storage management functions are offloaded to OSD, the construction of an intelligent OSD has caught the fancy of most researchers.

In this paper, specific intelligence object is designed for data allocation in OSD. Since all read/write requests are handled via object interface (object-id, offset, length), the latter two are fully utilized for next space allocation convenience. And Section 2 presents current storage architectures. Technical Background is proposed in Section 3. Design of Proposed system is discussed in Section 4. Section 5 is intended to describe related works and conclude in Section 6.

2. Current Storage Architectures

There are two types of network storage systems, each distinguished by their command sets. First is the SCSI block I/O command set, used by Storage Area Networks (SANs), which provides high random I/O and data throughput performance via direct access to the data at the level of the disk drive or fiber channel. Second is the Network Attached Storage (NAS) systems that use NFS or CIFS command sets for accessing data with the benefit that multiple nodes can access the data as the metadata on the media is shared. Linux clusters require both excellent performance and data sharing from their storage systems.

In order to get the benefits of both high performance and data sharing, a new storage design is required that provides both the performance benefits of direct access to disk and the ease of administration provided by shared files and metadata. That new storage system design is the Object-based Storage Architecture [2].

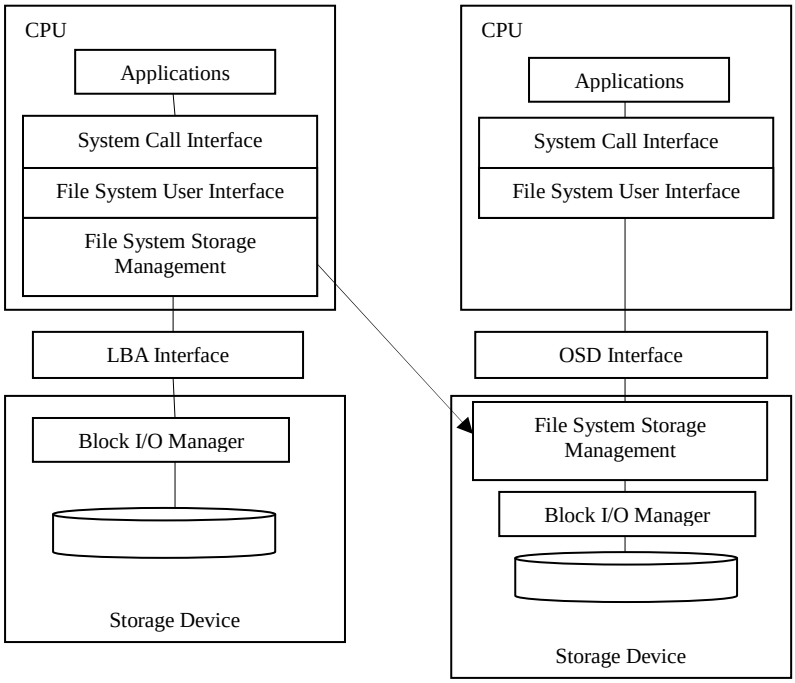


Figure 1: Block-based versus Object-based Storage Model

3. Technical Background

3.1 Block-based and Object-based Storage

In traditional block-based disks that represent storage as a linear array of bytes and export a logical block address (LBA) interface. And most of these file systems is separated into a storage component (space allocation and data access) and a user component (namespace, access control, concurrency management). Since data layout is managed by the file system, its related metadata must also need to be transferred for executing computation against the data blocks, which is a non-trivial task.

Coming back to Object-based Storage Device, it exports an object interface in which the functionality of the storage component is moved to the storage device, while the user component remains the host as illustrated in Figure 1. Since OSD has knowledge about object layout, managing unused space itself is not a difficult one. Much of metadata can be associated and stored directly with each data object and is automatically carried between layers and across devices. The use of OSD-enabled storage devices in a system allows metadata layers to be collapsed, simplifying the storage system and improving scalability. With reduced processing requirements and overhead, a single server can manage a much larger number of devices than with traditional block-based interfaces. This allows users to build storage clusters for very large installations or to use more economical systems than is possible with block-based storage [5].

3.2 Linked List

A linked list represents data structure where each node has at least two members, data itself in the list plus address of the next node. These types are called Singly Linked List in which they only point to the future node but not the previous. Another type is called Doubly Linked List, there is a node for each element, where nodes consists of a part in it the element is stored, a link to the next node, and a link to the previous node. The last node links to nothing i.e there are no nodes after it. Also, there are no nodes before the first node. There is a link to the beginning of the list called head.

For proposed system, the unused spaces need to be arranged flexibly so that nodes can be close another. Knowing the current node, back and forth addresses can be getting. Thus, the results can save time.

4. Design of proposed system

4.1 Object Storage Hierarchy

Traditionally, there are two files associated with each object, data for storing data information for the object and attribute for storing metadata information for the object as defined by

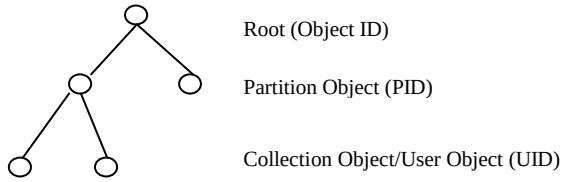


Figure 2: Object Storage File Hierarchy

the T10 standard. Like existing OSD architecture, the proposed system composed of four basic user objects: root, partition, collection and user object. The files for root object data and metadata are stored under the osd_base directory itself. Data file is named as root_data while the metadata file is named as root_attr. The partition objects are stored under a sub_directory identified by the partition_id. According to the T10 standard, there is no data information associated with root, partition and collection objects. The root objects store list of all partitions, partition data file stores list of all user/collection objects and collection data files stores list of all member objects [3]. OSD based Object File Hierarchy is shown in Figure 2.

4.2 Intelligent Object

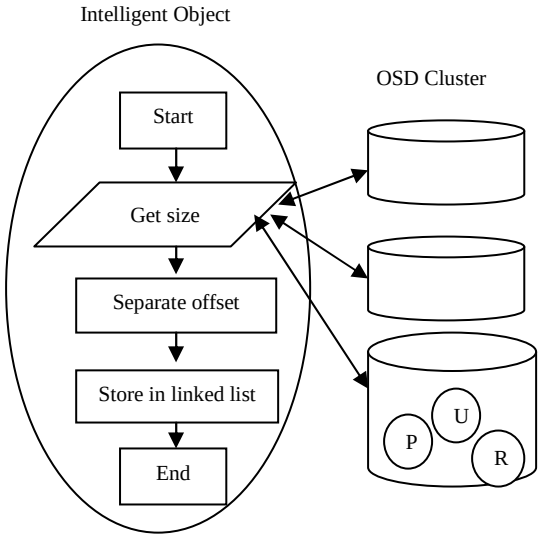
This specific object is embedded in component object. It always checks whether any changes in the attributes for space management and fast indexing. It contacts with the component object inquiring free space in the current OSD size and also with user object for used spaces.

According to T10 Standard, each object is accessed using (OID, offset, length) format, thus, intelligent object only needs to use current object's offset and length. The procedures of the proposed system are designed in the followings.

1. The object always contacts with both of the user object and component object via MPI (Message Passing Interface) in which getting specific object's offset, length and size.
2. Separate empty and non-empty spaces in current OSD by calculating difference between them.
3. Meanwhile, it stores allocated offsets in linked list. Otherwise, let it says S and B are the same size; the result for D will be zero or null which means there can no longer get for the next any request and this process need more space for continuing its work. In those cases, the intelligent object will inform the component object for more free space requirement. The process flow of the intelligent object is illustrated in Figure 3.

4.2.1 Message Passing Interface

MPI is a standardized and portable message-passing system. Message-passing systems are used especially on distributed machines with



U: User Object
P: Partition Object
R: Root Object

Figure 3: Process Flow of Intelligent Object

separate memory for executing parallel applications. The proposed system will perform master-slave message-passing pattern. The intelligent object will behave like master and the rest of the OSD clusters are the slaves. The master will inquiry the user objects in the cluster and gets responses including current object's offset, length and size.

4.2.2 Separate Empty and Non-empty Spaces

Getting MPI message, the intelligent object separates empty and non-empty spaces for the current OSD. It weights load and make decisions whether there is enough allocation to continue the work or not.

Let S ={1,2,3,4,5,6} for maximum OSD size and B={2,3,4} for current work load and difference D can be get: D= S \ B= { 1,5,6} and the object know the process can be progressed.

4.2.3 Offsets Linked List

All unallocated spaces in the OSD are organized by a linked list. Files may be stored on OSDC with two manners: with or without striping. In the case of striping, all arbitrarily sized objects are put into OSDs by allocating data continuously on disk. In the case of with stripping, one large file may be stripped into multiple size objects and all these objects stored in different OSDs for parallel data access. But, reorganizing objects which

belongs to the same file in one large object becomes more crucial. Consequently, these multiple scattered objet attributes are arranged in Linked List Order for fast searching. For example, if requested size is not enough to store in current OSD, the remaining objects need to be stored in other place. Not to disconnect, the former data object is pointed to the next other.

4.3 Component Object

In the proposed system, the component object maintains detail information about objects in each OSD. It always contacts with intelligent object via Message Passing Interface for informing current space used and unused. In addition, it weights current load and available space and if there any negative result comes out knowing from the intelligent object, consults with other component object in the group OSD for requesting required spaces.

4.4. Tree Hierarchy and Linked List

While development in the storage area continues, great things are being done for users' convenience. Researchers have been using a variety of B Tree Management for fast searching and storage efficiency. By theory, a B-tree is with 3 keys/node . Keys in internal nodes are surrounded by pointers, or record offsets, to keys that are less than or greater than, the key value. For example, all keys less than 22 are to the left and all keys greater than 22 are to the right [6]. So, their allocation is arranged in serially and we have to find the result sequentially. But, in real work, free spaces in memory cannot be in those like because files may be scattered and allocation will not be predicted.

With Linked List, we don't need to arrange the spaces in order, however, it behaves "First comes first allocate", data can be inserted and deleted more flexible. The important thing is to record the addresses of the previous and next nodes. And, contrasting with "Tree" is that insertion will mostly be taken place in the end of the List. Here, the proposed system is only intended to support for files with required spaces faster. Using Linked List, the jobs will be done well with the ability of the intelligent object.

5. Related Works

Object-based Model in Ceph, it offloads handling of low-level storage details to the storage devices. Storage devices are accessed through object interface and in addition, metadata management is decoupled from data management. Files are broken up into objects and striped across OSDs. File to object mapping is done by pseudo-random RUSH algorithm for achieving load balancing [4].

In iRBO [7], the author proposed specific intelligent object for load balancing using rule-based approach. He describes objects for defragmentation as an example but it is not always impossible to defragment object because frequent I/O access can degrade performance.

Gongye Zhou [1] and his accompany presented B+Tree Management Method of Object Attributes for Object-based Storage. Although tree hierarchy is popular in current storage technologies, the overhead of it is not small. In addition, it uses two B+Tree: one for object and the other for attributes. It can suffer from cost doubly.

6. Conclusion

Today storage structure is focused on object-based approach. However, managing space allocation with intelligent is rarely discussed in most research. The idea of having intelligent object can be significantly benefited in both space allocation and fast data searching.

In this paper, the mission of the space allocation depends on not only component object but also for intelligent object. The performance of the whole system management greatly concerned with these two components ability. In addition, either for creating or deleting request, the required linked list can be updated in not heavily. As a future work, we would like to focus on more specific objects into OSD to do other more missions and then construct an intelligent autonomous OSD Cluster.

References

[1] G. Zhou, L. Yuan, J. Chen, B+Tree Management Method of Object Attributes for Object-Based Storage. International Conference on Networking Architecture and Storage (NAS 2007).

[2] Panasas, Object Storage Architecture, White Paper. www.panasa.com.

[3] R. Shiraguppi. Design and Implementation of an OSD System and Pre-fetching Models, 2008.

[4] Sage A. Weil. CEPH: RELIABLE, SCALABLE, AND HIGH-PERFORMANCE DISTRIBUTED STORAGE, December 2007.

[5] Seagate Research. The Advantages of Object-Based Storage-Secure, Scalable, Dynamic Storage Devices, April 2005.

[6] Thomas Niemann, Sorting and Searching Algorithms: A Cookbook.

[7] YinLiang Yue, Fang Wang, Dan Feng. iRBO: Intelligent Role-Based Object for Object-based Storage Device. International Conference on Computational Intelligence and Security, 2007.