International Journal of Computational Intelligence Systems, Vol. 5, No. 6 (November, 2012), 1173-1183

Oasa: An Active Storage Architecture for Object-based Storage System

Shuibing He [1,*] , Xianbin Xu [2] , Yuanhua Yang [3]

[1] *School of Computer, Wuhan University,*
Luojiashan Road No.16,
Wuhan, 430072, China

[*] *Key Laboratory of High Confidence Software Technologies,Peking University, Ministry of Education,*
Beijing, 100871, China

E-mail:heshuibing@whu.edu.cn

[2] *School of Computer, Wuhan University,*
Luojiashan Road No.16,
Wuhan, 430072, China

E-mail:xbxu@whu.edu.cn

[3] *School of Computer, Wuhan University,*
Luojiashan Road No.16,
Wuhan, 430072, China

E-mail:yangyuanhua123@163.com

Abstract

Active storage can largely reduce the network traffic and application execution time. In this paper, we present the design and implementation of an active storage architecture called Oasa for object-based storage system. Compared with previous work, Oasa has the following features.(1) It provides a flexible and efficient way for user to process data. User functions can process data of one user object or multiple objects at a time. (2) Oasa supports multiple patterns of user functions: both the input and output of the functions can a) come from network or disk or b) go to network or disk. (3) It keeps compatible with the current T10 OSD standard and requires little extra modification to execute user functions. Using the extended OSD commands, user can conveniently create, delete, associate and execute user functions with user objects. We also evaluate the performance of Oasa by running a typical application-data selection. It is a representative data analysis application widely used in solving real problems. Experimental results show that when the proposed active storage functions are enabled for object-based storage system, the client can obtain upto 61.9% reduction of application execution time.

Keywords: active storage, object-based storage system, object-based storage device, intelligent storage, T10 standard

1. Introduction

With the development of VLSI technology, more and more storage devices are built with powerful processors and plenty of memories. However, the considerable processing capabilities of the storage device have not been fully utilized. In addition,

some I/O intensive applications often move a large amount of data between the compute nodes and the storage nodes thus put unprecedented pressure on the network bandwidth[1]. On this occasion, active storage technology[2,3,4,5] has been prosed and is proved to be one of the most effective approaches to reduce the bandwidth requirements between storage and compute nodes. By exploiting the under-utilized processing power of storage devices to process data on the device without moving it to the host, active storage is able to not only reduce the network traffic, but also provide aggregative processing intelligence when multiple devices are used parallelly.

Numerous academic research institutions have made contributions to the active storage research field, such as Active Disk[3] and IDISK[6]. However, with the narrow storage interface(e.g., IDE/SCSI), active storage may result in complicated management and high overhead.

On the other hand, object-based storage[7] has gained enormous popularity in storage area. To achieve goals such as cross-platform data sharing, policy-based security, direct access, and scalability, many object-based storage systems (e.g., Lustre[8], Storage Tank[9], Centera[10]and Ceph[11]) have been developed by the industrial community. In order to regulate the more and more sophisticated object-based technology, the object-based storage interface standard[12] (also referred as T10 OSD standard) is being developed by the Storage Network Industry Association. With more expressive object interface, there is a potential for object-based storage devices (OSDs) to be more intelligent.

To further improve the object-based storage, there have been several efforts to integrate active storage into the object-based storage technology. John et al.[13] proposed an active storage framework for object-based storage device. Piernas et al.[14] gives an active storage strategies implemented in user-space for Lustre parallel File System[10]. Qin et al.[15] present a hybrid approach combining a request-driven model and a policy-driven model to execute code on OSDs. However, these works are preliminary, and user function can process data of one object at a time, and this fixed data processing pattern sometimes is not suitable to satisfy the users requirements.

In this paper, we present the design and implementation of an active storage architecture called Oasa that is based on the current T10 OSD standard. Specifically, Oasa has the following unique advantages.

First, Oasa provides a flexible and efficient way for user to process data. User function (application-specific code) can process data of one user object or multiple objects at a time. Second, Oasa supports multiple patterns of user functions: both the input and output of the functions can a) come from network or disk or b) go to network or disk.

Third, Oasa keeps compatible with the current T10 OSD standard and requires little extra modification to execute user functions. Using the extended OSD commands, user can conveniently create, delete, associate and execute user function with user object.

We also evaluate the performance of Oasa by running a typical application: data selection which is a representative data analysis application and widely used to solve real-word problems.

The rest of this paper is organized as follows. In section 2, we give an overview of object-based storage and the related work on active storage. Then we describe the design and implementation of Oasa in section 3. In section 4, we run a real application on the Oasa prototype and evaluate the performance of Oasa. Finally, section 5 concludes the paper and points out some future work.

2. Related work

In this section, we first give an overview of object-based storage. Then we present the related work on active storage.

2.1. Overview of object-based storage

An object is a storage container with variable-length and can be used to store any type of data. Besides data, an object has numerous attributes that are used to describe the characteristics of the data. An object behaves exactly like a file, it can be created and deleted, and can grow and shrink their size during their lifetimes. Different from traditional file-based

storage architecture, Object-based storage divides the file system into two components: the user-related component and the storage-related component, and offloads the latter component into the object-based storage device (OSD).

The architecture of the Object-based storage system (OBSS) is shown in Fig. 1. The OBSS consists of three main parts: the Metadata Server (MDS), the OSDs, and the clients. The MDS provides objects mapping information and authentication for clients' data access. When a client accesses the data in an OSD, it first contacts with the MDS and gets the mapping information about the objects. Then the client interacts with the OSD directly. Here the request contains object ID, an offset within the object, attribute values, and so on. Finally, the OSD receives the object-based request and performs corresponding operations.

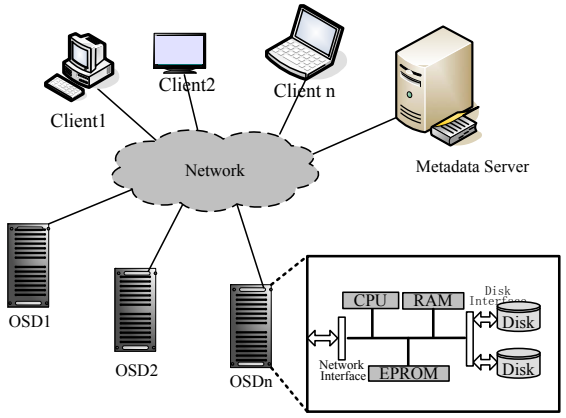

Fig. 1. The architecture of the object-based storage system
.

The OSD is one of the cornerstones of the OBSS[16]. It is an versatile storage device that contains CPU, memory, the storage media (disk), and the network interface which allows it to manage the local object store, autonomously serve, and store data from the network. Generally speaking, the OSD provides three major functions: object management, device security management, and network communication.

In order to further promote the development of object-based technology, the International Committee for Information Technology Standards (ICITS) under Storage Networking Industry Association (SNIA) developed the T10 OSD standard. The T10 OSD standard[12] defines the OSD model and the basic command set (e.g., READ and WRITE command) that operate data on the OSD. Comparing with the traditional file, the object is a more expressive interface. Each object is identified by an object ID, and is accessed by offset, length, and so on. Besides user objects, there are three kinds of objects, namely, root objects, partition objects, and collection objects, used to address user objects more efficiently. In addition to the data access commands, the T10 OSD standard also defines attribute commands (e.g., GET ATTRIBUTES and SET ATTRIBUTES) that are responsible for attributes retrieval and setting.

2.2. Related work on active storage

The idea of active storage was first developed in the database area, such as CASSM[17] and RAP[18]. However, as disk drive has limited performance and the cost is expensive in the earlier time, the database machines are quickly eliminated by the market.

With the development of VLSI technology, numerous researchers focus on active storage technology again. Riedel et al.[4] proposed Active Disks, which exploits the processing power on individual disk drives to migrate and execute of general-purpose code on disk drives. Keeton et al.[6] presented intelligent disks (IDISKs), designed for decision support database system. Similar to Active Disks, IDISKs use low-cost embedded processors, main memory, and high-speed serial communication links on each disk. IDISKs are connected to each other via these serial links and high-speed crossbar switches, overcoming the I/O bus bottleneck of conventional systems. Our work on active storage, however, focuses on OSDs with commodity disk drives, general purpose CPUs, and common operating systems.

Previous work has focused on limited applications such as image processing, decision support and data mining. Acharya et al.[3] presented a stream-based programming model of active disks, in which an application is divided into a host part and a disk-resident part(called disklet). Disklet is a piece of JAVA code, and it can be efficiently and safely

executed on the processors embedded in the disk drives. A disklet takes one or more streams as input and generates one or more streams as output. MapReduce[19,20], a concept similar to active storage has also been employed in cluster computing field, MapReduce splits the computations and maps them to many computers processing the data locally, then the sub-results of the split computation are merged for form the global result of the problem.

The above researches are built on the storage systems based on the block-level interface. This makes it hard to handle the block mapping and results in surprisingly high overhead. Since object-based storage technology may significantly influence the networked storage industry, a lot of researchers have gradually made efforts to integrate active storage into the object-based storage system. Piernas et al. gave an active storage strategy implemented in user-space for Lustre parallel File System[14,10], which proves to be faster, more flexible, portable, and readily deployable than the kernel-space version. Huston et al. presented an active storage architecture called diamond[22], used for interactive search of non-indexed data from the massive storage system. The idea of object-based storage has been embodied in diamond. For instance, the object attribute is used to determine whether an object may be of interest to the users. By running a set of application-generated filters inside storage devices, diamond can perform efficient filtering of large data collections. However, these two systems are designed for specific storage platforms with private storage protocols. In contrast, Oasa is designed for general object-based storage platform and do not comply with the T10 OSD standard.

In order to promote the development of object-based storage technology, several work tried to integrate the active storage technology into the object-based storage technology based on the T10 OSD standard[15,21,13]. Recently, Xie et al. presented an object-based active storage framework with a preliminary security solution[23]. However, these work is preliminary and lack of detailed implementations, so it is difficult to make the solutions be practical and convincible. In contrast, Oasa provides more detailed implementation of how to gain benefits from the object-based active storage technology. Furthermore, previous work only allows one user function to process data of one object at a time. This fixed data processing pattern may limit the potential of the performance improvement when a lot of small requests are served. Contrastively, Oasa provides more flexible way for users to process data. By allowing user function to process data of multiple objects in one command, Oasa can further improve storage system performance and better meet user demand for data access.

3. Oasa design and implementation

In this section, we first describe the design of Oasa. Then we present the details on implementations of Oasa.

3.1. Oasa architecture

In traditional OSD software stack, there isn't a component supporting active storage. As a result, we modify the OSD software design based on our earlier work[24] to employ the concept of active storage.

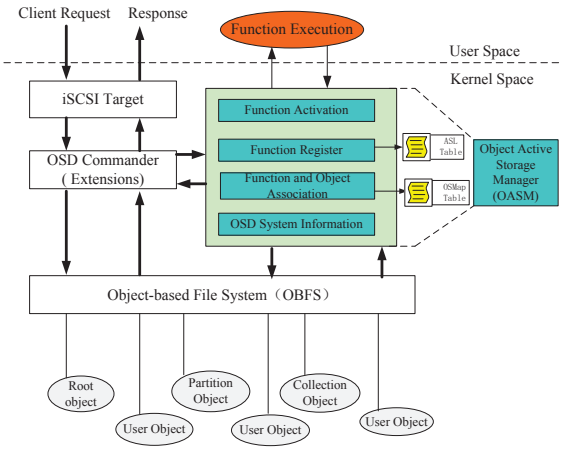

Fig. 2. Oasa architecture overview

Fig. 2 shows the architecture of Oasa in an OSD device. As depicted in Fig. 2, Oasa includes four modules, namely, the iSCSI Target, the OSD Commander, the Object-based File System (OBFS), and the Object Active Storage Manager (OASM). The last component OASM does not exist in traditional

OSD software stack. OSD Commander uses OBFS to perform object (data and attributes) access and retrieval, as well as OASM to provide active storage service for clients to execute their functions on the OSD device. Oasa is able to handle two kinds of OSD commands, namely, the existing commands defined in the T10 OSD standard and the new extensional commands proposed in this paper, and they were referred as STD CMD and EXT CMD respectively in later sections.

Typically, a client request issued to an OSD integrated with OASM will be processed as follows.

(1) ISCSI Target receives the iSCSI commands and forwards the OSD commands carried in the iSCSI package to OSD Commander.
(2) OSD Commander resolves the command according to its type. If the command is a STD CMD, it will be delivered to the OBFS. Otherwise, it will be delegated to the OASM for further processing.
(3) OASM prepares the environments for execution of application function, and then invokes a process in user space to execute application code.
(4) When the execution of a user function is completed, OASM writes the result to the disks through OBFS or returns it to the client through network.

3.2. Object-based active storage manager

OASM is the core of employing the idea of active storage. In order to perform their own functions conveniently on the OSD for every client, OASM provides a transparent, safe mechanism that consists of the following aspects.

3.2.1. Application function management

OASM is need to addresses the problem of how to express, offload, identify, and execute an application function between the client and the OSD. In Oasa, an application-specific function is denoted by a piece of code (written in C/C++/Java) and it is delivered to the device at first. A client uses the function register interface to offload their function code on the OSD , and the OSD returns the client with a function ID used to identify which function should be executed later. In order to locate the function code in the disk effectively, OASM maintains an active storage location table(ASL table) which can quickly show the disk location according to the function ID.

However, the function code may not be needed any longer, so the client can delete it freely. Therefore, OASM also provides a function cancel interface resolved in deleting the function code and the related entry in the ASL table.

3.2.2. Association between object and function

In addition to the function from the clients, OASM also defines what data a function can process. In an OSD device, data is accessed by the interface of object, OASM uses the association between object and function to describes which object(s) can be processed by which function according to a function ID.

There are two kinds of association between object and function. The first is *default association*, and on this scenario an object does not need to specify the function ID and it can be processed by a default function. OASM also maintains an object-function map table(OFMap table), in which each entry is in the form of $< objectID, functionID >$. When a client sends a request including an EXT CMD to an OSD, OASM will check the EXT CMD's parameters. If a function ID is not specified in the parameters, OASM will use the object ID as a key to search the default function ID in the OFMap table. If found, then OASM will continue to process with the function identified by the search result. Otherwise, an error message will be returned to the client.

The second association is the *explicit association*, where the object ID and the function ID needs to be specified together. However, if the disk location of the related function given by the client does not exist in the ASL table, OASM also returns an error message to the client.

As OASM provides interface for client to get rid of their functions resided on an OSD device, OASM also allows client to remove the default association between object and function. On this case, an entry is needed to be deleted from the OFMap table.

3.2.3. *Execution of function*

Ultimately, the application function should be executed on the disk on demand of client's request. As our OSD is usually equipped with mediocre PC and operating system (e.g., Linux), OASM is in charge of building the running environment for the execution of the application function. OASM first prepares parameters for the function code, and then invokes the code. After the function code is finished, OASM deals with the result according the OSD command type.

In order to provide a safe, transparent approach for clients to execute their functions on an OSD device, OASM provides the OSD system information interface to describe the platform-related information (e.g., OSD CPU architecture, operating system kernel version and C library, if needed) for the clients. An advantage of such design is that the clients are able to execute their codes on cross-platforms.

3.3. *Flexible data processing in Oasa*

Different from previous researches[15], Oasa provides more flexible approach for user to process data. Oasa achieves this goal through the following two ways.

3.3.1. *Flexible association between object and function*

In the existing OSD standard, a client can access the data of one user object with one OSD command, thus these earlier work[13,15] based on the existing OSD standard in fact provides only an one-to-one association between object and function. In this paper, we extend the association to an many-to-one association: in one EXT CMD one user function is able to process multiple objects. As a result, more flexible granularities, such as part of a user object, the whole user object and multiple objects are supported by an EXT CMD. In Oasa, we use collection object defined in the current OSD standard to represent multiple objects, and a user can deal with multiple objects by specifying the collection object identifier in the EXT CMD.

Employing the many-to-one association between object and function brings benefit. Namely, the OSD command transmission times between clients and OSDs can be largely reduced. As the iSCSI protocol which is used to transfer commands and data is a heavy-weight protocol[25], the less interaction the less overhead of command resolving. For instance, a remote client that needs to process 100000 objects maybe need to send 100000 requests if the one-to-one association is adopted. However, if the many-to-one association is used, 99999 requests can be omitted. From the above discuss, we believe that our design is specifically good for the applications with a large amount of small objects because in those cases protocol overhead is considerable in term of the whole data transfer process.

3.3.2. *Flexible data process patterns*

Besides the association between object and function, the input and output of a function also affect the flexibility of the active storage service. In this paper, we use process pattern to represent the input and output of a function, and it indicates the various sources and destinations of the data processed by a function.

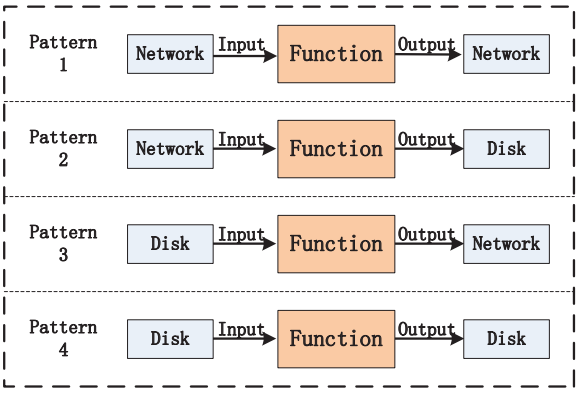

Fig. 3. Data process patterns in Oasa

Oasa provides four process patterns for clients to execute their function on an OSD device. As indicated in Fig. 3, the input and output of a function can be both network client and disk. Similar to the previous work[13,14], data flows in pattern 1 and pattern 3 are as same as that in the normal OSD WRITE and

READ command. On this occasion, intelligent operation such as data filter (e.g. encryption and compression) can be performed as pattern 1, and other operations (e.g. decryption and decompression) can be performed as pattern 3. In this paper, we also extend the patterns to pattern 2 and pattern 4. Pattern 2 decreases the slow operations on the disk, and pattern 4 reduces the transmission on the network, so they can further improve the performance of the whole system.

3.4. Extensions to the T10 OSD standard

In order to integrate the concept of active storage and the object-based storage technology, we make new interface for OSD device based on the existing object interface. Through modifying several existing commands and their corresponding Data Out-Buffer, Oasa keeps compatible with the current T10 OSD standard and makes limited extend to execute user functions.

Firstly, we change some fields in four OSD commands (CREATE, REMOVE, READ and WRITE) to employ active storage. From the command descriptor block (CDB) of these commands, we find they have similar structure: the bit 3 to 0 of byte at offset 11 are reserved, so we use these 4 bits as the active storage control (ASC) field to show distinction.

Table 1 shows the modified CDB format of the CREATE command. When the ASC=0000b, the meaning of the CREATE command is as the same as that in the current OSD standard. However, when the ASC is set to be other value, the CREATE command becomes to be a EXT CMD supporting special purpose for active storage. Table 2 gives the description of CREATE and REMOVE command with different ASC values, and the descriptions of other commands are omitted here.

Then we also add five new fields (F1-F5) to the Data-out Buffer of every command to deliver parameters and additional information for the extended command. Table 3 describes the format of Data-Out Buffer, and the F1 field specifies the application function ID, the F2 field gives the user object ID that needed to processed, the F3 field is used to transfer the user's executable code, the F4 field specifies the parameters of the function and the F5 field is used to locate the result.

Table 1. CREATE command format

Bit Byte	7	6	5	4	3	2	1	0
8	MSB	SERVICE ACTION(8882h)						
9								LSB
10	Reserved			DPO	FUA	ISOLATION		
11	Reserved		GET/SET		Reserved → ASC			
12	TIMESTAMPS CONTROL							
...	Reserved							
16	MSB	PARTION_ID						
23								LSB
24	MSB	REQUESTED USER_OBJECT_ID						
31								LSB
...	...							
...	...							
235	...							

Table 2. Extended commands

Commands	ASC	Description
CREATE	0000	Original CREATE command
	0001	Function register
	0011	User object makes association with one function
	0100	Collection object makes association with one function
	others	Reserved
REMOVE	0000	Original REMOVE command
	0001	Function cancel
	0011	Association between a user object and a function is canceled
	0100	Association between collection object and a function is canceled
	others	Reserved

With the aid of the new fields, the new EXT CMDs are qualified for executing the clients' function on an OSD device. Of course, different EXT CMD may need different fields mentioned above, so they should be selectively specified according to the type of the command rather than specified all together. For example, when the ASC in an EXT CREATE CDB is 0001b, it is used as a function register command. On this occasion, the REQUESTED USER_OBJECT_ID field at byte offset 24-31 in the CREATE CDB is used to fill the expected function

ID for a client. If the function ID field is zero, then the OSD will automatically allocate a function ID for the client. At the same time, the function code is filled in the F3 field of the Data-Out Buffer, and the length of the function code is assigned by the content at byte offset 34-41(8 bytes) in the CREATE CDB.

Table 3. Data-Out Buffer format

Bit / Byte	7	6	5	4	3	2	1	0
0	MSB	(F1)Function ID, if any						
15								LSB
16	MSB	(F2)USER_OBJECT ID1, if any						
31								LSB
...	MSB	(F2)USER_OBJECT ID2, if any						
...								LSB
j	MSB	(F3)Function data,if any						
k−1								LSB
k	MSB	(F4)Function Parameters1, if any						
l								LSB
...	MSB	Function Parameters2, if any						
m−1								LSB
m	MSB	(F5)Data offset1, data length1, if any						
m+15								LSB
...	MSB	(F5)Data offset2, data length2, if any						
...								LSB

3.5. Implementation

We implement a prototype of Oasa based on our earlier work[24] which includes the OSD software stack in Linux. A client downloads the function code on the OSD through the extended CREATE command, and the OSD stores the code (similar to the WRITE operation) on its disk and returns back a function ID. As the code will be called hereafter, one concern is how to locate the code according to the function ID. Though both the object ID and the file path both can be used to identify the function code in Linux, we adopt the later due to the convenience for code execution. In order to effectively search the entry of the ASL table consisting of numerous entries, a hash table is maintained to manage the mapping information.

The second concern is how to execute the function code in Linux. As OASM is in the kernel space while the function code is invoked in the user space, we use the function call_usermodehelper() in Linux to execute application function codes in kernel space and deliver corresponding parameters to function codes.

The last concern is how to address the communication between OASM and the user processes derived from the function codes. In our implementation, OBFS is built on a general purpose file system, so OASM and the user processes exchange data in different space by accessing a data file. In order to improve the efficiency, the inter-process communication mechanism—pipe is used: user writes the result to a pipe, and OASM gets the result though a pipe read operation.

4. Experiments

In this section, we will evaluate the performance of Oasa by running a representive application-data selection.

4.1. Experimental Setup

Our experiment test bed consisted of a client and an OSD. The client and the OSD have the same hardware components, and they are connected through the gigabit Ethernet network. The configuration of each node is shown in table 4.

Table 4. CONFIGURATION

	CPU	Memory	Disk	Network
Client	Xeon 3.0G	512M DDR	200GB/SATA	BCM5700
OSD				
Switch	Cisco Catalyst 3750 GE switch			
OS	Redhat 9, Kernel Version 2.4.20			

We use a typical application-data selection to discuss the performance of OASA. Selecting the requested data from a large-scale data set is a representative operation in database field and it is widely used in real-world problem solving. In our test, the data set is a data sequence consisting of millions of data, each of which is 0-9, and the large-scale sequence is stored as a user object in the OSD.

We test the performance of the application under the traditional storage architecture and the new architecture with active storage. In order to show the

difference, hereafter the former is called TS test and the latter is called AS test.

In the TS test, a client first fetches data to the local memory through a OSD READ command from the OSD device, and then performs the data selection operation. The execution time of the application consists of object read time through network and object process time on the host. In the AS test, a remote client first downloads the function code onto the OSD device, and then associates the function code with the user object including the data set. After that, the client requests the data selection operation on the OSD device through a new EXT CMD. Finally, the OSD returns the result to the client through network. In this test, the execution time of the application consists of object read time, object process time on the OSD, and the result transfer time.

4.2. Results

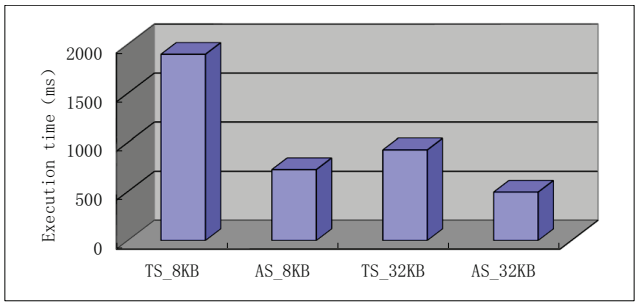

Fig. 4. Application performance under different read size

Fig. 4 shows the application execution time with the data selection condition is "data value is less than 2". We test the performance of two scenarios: the object read size of each time is 8KB and 32KB. As can be seen from Fig. 4, the active storage test under object read size 8KB and 32KB can reduce the application execution time 61.9% and 46.5% respectively comparing with that in TS tests. This result indicates that active storage can improve the application's overall performance. At the same time, we can find that the application execution time has a relationship with the object read size: a large size read will further improve the performance comparing with a smaller request size. This is because the iSCSI protocol is a heavy weight protocol, and a large size read operation each time will reduce the time of data transmission, thus reduce network overhead.

Table 5. The percentage of the data in the result

Conditions	Percentage
Value < 3	19.04%
Value < 5	38.09%
Value < 7	57.1%
Value < 9	76.2%

In addition, we test the application execution time under different data selection conditions. The percentage of the data in the result accounts for that in the original data set under different conditions is shown in Table 5.

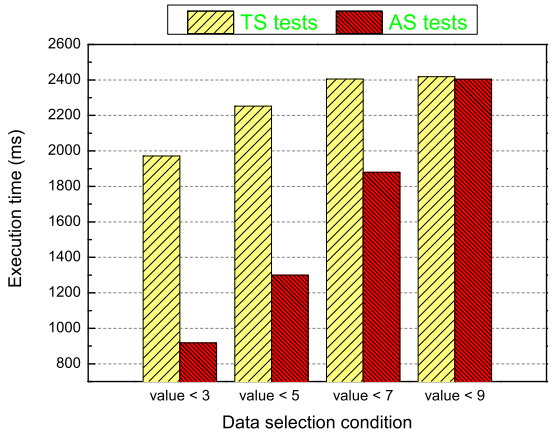

Fig. 5. Application performance under different selection condition

Fig. 5 describes the application execution time under different data selection conditions where the object read size is 8KB. From the figure we can observe that the application execution time in AS test under different conditions are decreased by 53.4%, 42.29%, 21.83%, and 0.58%. Although the performance improvements in AS tests continues to decline, the performance of active storage is always better than that in traditional storage test. With the change of the data selection conditions, more and more data need to be processed on the OSD, even the data in the result is nearly to the amount of the original data sequence. As a result, the difference

between execution time in AS test and TS test becomes small. From the above discusses, it can be seen when the returned data is much smaller than the original data, the benefit of active storage is particularly evident. Of course, if more than one OSD are deployed in the system to parallel process, benefit of active storage will be more significant.

5. Conclusions and future work

In this paper, we propose an active storage architecture called Oasa to integrate the active storage technology into the object-based storage system. Oasa provides a flexible and efficient way for user to process data, keeps compatible with the current T10 OSD standard, and requires little extra modification to execute user functions.

We design and implement the prototype of Oasa on the Linux operating system, we evaluate the prototype's performance of Oasa by running a typical application-data selection, which is a representative data analysis application and widely used in the real world. Experimental results show that Oasa can obtain upto most 61.9% reduction of application execution time.

In future work, we would like to test our prototype with more extensive applications. Moreover, we think it is a more promising approach to employ active storage on a distributed computer system. Therefore, how to determine a user function should be executed in a host computer or in an OSD, and how to coordinate the underlying OSDs to finish a task, are the directions of future research on active storage.

Acknowledgement

We would like to thank Bin Wang, Yulai xie, Yanli Yuan and Changhu Ma for their help in this study. This work was funded by Fundamental Research Funds for the Central Universities No. 3101012, Key Laboratory of High Confidence Software Technologies Program No. HCST201104.

References

1. P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Car- son, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lu- cas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yellick, "Exascale computing study: Technology challenges in achieving exascale systems," Tech. Rep. TR-2008-13, DARPA, September 2008.
2. X. Ma, A. Reddy, I. Center, and C. San Jose, "Mvss: an active storage architecture," *IEEE Transactions On Parallel and Distributed Systems*, **14**,993–1005 (2003).
3. A. Acharya, M. Uysal, and J. Saltz, "Active disks: Programming model, algorithms and evaluation," *ACM SIGPLAN Notices*, **33**,81–91 (1998).
4. E. Riedel, G. A. Gibson, and C. Faloutsos, "Active storage for large-scale data mining and multimedia,"*Proceedings of the 24rd International Conference on Very Large Data Bases*, 62–73 (1998).
5. H. Tang, A. Gulbeden, J. Zhou, W. Strathearn, T. Yang, and L. Chu, "The panasas activescale storage cluster- delivering scalable high bandwidth storage," *Proceed- ings of the ACM/IEEE SC2004 Conference on Supercom- puting*, 53–62 (2004).
6. K. Keeton, D. A. Patterson, and J. M. Hellerstein, "A case for intelligent disks (idisks)," *ACM SIGMOD Record*, **27**,42–52 (1998).
7. M. Mesnier, G. Ganger, and E. Riedel, "Object-based storage: pushing more functionality into storage," *IEEE Potentials*, **24**,31–34 (2005).
8. P. Schwan, "Lustre: Building a file system for 1000-node clusters," *Proceedings of the 2003 Linux Symposium*, 380–386 (2003).
9. J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg, "Ibm storage tank-a heterogeneous scalable san file system," *IBM Systems Journal*, **42**,250–267 (2003).
10. D. Nagle and B. Welch, "object-based cluster storage system," *Proceedings of the 23st IEEE/14th NASA Goddard Conference on Mass Storage Systems and Tech- nologies*, (2006).
11. S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," *Proceedings of the 7th symposium on Operating Systems Design and Implement*,307–320 (2006).
12. R. O. Weber, "Information technologyscsi object-based storage device commands -2 (osd-2), revision 5," Tech. Rep. Technical report, INCITS Technical Committee T10/1729-D, Jan 2009.
13. T. M. John, A. T. Ramani, and J. A. Chandy, "Active storage using object-based devices," *Proceedings of the IEEE Interna- tional Conference on Cluster Com-*

puting, 472–478 (2008).
14. J. Piernas, J. Nieplocha, and E. J. Felix, "Evaluation of active storage strategies for the lustre parallel file system,"*Proceedings of the 2007 ACM/IEEE conference on Supercomputing*,1–10,(2007).
15. L. Qin and D. Feng, "Active storage framework for object-based storage device," *Proceedings of the IEEE 20th International Conference on Advanced Information Networking and Applications*,97–101 (2006).
16. S. He and D. Feng, "Implementation and performance evaluation of an object-based storage device," *Proceedings of the International Workshop on Storage Network Architecture and Parallel I/Os(SNAPI'07)*,129–136 (2007).
17. S. Y. W. Su and G. J. Lipovski, "Cassm: A cellular system for very large data bases," *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 456–472 (1975).
18. E. A. Ozkarahan, S. A. Schuster, and K. C. Smith, "Rap: an associative processor for data base management," *Proceedings of the AFIPS Joint Computer Conferences*, 379–387 (1975).
19. J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Proceedings of the 6th symposium on Operating Systems Design and Implement*,138–150 (2004).
20. M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," *Proceedings of the 8th symposium on Operating Systems Design and Implement*,29–42 (2008).
21. A. Devulapalli, I. Murugandi, D. Xu, and P. Wyckoff, "Design of an intelligent object-based storage device," `http://www.osc.edu/research/network_file/projects/object/papers/istor-tr.pdf`
22. L. Huston, R. Sukthankar, R. Wickremesinghe, M. Satyanarayanan, G. R. Ganger, E. Riedel, and A. Ailamaki, "Diamond: A storage architecture for early discard ininteractive search," *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, 73–86 (2004).
23. Y. Xie, K. Muniswamy-Reddy, D. Feng, D. Long, Y. Kang, Z. Niu, and Z. Tan, "Design and evaluation of oasis: An active storage framework based on t10 osd standard," *MSST*,1–12 (2011).
24. S. He and D. Feng, "Design of an object-based storage device based on i/o processor," *ACM SIGOPS Operating Systems Review*, **42**, 30–35 (2008).
25. B. K. Kancherla, G. M. Narayan, and K. Gopinath, Performance evaluation of multiple tcp connections in iscsi, *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*,239–244 (2007).