NLOV: An Innovative Object-oriented Storage System Based on BerkeleyDB [*]

Ge GuangHe, Deng Wanxi, Qi Lu, Li Yuqing, Wang Gang, Liu Xiaoguang, Liu Jing
Dept. of Computer, College of Information Technical Science,
Nankai University, 300071, Tianjin, China
gypromise@163.com

Abstract

An object-oriented storage system NLOV (Network Logical Object Volume) based on BerkeleyDB is implemented. It provides a block level object-oriented storage interface. It uses NBD and AOE as transport protocols and supports interoperation between the two protocols. Compared with other object-oriented storage system, such as Lustre and Ceph, NLOV has good simplicity, better flexibility and applicability. Some experiments are also done to examine the performance of NLOV.

1. Introduction

Recently, with the progress of Internet and storage technology, a storage based on network has become more and more prevalent. Traditional block-oriented storage devices are non-intelligent. This gets novel large-scale network storage systems into a performance/management mess. Object-oriented storage technology solves this problem very well because of the intelligence of OSD (Object-oriented System Device) [1,2]. OSD organizes user data into objects instead of blocks. OSD knows space utilization. This ability helps synchronization, reconstruction and extension performance. OSD also has inherent advantages in security, management and so on.

In this paper, an object-oriented storage system NLOV is developed. It provides a block level object-oriented storage interface. It is simple, flexible and has high applicability.

The rest of this paper is organized as follow. Related works are introduced in the next section. In Section 3, we describe key ideas of NLOV. Experimental results and analysis are given in Section 4. Finally, conclusion and future works are given.

2. Related Work

Object-oriented storage technology originates from NASD (Network Attached Secure Disks) [4] project. The main idea is organizing data as vary size *object* by which we can visit the storage system through object interface as well as management and visit security are supervised by device itself. ANSI-X3T10 [5] standard had been issued by object-oriented storage device group which is a bench of SNIA (storage Networking Industry Association) established in 1999.

PanFS [6] is an object-oriented file system developed by Panasas Company. It is composed of four parts: network, MDS (Metadata Server), OSD and client. Client can access OSD bypassing MDS. Only file metadata is managed by MDS.

Lustre [7] is an open source distributed file system developed by Cluster File System Company. It consist of client、MDS and OST (Object Storage Target). Transport protocol used is Portals which is bring forward by Sandia Company. It supports multiple networks such as TCP/IP, Quadrics, Infiniband etc. XML, LDAP and SNMP are used to manage system.

Centera [8] is an object-oriented system developed by EMC. It uses the hash value of the content of a object as its ID. Therefore, it is just suitable for the application in which updating is less frequent.

Ceph [9] is a distributed file system developed by a group in UCSC. Both data and metadata are stored in OSD and EBOFS is used for management. An object distribution algorithm CRUSH places objects properly considering multi-level hardware configuration.

pNFS [10] offers a general storage system for various storage systems based on block, file or object etc. LAYOUT is introduced in its transport protocol NFSv4. The data LAYOUT must be fectched from server for data access.

[*] This paper is supported partly by the National High Technology Research and Development Program of China (2008AA01Z401), NSFC of China (90612001), RFDP of China (20070055054), and Science and Technology Development Plan of Tianjin (08JCYBJC13000)

3. The Design of NLOV

3.1. System Structure

NLOV is consisted of 3 logical parts: storage node, MDS and client. Fig 1 shows the system structure.

3.1.1. Object. An object is consisted of three segments: ID, attributes and data. ID is a 96 bits integer. High 32 bits denotes object type and low 64 bits is the object number. There are four kinds of objects: user data objects, LOV objects, mapping table objects, and node information objects. Namely, we store data and metadata consistently by objects. Object number has different meaning according to object type. The attributes are divided into general attributes and user defined attributes. The former likes inode, contains general information of object such as size, access time, access control and so on. The latter can store any information.

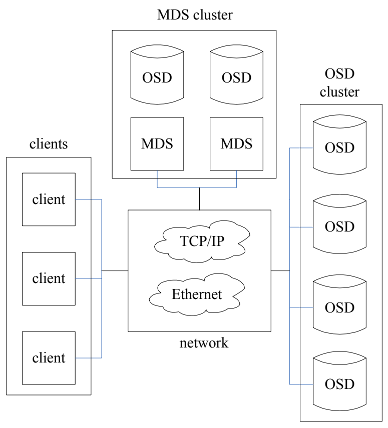


Figure 1. **System structure.**

3.1.2. Storage node. A storage node is just an OSD. We implement OSD by traditional hard disk + Berkeley DB [3] based software because of rarity of "real OSD". The software has three modules: Data Server is in charge of receiving requests from clients and sending replies back; Manager reports to MDS when the node joins or leaves; Migrator moves data to other node when nodes join or leave.

3.1.3. MDS. MDS performs two kinds of operations: One kind includes LOV creation, deletion and activation required by clients. The other is node status maintaining operations. MDS maintains a key data structure - mapping table which records object id hash space distribution across storage nodes. The MDS software is consisted of four modules: OSD module stores metadata in object fashion; MDS Handler deals with the requests from client and storage node; OSD map module maintains the mapping table; Map Updater maintains connecting message between clients and storage nodes as well as a update list.

3.1.4. Client. Client provides an access interface to users. It is a logical volume device driver running in kernel. It is consisted of two modules. The top layer is LOV module which is in charge of translating block fashion requests into object requests. A logical volume is split into fixed-size segments, and each segment is treated as an object. Therefore, each object ID is a triple (object type, LOV number and segment number). The hash value of each object is calculated, and the lowest 32 bits of it is truncated and stored in the mapping table with together with the ID. The lower layer of client is OSD_Map_Updater - a daemon process which maintains the mapping table.

3.2. Object Distribution Algorithm

The object distribution algorithm of NLOV is DIM (Dynamic Interval Mapping) [11]. The main idea is a two-stage mapping. At the first stage, all objects are mapped to the range [0, 1) using MD5 algorithm. At the second stage, the range is mapped to storage nodes according to relative ability of storage nodes. Fig 2 shows the mapping method.

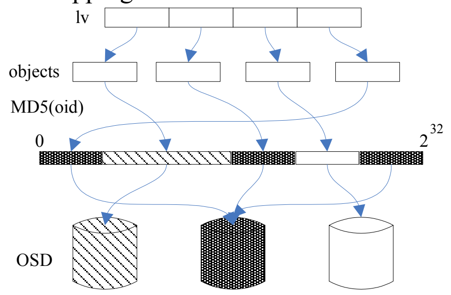


Figure 2. **Mapping method.**

When new storage nodes are added, we assign hash subranges to new nodes by cutting subranges from every old node instead of re-dividing the whole range for all nodes. When nodes leave, the reverse method is applied. Apparently, this method ensures optimal migration load.

We extend DIM to support two-way replication. We maintain a replica mapping table for each storage nodes. The basic mapping method and join/leave strategy are similar to DIM algorithm. The only difference is that the hash range is mapped to all nodes except the node in which original objects reside. Fig 3 shows the method. The first storage node stores the original objects, and the mapping table distributes replica objects over all other nodes. This method can't guarantee optimal load balance when storage nodes are different. But the simulation result shows that the distribution is near-optimal.

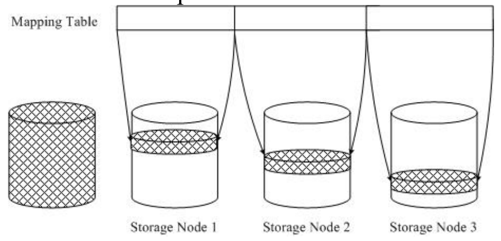


Figure 3. **Replica mapping.**

3.3. Transport protocols

NLOV supports two protocols: NBD [14] (Network Block Device) and AOE [15] (ATA over Ethernet). NBD is a fast lightweight protocol over TCP/IP developed by Pavel Machek in 1997. AOE is developed by Brantley Coile. It transports ATA requests at Ethernet layer. NLOV supports using the two protocols mixedly. So users can choose proper protocol according to different network condition and application requirement. We implement a virtual protocol layer in Linux kernel to hide the difference among concrete protocols. This layer acts like virtual filesystem layer (VFS) in Linux kernel. It makes adding new protocols easy.

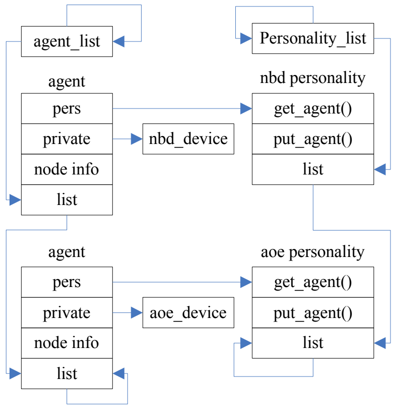


Figure 4. **Virtual protocol layer.**

To implement virtual protocol layer, we mainly define two structures: agent and agent_personality. Each agent represents an initiator. It stores the common attributes of protocols. The field "private" stores the specific attributes and methods of a concrete protocol. The field "agent_personality" points a function hook table, which provides a common template for every initiator. Fig 4 shows the structure:

In kernel space, agents are implemented using devices. In user space, we use class inheritance to implement agents. Fig 5 shows the class diagram.

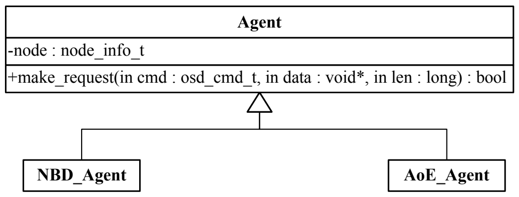

Agent
-node : node_info_t
+make_request(in cmd : osd_cmd_t, in data : void*, in len : long) : bool
NBD_Agent
AoE_Agent

Figure 5. **Agent class diagram.**

We modified NBD and AOE to provide the interface for NLOV. We modified NBD message format to handle object requests. The new format (we call it NOD) is showed in Fig 6. In which the field "osd_cmd" stores the object request message. The format of the object request message is omitted because of the length limit of the paper. Similar modification is applied to AOE.

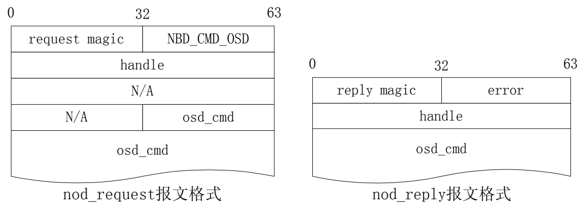


Figure 6. **NOD message format.**

3.4. Implement OSD by BerkeleyDB

The OSD in system is implemented by BerkeleyDB. BerkeleyDB is an embedded database library which supports many languages such as C, C++, Java, Perl, Python, PHP, Tcl etc. It stores each data value with an associated key. It also supports one key to multiple values mapping. It has been implemented in many platforms such as UNIX, Linux, Windows and some real time OS. OSD interface is showed in Table 1.

Table 1. **OSD Interface.**

Interface	explanation
read	Read the data from object
write	Write the data to object
remove	Delete the object
get_attr	Get the attribution of object
set_attr	Set the attribution of object
exists	Whether is the object exist
list	List the object current exist

Fig 7 shows the organization of the objects in an OSD. Every OSD has a unique root object with a specific key and the data is a list of object IDs which reside in this OSD. The list will be updated when the object are added or deleted. The operation "list" is implemented by getting the ID list from the root object.

For each object, the pair (ID, data) is stored in

Berkeley DB. Therefore "read" and "write" operations can be implemented through get and put methods of BerkeleyDB using object ID as the key.

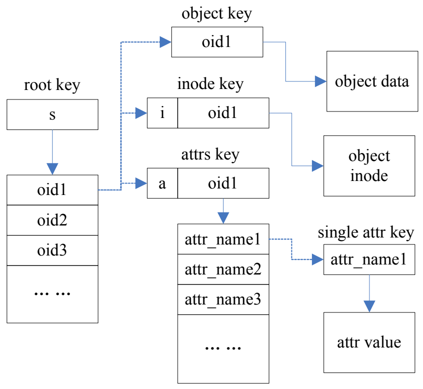


Figure 7. **Object organization.**

An "inode pair" is consisted of the object ID appended with a character 'i' as the key and the common object attributes as the data. Every object has a unique inode. Transaction mechanism provided by Berkeley DB is used to maintain consistency between a object and its inode.

Similarly, an "attribute pair" is composed of the object ID appended with a character "a" as the key and the names of user-defined object attributes as the data. For each attribute, the (name, value) pair is stored in Berkeley DB. Therefore, we can query all user-defined attribute names of an object from its attribute list, and then query any attribute value by its name as the key. The attribute list and the attributes are not necessary components of an object. They are constructed by "set_attr" operation and destroyed when the object is destroyed.

4. Experimental Results

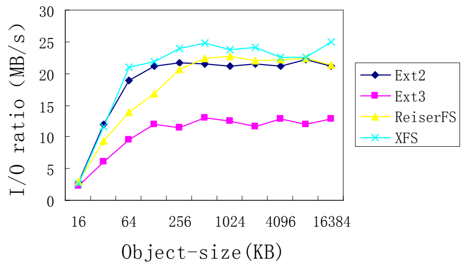
4.1. Environment

Because the client software is just a device driver running in kernel, so we choose RedHat Linux AS4 (kernel version is 2.6.9) as our OS platform. The software running on MDS and storage nodes is in user space, so we developed them by ACE (Adaptive Communication Environment) [12, 13] that provides a lot of cross-platform C++ communication modules.

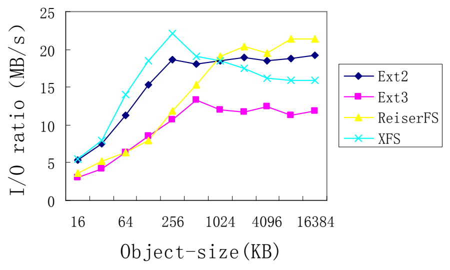
4.2. Performance of BerkeleyDB

We test the performance of OSD based on BerkeleyDB. The configuration is as follow: an Intel Celerlon 3.06 GHz CPU, a 80GB Seagate hard disk and a 40GB hard disk, 1GB memory, Red Hat Enterprise Linux AS4 update3, BerkeleyDB 4.5.20, ext2, ext3, reiserfs and xfs as underlying file system separately. Because traditional test tools aim at block device or file system is incompatible with OSD which is based on object, so we use OSD test tools OSDbench developing by myself.

Fig 8 shows the write performance. Fig 8.a shows the write (create) performance. We can see that the I/O rate increases steadily before the object size is less than 256KB and holds the line after this point. Ext3 exhibits the worst performance, reiserfs and ext2 are comparable and xfs is the best. The rewrite (update) performance is showed in Fig 8.b. xfs reaches the peak when object size is 256KB and then decreases as object size climbs while others hold the line. We also see that rewrite performance is a bit lower than write performance because of Berkeley DB.



a. write



b. rewrite

Figure 8. **Write performance.**

Read performance is shown in Fig 9. Fig 9.a shows the sequential read performance. The curves of ext3 and reiserfs fluctuate fiercely compared with ext2 and xfs. The performance of ext2 is just half of xfs. Random read performance is same. Reiserfs performs badly when object size is small because it packs many small files into single big block. Xfs have good performance attribute to every distributing group has a B++ tree which provides fast searching.

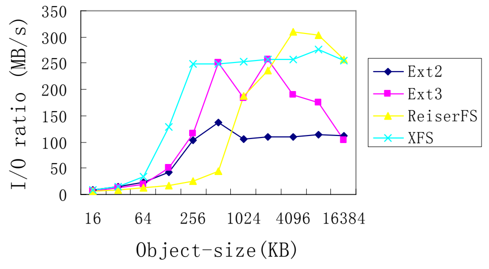
4.3. Impact of Object-size

We test the impact of object size. The configuration of the two computers used is showed in Table 2

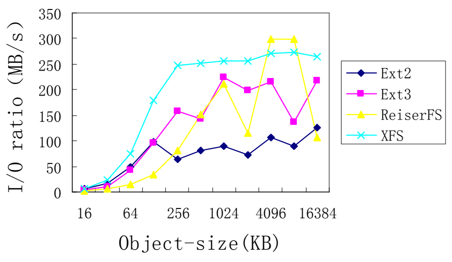
The underlying file system is ext2. Iozone is used. other parameters is as follow: 64KB request size, NBD transport protocol, 500MB test file size.

Table 2

	CPU	memory	HD
Storage node	AMD Sempron 2500+	1GB DDR	Seagate 160G
MDS client	Intel Celeron CPU 3.06GHz	512M DDR	Seagate 80G



a. sequential read



b. random read

Figure 9. **Read performance.**

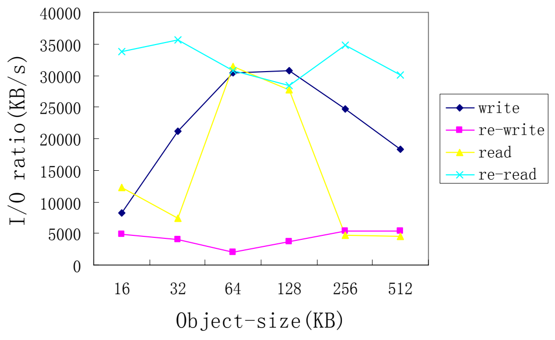


Figure 10. **Impact of Objec-size.**

Fig 10 shows the NLOV performance when object size between 16KB and 512KB. We can see that rewrite performance is low no matter how variety of object size. In the OSD experiment (section 4.2), we have came to a conclusion that the performance of OSD become better as the object size climbs. The bad rewrite performance attributes to the object size is small in this test. The write performance decreases after object size is greater than 256K because each object is part read/write by reason of the request size is 64K far smaller than object size and BerkeleyDB whole read/write is better than part read/write. As to re-read is better than read performance just because having the cache both in the client file system and OSD of storage node.

4.4. Impact of Request-size

We find that the NLOV performance is best when object size is 64KB (section 4.3), so we choice 64KB as the constant object size, other configurations were same with section 4.3.

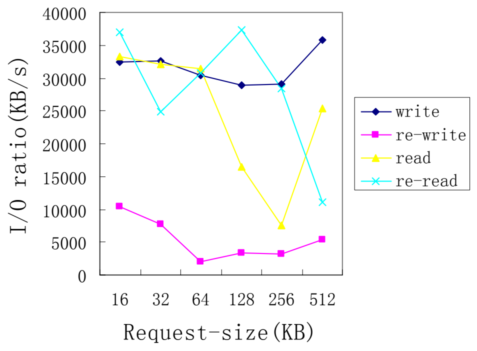


Figure 11. **Impact of Request-size.**

Fig 11 shows the NLOV performance when request size between 16KB and 512KB, we can see that both write and rewrite performance have not distinct fluctuation with the change of request size. On the contrary, read and re-read performance have great fluctuation for the reason of cache in both client and OSD.

4.5. Scalability

Scalability is very important for distributed systems. We have tested NLOV systems with different sizes. The configuration is showed in Table 3

Table 3

	CPU	memory	HD
Node 1	Sempron 2500+	1GB	160G
Node 2	PentiumD double	1GB DDR2	160G SATA
Node 3	Sempron 2500+	512MB	60G
Node 4	Celeron 3.06GHz	1G MB	80G
Node 5	Celeron 3.06GHz	512MB	80G
MDS and client	Celeron 3.06GHz	512M DDR	80G

From Table 3 we can see that configuration of each node is different, so we give a weight for every node according to it's memory ensure that the node with less memory to deal less request. The underlying file system is ext2. Iozone is used. Other parameters are as follow: 64KB request, 64KB object, 1GB test file NBD transport protocol.

From Fig 12 we can see that write performance raises sharply with adding storage node, however, rewrite performance is not increase correspondingly for the bad performance of rewrite (section 4.2). The read and re-read performance have no distinct ascend attribute to two reasons. First, one node has big fluctuation. That will counteract each other when more nodes were added. Second, we just considered the weight of memory but neglected other aspects which also affect the performance, such as CPU, HD etc.

6. Conclusion

We implemented an object-oriented storage system NLOV (Network Logical Object Volume) based on BerkeleyDB. It provides a block level object-oriented storage interface. It uses NBD and AOE as transport protocols and supports interoperation between the two protocols. Compared with other object-oriented storage system, such as Lustre and Ceph, NLOV has good simplicity, better flexibility and applicability. Some experiments are also done to examine the performance of NLOV.

At present, the performance of OSD is restricted by BerkeleyDB because it is a database software. We will research continually in some aspects include optimizing performance of OSD, developing an object-oriented QoS and enhancing the security of NLOV.

References

[1] M. Mesnier, G.R. Ganger, and E. Riedel. “Object-based storage.” Communications Magazine, IEEE, 2003, Vol 41:84~90.

[2] Michael Factor, Kalman Meth and etc. “Object storage: The future building block for storage systems.” IBM technology paper, 2005.6.

[3] Oracle, BerkeleyDB release4.5 Documentation, 2006

[4] G. Gibson et al, “A cost-effective, high-bandwidth storage architecture”, Proceedings of the ACM 8th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 92–103, 1998.

[5] T10 work group. SCSI object-based storage device command revision 1.0 [S]. T10/1355-D working draft, 2004.7.

[6] Panasas. Panfs, “Object-based architecture.” Panasas technology whitepaper, 2004.1.

[7] P. J. Braam, “The Lustre storage architecture.” Lustre Technical Report, 2002.2.

[8] EMC Centera, content addressed storage, product description. http://www.emc.com/pdf/products/centera/centera guide.pdf. 2002

[9] Sage Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, Carlos Maltzahn, Ceph, “A Scalable, High-Performance Distributed File System”, Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI ‘06), November 2006.

[10] Dean Hildebrand and Peter Honeyman, "Exporting Storage Systems in a Scalable Manner with pNFS," in Proceedings of the 22nd IEEE - 13th NASA Goddard (MSST2005) Conference on Mass Storage Systems and Technologies, Monterey, California, April 2005.

[11] Liu Zhong, Zhou XingMing. “A Data Object Distribution Algorithm Based on Dynamic Interval Mapping” , Journal of Software, 2005,16(11):1886-1893.

[12] Douglas C. Schmidt, Stephen D. Huston, “C++ Network Programming, Volume 1: Mastering Complexity with ACE and Patterns”, Addison Wesley, 2001.

[13] Douglas C. Schmidt, Stephen D. Huston, “C++ Network Programming, Volume 2: Systematic Reuse with ACE and Frameworks”, Addison Wesley, 2002.

[14] P T Breuer, A Marin Lopez, Arturo Garcia. The Network Block Device, [J].Linux Journal, 2000 ,73.

[15] http://www.coraid.com