Title

Design and Evaluation of Oasis: An Active Storage Framework based on T10 OSD Standard

ISBN

9781457704284

Authors

Xie, Yulai
Muniswamy-Reddy, Kiran-Kumar
Feng, Dan
et al.

Publication Date

2011-05-01

DOI

10.1109/msst.2011.5937220

Peer reviewed

Design and Evaluation of Oasis: An Active Storage Framework based on T10 OSD Standard

Yulai Xie[†‡], Kiran-Kumar Muniswamy-Reddy[§], Dan Feng[†¶], Darrell D. E. Long[‡¶]
Yangwook Kang[‡], Zhongying Niu[†], Zhipeng Tan[†¶]
[†]*School of Computer, Huazhong University of Science and Technology*
Wuhan National Laboratory for Optoelectronics
[‡]*University of California, Santa Cruz*
[§]*Harvard University*
Email: ylxie@smail.hust.edu.cn, kiran@eecs.harvard.edu, dfeng@hust.edu.cn
darrell@cs.ucsc.edu, ywkang@soe.ucsc.edu, niuzhy@gmail.com, zhipengtan@163.com

***Abstract*—In this paper, we present the design and performance evaluation of Oasis, an active storage framework for object-based storage systems that complies with the current T10 OSD standard. In contrast with previous work, Oasis has the following advantages. First, Oasis enables users to transparently process the OSD object and supports different processing granularity (from the single object to all the objects in the OSD) by extending the OSD object attribute page defined in the T10 OSD standard. Second, Oasis provides an easy and efficient way for users to manage the application functions in the OSD by using the existing OSD commands. Third, Oasis can authorize the execution of the application function in the OSD by enhancing the T10 OSD security protocol, allowing only authorized users to use the system.**

We evaluate the performance and scalability of our system implementation on Oasis by running three typical applications. The results indicate that active storage far outperforms the traditional object-based storage system in applications that filter data on the OSD. We also experiment with Java based applications and C based applications. Our experiments indicate that Java based applications may be bottlenecked for I/O-intensive applications, while for applications that do not heavily rely on the I/O operations, both Java based applications and C based applications achieve comparable performance. Our microbenchmarks indicate that Oasis implementation overhead is minimal compared to the Intel OSD reference implementation, between 1.2% to 5.9% for Read commands and 0.6% to 9.9% for Write commands.

I. Introduction

Recently, object-based storage [1], which combines the advantages of the data sharing and secure capabilities of NAS [22] with the high-speed, direct-access of SAN [21], has been the subject of extensive research and development in the storage area. Numerous prototype systems (e.g., Lustre [7], Panasas [6] and Ceph [23]) using object-based technology have been developed by the industrial R&D community. The object-based storage interface standard (also referred to as the T10 OSD standard [10]), which is being developed by the Object-based Storage Device (OSD) technical working group within the Storage Networking Industry Association (SNIA) and the INCITS T10 Technical Committee, has defined the basic command set for the SCSI object-based storage device. It aims to promote object-based storage technology and further increase the market share for object storage products.

On the other hand, numerous academic research institutions have made contributions to the T10 OSD standard by integrating various technologies, such as Quality of Service (QoS) [13] and storage security [12], into the standard. However, the existing T10 OSD standard does not sufficiently expose the intelligence/capabilities of object storage devices. Since object storage devices can manage the location of object data itself and the object has attributes that contain rich semantic information, the object storage devices have the potential to be much more intelligent than a storage device based on the traditional block interface.

In addition, active storage technology [2] [3] [4] has shown to be one of the most interesting approaches to express the intelligence of storage devices. By exploiting the processing power of the storage device, active storage is not only able to filter data and reduce the bandwidth requirement on the network, but also provide aggregation processing capabilities through the parallelism of the disks.

There have been several efforts to integrate active storage technology into the T10 OSD standard. Qin et al. [8] proposed a hybrid approach to scheduling code in object-based storage device to execute. Both John et al. [9] and Devulapalli et al. [24] presented an implementation of active storage framework for the T10 OSD standard. However, their implementations are preliminary, and do not validate their systems on a variety of data intensive applications, thus they cannot fully demonstrate the advantage of object-based technology. Besides, to make the OSD product involved in active storage framework widely accepted by OSD consumers, we believe that some important characteristics should be considered. Specifically, supporting transparent and multi-granularity processing, convenient management and security.

Earlier work has pointed out the importance of the above characteristics. For example, MVSS [4] has stated the necessity of transparent and multi-granularity processing. Security has been examined in the SRPC framework [31] and management has been discussed in the work on iOSD [24],

¶Corresponding Author, 978-1-4577-0428-4/11/$26.00

Load-managed Active Storage [32] and Lerna [26]. Basically, these characteristics will make OSDs easier to use and more sophisticated.

In this paper, we present the design and performance evaluation of an active storage framework called *Oasis* that integrates the above features and is based on the current T10 OSD standard. Specifically, Oasis has the following unique advantages.

First, Oasis frees users from needing to remember the details of *application functions* (application-specific code that can be downloaded and executed on the user data, e.g., compression, classification, etc) and enables users to transparently process the OSD object. In addition, Oasis supports different processing granularity (from the single object to all the objects in the OSD) by extending the OSD object attribute page defined in the T10 OSD standard.

Second, Oasis provides an easy and efficient way for users to manage application functions. Users can conveniently create, remove and list application functions in the OSD whenever they like by using existing OSD commands.

Third, as a preliminary security solution, Oasis can authorize the execution of the application function in the OSD by enhancing the T10 OSD security protocol, thus preventing unauthorized users from intentionally destroying the system.

We also evaluate the performance and scalability of Oasis by running three typical applications: Database Selection, Blowfish Decryption and Edge Detection. They are all representative of the data analysis applications in the real world and are widely used in various fields. We also examine the impact of the C and Java programming languages on the code execution efficiency, and evaluate our system implementation overhead by comparing Oasis with the Intel OSD reference implementation. Our work on design and evaluation on the standard provides an important reference to the OSD community.

The rest of the paper is organized as follows. We summarize background and related work in Section II and elaborate the design and implementation of Oasis in Section III. In Section IV, we evaluate the implementation of Oasis and discuss the results of running various applications on the Oasis prototype. In Section V, we conclude the paper and point out directions for future research.

II. Background and Related work

In this section, we first give an overview of object-based storage and the T10 OSD standard. Second, we present the related work on active storage and then motivate our research.

A. Object-Based Storage and the T10 OSD Standard

With the rapidly escalating storage requirements of enterprises, object-based storage [1] has emerged as one of the most promising technical solutions to next-generation storage systems in the past few years. It offloads storage management functions from the host operating system to the intelligent object-based storage device (OSD) that manages its own storage space and exports an expressive object interface. Object-based storage systems, such as Lustre [7], Panasas [6] and Ceph [23] that combine the advantages of NAS [22] and SAN [21], can provide high throughput, reliability, availability and scalability.

The object-based storage interface standard (also referred to as the T10 OSD standard) aims to promote the development of the object-based storage technology. A recent revision, i.e., the T10 OSD-2, was ratified by ANSI in January 2009. It defines the basic SCSI object-based storage device commands (e.g., READ and WRITE command) that are responsible for reading or writing object data from/to the OSD. Besides, it defines four kinds of objects, namely, root object, partition object, collection object and user object. Root, partition and collection objects are a great aid for addressing and retrieving user objects, which store user data (such as files and database records). Each object is identified by an object ID, and consists of data and attributes that express the specific characteristic of object (e.g., creation time, access time, etc). These attributes are organized into pages for identification and reference. Each attribute item in the attribute page can be indexed by a combination of attribute page number and attribute number. Moreover, this standard defines a capability-based security model. A client wishing to access to storage devices can acquire a capability from a metadata server and then presents it to the devices with the I/O requests, thus the storage devices can authorize the access to object data according to the capability.

B. Existing Active Storage Approaches

Active storage [2] [3] [4] [25] [26] [27] [28], which enables computation inside storage devices, has long been an important way to make device intelligent and optimize the system performance. In the earliest work, researchers developed various database machines [16] to increase the performance of database application by exploiting the processing power within the disk arm. These machines failed to gain wide acceptance as they used non-commodity hardware and the performance gains were limited. With the development of the VLSI (Very Large Scale Integrated circuit) technology that makes it possible for the disk drive to have more powerful processing capability, researchers proposed the active disk project [2] [3] to re-examine the database machine work. By partitioning applications (e.g., data mining, image processing) into the host and disk portions, the Active Disk system is able to obtain higher throughput and less response time. In response to the storage and computational demand for DSS (Decision Support Systems) and data warehousing workloads, Keeton et al. [17] presented a computer architecture that utilizes "intelligent" disks, which exploit the low-cost embedded processing capability and improve cost-performance by offloading general-purpose computation from expensive desktop processors. The MapReduce [29] [30] software framework also employs a concept similar to active storage. It splits a large data set into many pieces and distributes them into many commodity-hardware computers that then process the data locally, and merges the results into the output. The recent work [28] also

provides an approach to deploying active storage technology on parallel I/O software stack by extending the MPI-IO interface.

The above work is built on the storage systems based on the block-level interface. Since object-based storage technology may be the next wave in the storage field, a lot of studies have gradually focused on building the active storage system on object-based storage platforms. Huston et al. [18] presented *diamond*, an active storage architecture designed to address the issue of searching non-indexed data from the massive storage system. This system uses the concept of object-based storage, such as object attributes, to perform semantic filter processing in the device. Piernas et al. [5] presented an active storage framework for Lustre [7], which is implemented in user space and proves to be faster and more portable than the previous kernel-space version [33]. However, these two systems are not designed for the general object-based storage platform and do not comply with the T10 OSD standard.

Our work is closely related to Qin et al.'s [8], John et al.'s [9] and Devulapalli et al.'s [24]. They have also built their active storage frameworks on the T10 OSD standard. However, their evaluation is not comprehensive. In addition, few of them have taken into account transparent and multi-granularity processing, flexible management of the application function or concrete methods to enable the security of execution.

Recently, the SNIA OSD committee devoted themselves to add OSD intelligence into the third version of T10 OSD standard (i.e., T10 OSD-3). Researchers from the University of Connecticut submitted an active storage proposal to the OSD committee. The proposal elaborates how to enable the execution of remote methods in a virtual machine environment in object-based storage devices. The implementation and evaluation of the standard in our paper is based on our proposal that defines an extended function object, object commands and attribute pages to support active storage in OSD and has been submitted to the SNIA OSD technical working group.

III. Oasis Design and Implementation

In this section, first we will state the design objective of Oasis in terms of user case. Then we will elaborate the details on design and implementations of Oasis.

A. Design Objective and User Case

Our intuitive design objective mainly comes from the user requirements of transparent and multi-granularity processing, ease of management and security.

a) Transparent Processing

In most cases, a user of an OSD product doesn't wish to remember the details of application functions: how these application functions are programmed, the execution parameters or the identifiers of the application functions. This makes it hard for the user to explicitly schedule the application function to execute.

To enable transparent processing, we have to associate application functions with OSD objects and store this association information into the OSD system, so that when the OSD object is accessed (e.g., READ or WRITE), the associated application function can be automatically invoked to execute.

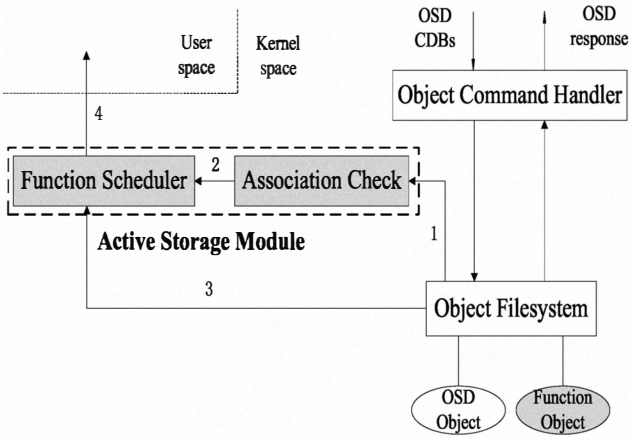


Fig. 1. Architecture of Oasis

b) Multi-granularity Processing

A user might want to encrypt anything from a file to a whole directory that contains hundreds of thousands of files. In many cases, encrypting each file one by one is not an efficient solution. In the OSD, we aim to utilize the existing four kinds of OSD objects (i.e., root, partition, collection and user object) that represent different granularity to enable multi-granularity processing.

c) Management

Management policy on previous work [24] [26] [32] has focused on mapping computation load to available hardware resource in terms of system performance. In addition to that, we believe users need an interface for managing application functions associated with OSD objects. For example, the user may download a compression algorithm to the OSD if he wants a file be compressed on the OSD, and may want to remove the algorithm from the OSD to save the storage space once it is no longer necessary. Sometimes, the user may just be curious about which application functions exist on the OSD. Our objective is to utilize existing OSD commands to enable efficient management.

d) Security

The execution of an application function must be safe. The reason is obvious, unsafe execution can result in the wrong results, e.g, a program accesses an invalid memory space. In addition, a user may not want the code that he downloaded to the OSD be used by other users, so access control should be considered. We will give a preliminary security solution to deal with these problems in Section III-F.

B. Architecture Overview

Figure 1 shows the architecture of Oasis in an OSD device. As depicted in Figure 1, Oasis consists of four modules,

TABLE I
THE EXTENDED PARTITION_ID AND USER_OBJECT_ID VALUE TO THE T10 OSD STANDARD

Partition_ID	User_Object_ID,Function_Object_ID or Collection_Object_ID	Description
0h	0h	Root object
0h	1h FFFF FFFF FFFF FFFFh	Reserved
1h to FFFFh	0h- FFFF h	Reserved
1h to FFFFh	**10000h to FFFF FFFF FFFF FFFFh**	**Function Object**
10000h to FFFF FFFF FFFF FFFFh	0h	Partition
	1h-0FFFh	Reserved
	1000h to BFFFh	Well known collections
	C000h to FFFFh	Reserved
10000h to FFFF FFFF FFFF FFFFh	10000h to FFFF FFFF FFFF FFFFh	Collection or User object

TABLE II
THE EXTENDED USER OBJECT INFORMATION ATTRIBUTES PAGE CONTENTS

Attribute Number	Length (bytes)	Attribute	Application Client Settable	OSD Logical Unit
0h	40	Page identification	No	Yes
1h	8	Partition_ID	No	Yes
2h	8	User_object_ID	No	Yes
3h	**8**	**Function_Object_ID**	**No**	**Yes**
4h	**8**	**Parameter**	**Yes**	**No**
5h to 8h		Reserved	No	Yes
9h	variable	Username	Yes	No
10h to FFFF FFFEh	...	...	...	...

namely, the Object Command Handler, the Object Filesystem, the Association Check and the Function Scheduler. *Object Command Handler* gets and analyzes OSD commands and forwards them to the Object Filesystem that is responsible for reading and writing the object data, as well as performing the management of OSD objects and function objects that represent the offloaded application functions. In this process, Oasis utilizes the commands that are applied to user objects to manage function objects only by specifying the partition identifier of the function objects, thus users (or administrators) can easily and conveniently perform the management of function objects without needing to add new commands to the existing T10 OSD command set or modify the current T10 OSD standard. *Association Check* is responsible for checking whether there exists any function object associated with the OSD object that is being read or written, reading the function object ID and parameters from the OSD objects' attributes if the association exists, and then passing these information to the *Function Scheduler*. Associating the function object with the OSD object provides a flexible and efficient approach to invoking the function object to execute and transparently process the OSD object during the read or write process. *Function Scheduler* is responsible for scheduling the related function objects to execute according to the function object ID and parameters acquired from the *Association Check*. Currently, the *Function Scheduler* performs schedule work on a first come first serve basis. However, when two different function objects need to be scheduled to execute at the same time, a flexible and efficient scheduling scheme is a must. We plan to employ sandbox technology similar to iOSD [24] to solve this problem in our future work. Oasis provides access control for the execution of the function objects by simply extending the Permission Bit Mask of the capability (see Section III-F). The function objects can be executed on the virtual machine in the user space and the execution results will be written to the local disk or returned to the client.

C. Function Object

According to the current T10 OSD standard, the four kinds of defined OSD objects are either used for storing user data (i.e., user object) or used for addressing and retrieving user data (i.e., root object, partition object and collection object). The function object is suggested to hold the offloaded application function (e.g., compression, classification, etc).

Similar to the existing four kinds of objects, a function object is identified by a function object ID and is located in a dedicated partition that is identified by a partition_ID set to 1h-FFFFh (see Table I). Besides, a function object contains attributes that describe the basic information of the function object (e.g., creation time, access time, etc). All the function objects are motivated to be executed in OSD to perform operations or analyses on user objects. A function object can be written using the C programming language or a cross-platform language such as Tcl/Python script or JAVA, and the OSD needs to implement the script interpreter or virtual machine to execute the corresponding functions. Figure 2 illustrates a piece of C and Java code for a function object that performs data filtering respectively. Both of them retrieve the data from the input stream, process the data and pass the result to the output stream. Except for the input stream and output stream, such as a file, a buffer or a pipe, there is no other way to communicate with the outside operation system.

Taking the application function as a kind of object provides the following benefits. First, similar to the user objects, we

```c
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
char ch[50];
long int content=0;
FILE *instream;
FILE *outstream;

if((outstream=fopen(argv[2], "w+" ))==NULL)
{
printf( "error" );
return -1;
}
if((instream=fopen(argv[1], "r" ))!=NULL)
{
do{
    if(fgets(ch,50,instream)==NULL)
      break;
    if(strcmp(ch, "\n" )==0)
      continue;
   else
     {
     content=atoi(ch);
     if(content<100)
       fprintf(outstream, "%s" ,ch);
}
}while(!feof(instream));
}
fclose(instream);
fclose(outstream);
return 0;
}
```

(a) The function object using C code

```java
import java.io.*;
import java.lang.Integer;
import java.awt.*;

Public class sort
{
Public static void main(String[] args)throws IOException{

File inputFile = new File(args[0]);
File outputFile = new File(args[1]);
FileReader data = new FileReader (inputFile);
FileWriter result = new FileWriter (outputFile);

BufferedReader br = new BufferedReader (data);
String s;
int i;

While ((s = br.readlint())!=null){
       if(s.length()!=0){
        i= Integer.parseInt(s);
         if(i<100){
          result.write(s);
          s= "\n" ;
         resule.write(s);
        }
    }
}

data.close();
result.close();

}
}
```

(b) The function object using Java code

Fig. 2. C and Java code for a function object that performs data filtering

can use the current object commands defined in the T10 OSD standard, such as the CREATE AND WRITE, REMOVE and LIST commands, to manage the function objects, which makes it more convenient to download, schedule and execute the application function. Second, the attributes of the function object provides a unique and effective way for users to understand and use the application function. Third, we can utilize the object-based storage security model to authorize the execution of the application function. We will elaborate these advantages in the following sections.

D. Association

Oasis allows users to associate a function object with an OSD object by saving the function object's ID and its parameters (e.g., encryption keys) in the OSD objects' attributes that are organized into attributes pages for identification and reference in the T10 OSD standard. Table II illustrates the extended User Object Information attribute page that contains a function object's ID and its parameters. By building the association through using the objects' attributes, the user can easily set and retrieve the association information by using OSD commands. For example, the SET ATTRIBUTES command in the current T10 OSD standard allows setting an attribute value in the OSD object information attribute page.

In addition, such an association design gains several salient advantages. First, the association operation makes it possible to invoke function objects to execute during the read or write process to OSD objects, thus making the data processing in the device completely transparent to the user. Second, this approach provides a simple and convenient way for users to flexibly apply different application functions to different kinds of files. For example, the user can apply an edge detection algorithm to an image file to acquire the edge feature of the image by associating the function object that represents the edge detection algorithm with the user object that represents the image file, while for the database file that contains millions of records, the user can apply an efficient database query to it by associating the function object that represents the database query with the user object that represents the database file. Third, associating function objects with different kinds of OSD objects can support different processing granularity. For example, associating a function object with the root object will affect the whole OSD logical unit, while associating a function object with a partition object will affect all the sub-partitions and files in it. This allows users to take a flexible and wide range of granularity according to different applications. For example, some applications aim at the handling of a single file, while other applications aim at a large-scale scan processing.

In addition, when more than one application function needs to be applied to the user data, for example, a file needing to be compressed first and then encrypted, users can associate multiple function objects with a single user object at a time. The execution order among them is determined by the Attribute Number. The function object identified by the

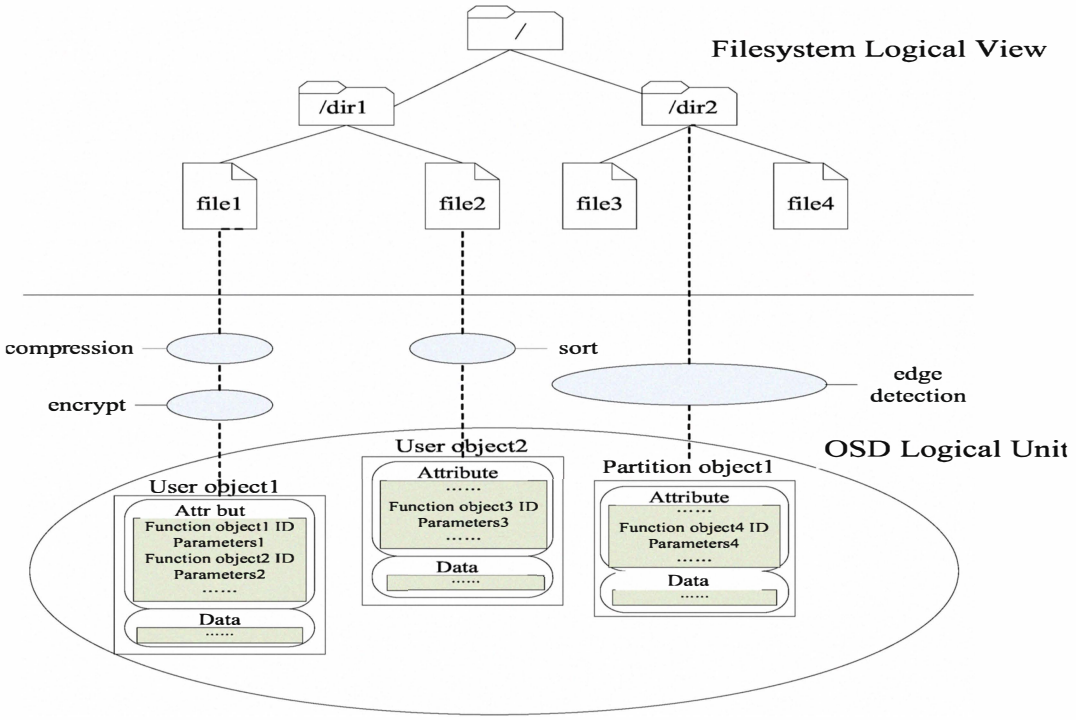


Fig. 3. Mapping from file to object and an example of the association between function objects and OSD objects in Oasis

Function_Object_ID with the smaller Attribute Number is executed first.

Figure 3 shows an example of the association between function objects and OSD objects in Oasis. As the figure shows, file1 is mapped to user object1 and file2 is mapped to user object2, while the directory /dir2 is mapped to partition object1. Associating the edge detection operation with partition object1 will make all user objects mapped from image files under /dir2 (such as file3 and file4) be processed by the edge detection function using parameters4. Associating the sort operation with user object2 will make user object2 mapped from file2 be processed by the sort function using parameters3. For user object1, two kinds of operations, the compression and encryption that are identified by function object1's and function object2's ID respectively have been associated with it. This will make user object1 be compressed first using parameters1 and then be encrypted using parameters2 in the write process, while the reverse processing steps will occur in the read process.

E. Management of Function Objects

For an administrator or even ordinary users, they are usually concerned with the following issues:

1) How to simply and flexibly download an application function to the storage device?

2) How to easily remove an application function from the storage device?

3) How to conveniently view which application functions are there in the storage device?

Since an application function is taken as a function object in the device and is essentially a data stream similar to a user object, Oasis uses the commands (such as CREATE AND WRITE, REMOVE and LIST), that are applied to user objects to manage the function object. The difference is that, as Oasis uses a dedicated partition to store all the function objects, thus all the commands directed to the objects with the same partition identifier as the dedicated partition that holds the function objects will operate these function objects.

Downloading an application function to storage devices will result in unpredictable risks, which will be discussed in Section III-F. Removing an application function from an object-based storage device helps reclaim space from function objects that are no longer used. Through sending the LIST command to an OSD, users can acquire a list of function objects, including the object' ID and attributes. This provides a convenient way for users to understand and make use of the function object and for administrator to better manage the function object. In fact, in order to confirm which function objects are associated with a specific OSD object, users can retrieve the corresponding OSD object attribute page through the GET ATTRIBUTES command that is defined in the T10 OSD standard.

As a result, practitioners/designers even do not have to add a new command, thus reducing the modification to the existing object file system and facilitating the implementation of active storage system on OSD platform. In addition, storing function objects in a dedicated partition makes them easy to find and access.

TABLE III
PERMISSIONS BIT MASK FORMAT

Bit	7	6	5	4	3	2	1	0
49	READ	WRITE	GET_ATTR	SET_ATTR	CREATE	REMOVE	OBJ_MGMT	APPEND
50	DEV_MGMT	GLOBAL	POL/SEC	M_OBJECT	QUERY	GBL_REM	FUN_EXE	Reserved

F. Preliminary Security Considerations

Executing a function object in an OSD can raise serious security risks, which are mainly caused by two aspects: bad code in function objects and illegal users who execute the function objects.

To ensure the function objects to be downloaded are well-written and secure, Oasis only allows the function objects developed by the OSD vendor to be downloaded since the vendor has professional knowledge and tools to write and validate the code. For common users, they only need to know what functions a function object can provide (such as, compression, backup, etc) and decide whether to use this function, but do not need to consider the implementation details on this function.

Some malicious users may modify the code that is created by the vendor and then download it to the OSD. We are trying to apply a security model that employs the public&private key to solve this problem. In one possible security solution, the vendor encrypts every section of code with a private key, and both users and OSDs have a public key to validate the code. The malicious users can access the code using a public key and modify it. But since they do not have the private key, they cannot encrypt the data. Hence they cannot get past the OSD's validation check. This prevents the users from downloading erroneous or unsafe code that may damage the OSD system.

Besides, Oasis can use the OSD security model defined in the T10 OSD standard to authorize the execution of the function object by simply adding a FUN_EXE bit (see Table III) to the Permissions Bit Mask field in the capability. Similar to the READ and WRITE bits that provide access control for common data read and write operations, the FUN_EXE bit provides access control for the execution of the application functions that have been migrated to the object-based storage device. A FUN_EXE bit set to one allows the function object to be executed on a user object, while a FUN_EXE bit set to zero prohibits the execution of the function object on a user object. A client wishing to access an OSD, requests such an extended capability from a metadata sever and sends it to the OSD as the part of the command. The OSD can then use this capability to authorize the execution of the function object, thus efficiently preventing unauthorized users from intentionally destroying the system. An obviously advantage of such an approach is that it is so simple that it requires minimal changes to the current T10 OSD security mechanism.

G. Additional Considerations

In addition to the case that multiple function objects can be associated with a user object, a function object can be also associated with a number of user objects at a time. When multiple clients request the same function object to process different user objects at the same time, the function object can be executed in different address space.

Moreover, a function object can be used as a system parser, but not specific to a certain OSD object. In this case, the function object can monitor system resources, data traffic, as well as key features such as object attributes. This will be helpful to designing self-adaptive system. By extending the semantics of the function object, the object-based storage system will become more intelligent.

H. Implementation Details

We prototyped Oasis (Object-based Active Storage System) on the Intel OSD reference implementation (REFv20) [11] which includes an initiator on the host side and a target on the OSD side (for one OSD). The initiator contains an OSD file system (OSDFS), an upper level OSD driver and an iSCSI device driver, and communicates with the targets on the OSDs through OSD commands. All the files and directories are stored as objects in the OSDs.

We implemented the function object in C and Java programming language. Both the C and Java code are compiled first before they are downloaded to the OSD. Upon receiving a piece of C or Java code, the OSD automatically converts them to function objects, and then assigns a function object ID for each function object. In accordance with the association approach outlined in Section III-D, they will be scheduled during the read or write process. In our system, we apply a Java virtual machine in the Linux operating system platform. Once the function object is scheduled, the java byte code will be interpreted to run. In the Section IV-C, we will specifically explore the execution efficiency of these two kinds of code by running tests on a variety of applications, in other words, to what extent will adopting a Java virtual machine affect the code execution efficiency though it enables the portability when compared to C.

IV. EVALUATION

In this section, we'll first evaluate the performance of Oasis through three kinds of widely used data analysis applications, i.e., Database Selection, Edge Detection and Blowfish Decryption respectively, and then analyze the overhead in building an Oasis system from the aspects of system implementation and management.

A. Experimental Setup

Our experiment test bed consisted of a host (or client) and 1, 2 or 4 OSDs. All of these nodes have the same hardware components, each with one Intel 604-pin EM64T

TABLE IV
CHARACTERISTICS OF DATA ANALYSIS APPLICATIONS

Name	Description	Input Data	% of Data Filtering
Database Selection	Non-index select operation that applies to the entire dataset and returns the records that match a given search condition.	1.77GB(33 million line records, each of which is a double.)	87.4%
Edge Detection	This application employs sobel edge detection algorithm [14] to perform convolution operation on entire images and extract the key features (i.e., edge) of them.	584.0MB(10000 images, each of which is a 8-bit map of 59.8k)	96.7%
Blowfish Decryption	This application employs the blowfish algorithm developed by Bruce Schneier [20] to decrypt an 8-byte record each time.	800MB(100 million line records)	0

Xeon 3.0 GHz processor, 512MB PC2700 DDR-SDRAM physical memory and a 250GB disk. The host and OSDs are connected via 1Gbps Ethernet. All of these machines run RedHat Linux 2.4.20.

B. Methodology and Workload

We evaluate the performance of Oasis by running three applications shown in Table IV. We choose these applications because they are representative of the data analysis applications in the real world and are widely used in various fields. For example, Database Selection is one of the most important query operations in the database system that apply to the entire dataset and return only Detection is an image processing algorithm that detects the edges or corners of "objects" in a scene(e.g., this application can detect the facial features of individuals in an image). Blowfish Decryption is an encryption algorithm that decrypts data in 8-byte blocks and is widely used in software such as SSH and in operating systems such as OpenBSD. We briefly describe these applications in the second column of Table IV. It should be noted that, we do not currently employ a real database in the OSD system. Instead, we have developed a filter applet that filters the records according to a certain degree of selectivity (e.g., applying a filter applet with a selectivity factor of ten to the dataset will return 1/10 of the total amount of data) to simulate the non-index operation that applies to the entire dataset. The third column shows the specified dataset used in each application. In the last column, we give how much percent of the input data set would be filtered in the OSD-side. For example, Edge Detection shows the maximum amount of data filtering of 96.7%, while Blowfish Decryption doesn't filter any data.

In an OSD, all of these three data analysis algorithms are encapsulated into function objects and have their object IDs for reference. For the dataset, both the database records and encryption items are contained in a file that is striped into OSD objects across all the OSDs. All the images to be processed are also evenly distributed across all the OSDs and each image in the OSDs is taken as an OSD object. We begin our test by running several processes on the host at the same time, and each process is responsible for reading or writing the OSD objects in an OSD. The function objects will be invoked to execute if they are already associated with the OSD objects that are being read or written. They acquire the execution parameters (e.g., encryption keys and selection conditions) from the attributes of the OSD objects and are executed in the user space to avoid disturbing the system kernel.

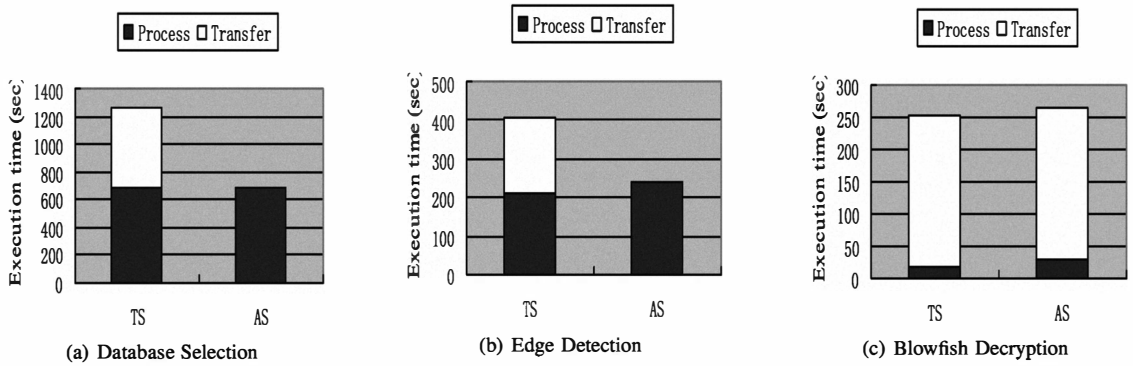
C. Application Performance

We evaluate the performance of Oasis by first analyzing the improvement on overall execution time, and then we look at the sensitivity analysis results, including the number of OSDs, the programming language of function objects, and the number of function objects that are associated with the same OSD object. We'll mainly analyze two cases: Traditional Storage (TS) and Active Storage (AS). The former means that the data stored in an OSD should be shipped to the host to process, while the latter means that the data should be processed in the OSD using the function object. It should be noted that, in the following experiments, all the executed function objects are implemented in C language unless otherwise indicated (e.g., in Figure 6).

a) Performance Improvement

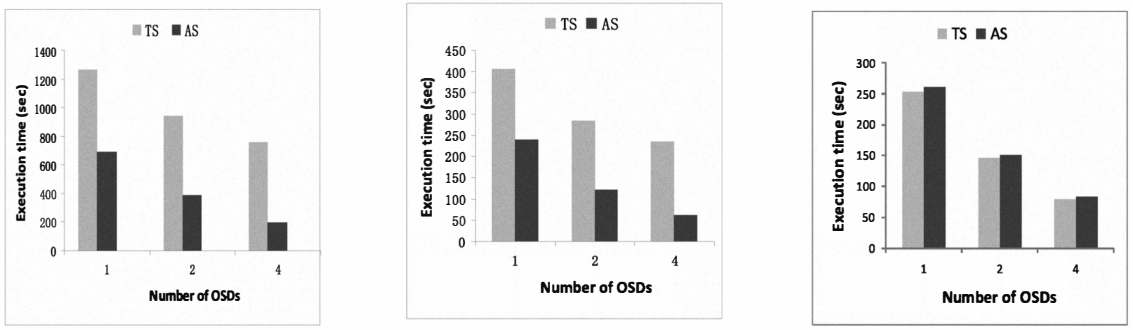
Figure 4 shows the execution time breakdown for different applications using one host and one OSD. We simply divide the execution time into a process part and a transfer part. The process part indicates the execution time of the application function, including the overhead of copying data from kernel space to user space, while the transfer part measures the communication overhead on the interconnect network between host and OSD. As the figure shows, the AS scheme improves performance significantly on both Database Selection and Edge Detection applications, 83.8% and 70.4% respectively. One can see that these improvements can be attributed to the dramatic time reduction on the Transfer part, from 292.6s to 3.9s in Database Selection and from 198.0s to 0.74s in Edge Detection application. This is because most of the data has been filtered on the OSD-side, which results in less data transferred from the OSD to the host. While, the performance of the AS scheme and the TS scheme in Blowfish Decryption are almost the same, the reason is that Blowfish Decryption algorithm doesn't filter data on the OSD-side (see Table IV).

We then look at the scalability of Oasis when the number of OSDs is increased. Figure 5 shows the performance of three applications in Oasis for the 1-OSD, 2-OSD and 4-OSD configurations, respectively. One can see that the performance



(a) Database Selection (b) Edge Detection (c) Blowfish Decryption

Fig. 4. Execution time breakdown for different applications with one OSD



(a) Database Selection (b) Edge Detection (c) Blowfish Decryption

Fig. 5. Execution time for different applications with different number of OSDs

of TS is scalable with the increase in the number of OSDs for all of the three applications. This is because the parallelism in the OSDs results in a great decrease in the transmission time over the interconnect network. We also observe that, in Figure 5(a) and Figure 5(b), the AS scheme outperforms the TS scheme significantly due to the reduction in data transfer, even with a single active storage node. We see that these improvements are consistent with the increase in the number of OSDs. For Blowfish Decryption application (see Figure 5(c)), as there exists no data reduction in the data transfer, AS scheme achieves comparable performance with the TS scheme even though the number of OSDs increases.

b) Impact of Language of Function Object

The C-powered function object may perform well in execution efficiency, but not well in portability when compared to a Java-powered function object. To analyze the impact of language of function object, we repeated the experiments with all three kinds of applications in three cases: processing data in host (TS), executing function objects written in C language in OSD (AS(C)) and executing function objects written in Java language in OSD (AS(Java)).

As illustrated in Figure 6, AS(C) and AS(Java) achieve comparable performance for both the Database Selection and Blowfish Decryption applications, while for Edge Detection, AS(C) far outperforms AS(Java) by a factor of 6.12. It should be noted that TS in the Edge Detection implementation also outperforms AS(Java) by a factor of 0.84. The reason for this is that, since a large number of I/O operations are required for the Edge Detection algorithm to generate the output image, the algorithm implementation using the Java language is significantly slower than the implementation using the C language. And even such performance degradation with the Java implementation may compromise the benefits of data reduction in the Edge Detection application achieved by the active storage technology.

However, for the application such as Blowfish Decryption, the algorithm is basically composed of the ADD and XOR instruction (not I/O bound), so the algorithms using C language and using java language will result in a comparable speed, implying that for non-I/O intensive applications, both the C and Java implementations of function objects can achieve comparable performance, while for I/O intensive applications, achieving a cross-platform implementation with the Java programming language means a potential performance bottleneck in the active storage system.

c) Impact of Multiple Function Objects

The above evaluation focuses on applying one application function on user data each time. However, sometimes, users may want to perform multiple operations on user data at a time. For example, users may need to first decrypt a large piece of data and then select the data that they want. Oasis supports function composition by associating multiple function

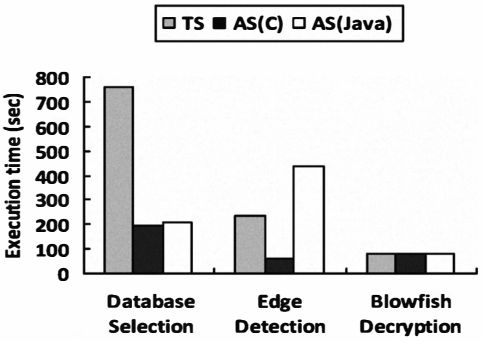


Fig. 6. Execution time for all three kinds of applications in the C and Java programming language. AS(C) means that the executed function object is implemented in C language, while AS(Java) means that the executed function object is implemented in Java language. In this experiment for all three applications, we use four OSDs for execution.

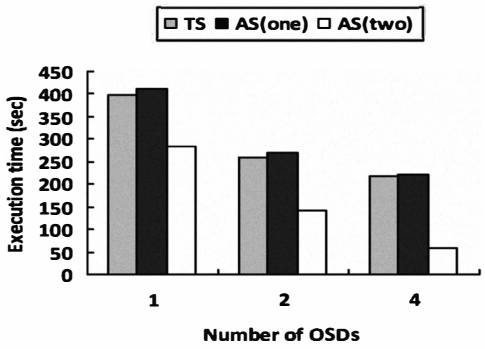


Fig. 7. Execution time on a hybrid application that stacks a Database Selection service on a Blowfish Decryption service with different number of OSDs. We analyze three cases: processing this hybrid application on host (TS), first decryption on OSD and then selection on host (AS(one)) and processing this hybrid application on OSDs (AS(two)).

objects with a single OSD object using object attributes (see Figure 3).

Figure 7 shows the execution time on a hybrid application that stacks a Database Selection service on a Blowfish Decryption service with different number of OSDs. We evaluate the impact of multiple function objects by partitioning this hybrid application between the host and OSDs, namely, processing this hybrid application on host (TS), first decryption on OSD and then selection on host (AS(one)) and processing this hybrid application on OSDs (AS(two)). The results show that AS(one) does not improve the system performance, as a matter of fact, decreases slightly by 0.7%-3.9% when compared to TS. The reason is that, offloading the Blowfish Decryption application to the OSDs doesn't bring data reduction across I/O interconnect, but incurs a small overhead over the traditional object storage system. However, the performance of AS(two) significantly outperforms TS by a factor of 0.41 to 2.63, and also outperforms AS(one) by a factor of 0.45 to 2.66. This shows that offloading Database Selection to the OSDs can significantly improve the system performance. Again, this is because the Database Selection application reduces the data needing to transmit over the interconnect network by filtering data on the OSD side. This indicates that, for a hybrid application that is composed of multiply applications, only applications that can make data reduction across the I/O interconnect can really benefit system performance.

D. Overhead Analysis

a) Implementation Overhead

As depicted in Figure 1, in an Oasis system, the *Association Check* module has to check whether there exists any function object associated with the OSD object that is being read or written by accessing the attributes of the OSD object during every read or write call even when no function object is associated with the OSD object. We evaluate this implementation overhead by comparing the completion time of reading and writing a file with different file sizes in the Oasis implementation when no function object is associated to the user object with the Intel OSD reference implementation under 1Gbps interconnect network. As shown in Figure 8, the implementation overhead of Oasis is minimal, between 1.2% to 5.9% for the read operation and 0.6% to 9.9% for the write operation, with the Intel OSD reference implementation as the baseline.

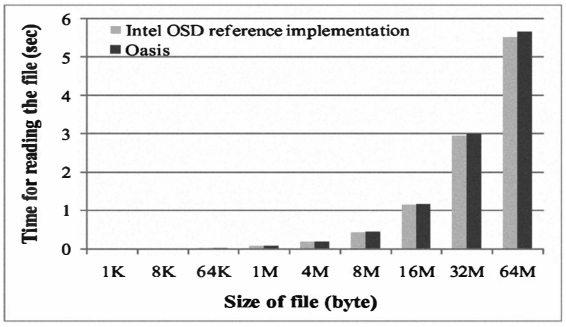
b) Management Overhead

Oasis manages the function object by cleverly employing the object commands defined in the current T10 OSD standard. Table V shows the completion time of various object commands for the management of function objects in Oasis under 100 Mbps interconnect network. For example, it takes 13.6 ms to create a function object with 1KB size by using the CREATE AND WRITE command, while only 7.8 ms to delete this function object by using the REMOVE command. In summary, it incurs an overhead as little as 2.8 ms to 13.6 ms for managing function objects, implying that Oasis provides an effective and time-saving way to manage the function objects by using the existing object commands.

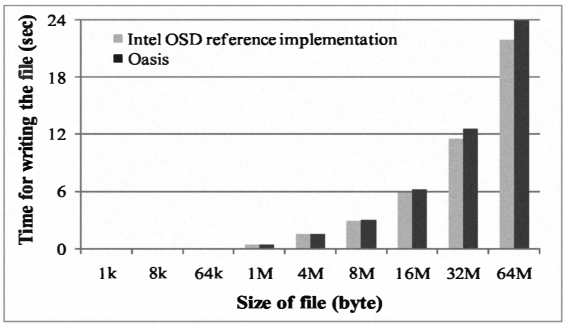
V. Conclusions and Future Work

In this paper, we presented the design and evaluation of Oasis, a framework that incorporates the active storage technology into object-based storage systems that comply seamlessly with the T10 OSD standard. In contrast with previous work, our design represents application functions in the OSD as function objects, allows them to be flexibly controlled by using the standard OSD commands, and supports transparently and variable-granularity processing by using object attributes. In addition, Oasis also supports capability-based access control by extending the object storage security model.

We prototyped Oasis on the Intel OSD reference implementation and implemented function objects in the C and Java programming languages, respectively. Experimental results on data analysis application show that Oasis achieves substantial performance improvements over the traditional object-based



(a) read overhead



(b) write overhead

Fig. 8. Implementation overhead of Oasis over the Intel OSD reference implementation

TABLE V
COMMANDS OVERHEAD FOR MANAGING FUNCTION OBJECT

Number	Management Description	Object Commands	Completion Time (ms)
1	Create a 1KB function object	CREATE AND WRITE	13.6
2	Associate a function object	SET ATTRIBUTES	2.8
3	retrieve a 1KB association information	GET ATTRIBUTES	12.1
4	List 512bytes function objects	LIST	4.5
5	Delete a 1KB function object	REMOVE	7.8

storage system in Database Selection and Edge Detection applications, while Blowfish Decryption doesn't achieve dramatic performance improvement as it doesn't filter data on the OSD-side. The data also shows that, though Java-powered code makes it possible for the application function to execute in a cross-platform environment, it may result in a system performance bottleneck with I/O-intensive applications, while for applications that do not heavily rely on the I/O operations, implementing the function object with both the Java and C programming languages will achieve comparable performance. We also evaluated the overhead of Oasis and demonstrated that, compared to the Intel OSD reference implementation, Oasis brings only a small implementation overhead, as little as 1.2% to 5.9% for read and 0.6% to 9.9% for write. Moreover, Oasis provides an effective and time-saving way to manage function objects.

In future work, we would like to employ sandbox technology [24] to enable the concurrent execution of multi-function objects. We would also like to evaluate the cases when host or OSDs are heavily loaded. In this case, we would like to employ the dynamic partition approach [18] to efficiently allocate computation workload between host and OSDs according to their processing capability.

ACKNOWLEDGMENTS

We would like to thank DJ Capelis, Ian Adams and anonymous reviewers for their valuable comments on this paper. We also thank Hong Jiang, Lei Tian, Yanli Yuan, Quanli Gui, Chengtao Lu, Yang Hu, Shuibing He, Wenhua Zhang and Tian Zan for their help in writing this paper. This work was supported in part by the National Basic Research 973 Program of China under Grant No. 2011CB302301, 863 project 2009AA01A402, NSFC No. 61025008, 60933002, 60873028, Changjiang innovative group of Education of China No. IRT0725. This work was also supported in part by the National Science Foundation under award IIP-0934401. We also thank the sponsors of the SSRC and CRIS, including the National Science Foundation, Los Almos National Laboratory, LSI, IBM Research, NetApp, Samsung Information Systems America, Seagate Technology, Northrop Grumman, Symantec, Hitachi, CITRIS, the Department of Energy Office of Science, the NASA Ames Research Center and Xyratex.

REFERENCES

[1] M. Mesnier, G. R. Ganger, E. Riedel. Object-based Storage. *IEEE Communications Magazine*, Vol. 41, No. 8, pp. 84-91,2003.
[2] A. Acharya, M. Auysal, and J. Saltz. Active disks: Programming model, algorithms and evaluation. In *Proc. of 8th international conference on Architectural support for programming languages and operating systems(ASPLOS)*, 1998.
[3] E. Riedel, G. Gibson, and C. Faloutsos. Active storage for large-scale data mining and multimedia. In *Proc. of the 24th International Conference on Very Large Data Bases(VLDB)*, 1998.
[4] X. Ma, A.L. N. Reddy. MVSS: an active storage architecture. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, Issue 10, pp. 993 - 1005,2003.
[5] J. Piernas, J. Nieplocha, E. J. Felix. Evaluation of Active Storage Strategies for the Lustre Parallel File System. In *Proc. of SC '07*, Nov. 2007.
[6] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, B. Zhou. Scalable Performance of the Panasas Parallel File System. In *Proc. of FAST '08*,2008.
[7] Lustre. http://www.lustre.org.
[8] L. Qin, D. Feng. Active Storage Framework for Object-based Storage Device. In *Proc. of the 20th International Conference on Advanced Information Networking and Applications*, Apr. 2006.
[9] T. M. John, A. T. Ramani, J. A. Chandy. Active storage using object-based devices. In *Proc. of the second International Workshop on High Performance I/O Systems and Data Intensive Computing*,2008.
[10] SCSI Object-Based Storage Device Commands -2 (OSD-2). Project T10/1729-D, Revision 5. T10 Technical Committee, INCITS, Jan. 2009.
[11] Intel Corporation. Intel iSCSI Reference Implementation. http://sourceforge.net/projects/intel-iscsi

[12] Z. Niu, K. Zhou, D. Feng, H. Jiang, F. Wang, H. Chai, W. Xiao, C. Li. Implementing and Evaluating Security Controls for an Object-Based Storage System. In *Proc. of the 24th IEEE Conference on Mass Storage Systems and Technologies*,2007.
[13] Y. Lu, D. H. C. Du, T. Ruwart. QoS Provisioning Framework for an OSD-based Storage System. In *Proc. of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies*,2005.
[14] http://www.pages.drexel.edu/~weg22/edge.html.
[15] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient Software-Based Fault Isolation. In *Proc. of the 14th ACM Symposium on Operating System Principles*,Dec. 1993.
[16] D. J. Dewitt and P. Hawthorn. A Performance Evaluation of Database Machine Architectures. In *Proc. of the 7th International conference on VLDB*,Sep. 1981.
[17] K. Keeton, D. A. Patterson, and J. M. Hellerstein. The Case for Intelligent Disks (IDISKs). *SIGMOD Record*, vol. 27, no. 3, pp. 42-51, Sep. 1998.
[18] L. Huston, R. Sukthankar, R. Wickremesinghe, M. Satyanarayanan, G. R. Ganger, E. Riedel, A. Ailamaki. Diamond: A Storage Architecture for Early Discard in Interactive Search. In *Proc. of the 3rd USENIX Conference on File and Storage Technologies (FAST '04)*,2004.
[19] G. C. Necula, and P. Lee, Safe Kernel Extensions Without Run-Time Checking. In *Proc.of OSDI'96*, Oct. 1996.
[20] http://www.schneier.com/code/bfsh-koc.zip
[21] T. Clark. Designing Storage Area Networks: A Practical Reference for Implementing Fibre Channel and IP SANS. Second Edition. Addison Wesley, 2003.
[22] D. F. Nagle, G. R. Ganger, J. Butler, G. Goodson, and C. Sabol. Network Support for Network-Attached Storage. In *Proc. of Hot Interconnects 1999*, Aug. 1999.
[23] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long. Ceph: A Scalable, High-Performance Distributed File System. In *Proc.of OSDI'06*, 2006.
[24] A. Devulapalli, I. T. Murugandi, D. Xu, P. Wyckoff. Design of an Intelligent Object-based Storage Device. http://www.osc.edu/research/network_file/proje- cts/object/papers/istor-tr.pdf
[25] C. W. Smullen, S. R. Tarapore, S. Gurumurthi, P. Ranganathan, M. Uysal. Active Storage Revisited: The Case for Power and Performance for Unstructured Data Processing Applications. *Proc. of the 5th conference on computing frontiers*,Page(s):293-304,2008
[26] S. V. Anastasiadis, R. G. Wickremesinghe, J. S. Chase. Lerna: An Active Storage Framework for Flexible Data Access and Management. *Proc. of the 14th IEEE International Symposium on High Performance Distributed Computing*, Page(s):176-187,2005
[27] Y. Zhang, D. Feng. An Active Storage System for High Performance Computing. *Proc. of the 22nd International Conference on Advanced Information Networking and Applications*, Page(s):644-651,2008
[28] S. W. Son, S. Lang, P. Carns, R. Ross, R. Thakur, B. Ozisikyilmaz, P. Kumar, W. K. Liao, A. Choudhary. Enabling Active Storage on Parallel I/O Software Stacks. *Proc. of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*,Page(s):1-12,2010
[29] J. Dean and S. Ghemawat. Mapreduce: Simplified Data Processing on Large Clusters. *Proc. of the USENIX Symposium on Operating System Design and Implementation*, pages 137-150, 2004
[30] M. Zahariz, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. *Proc. of the USENIX Symposium on Operating System Design and Implementation*, pages 29-42, 2008.
[31] M. Sivathanu, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. Evolving RPC for Active Storage. In *Proc. of 12th international conference on Architectural support for programming languages and operating systems(ASPLOS)*, 2002.
[32] R. Wickremesinghe, J. S. Chase, J. S. Vitter. Distributed Computing with Load-Managed Active Storage. In *Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing(HPDC)*, 2002.
[33] E. J. Felix, K. Fox, K. Regimbal and J. Nieplocha. Active storage processing in a parallel file system. *In Proc. of the 6th LCI International Conference on Linux Clusters: The HPC Revolution*, April 2006.