Chinese Journal of Electronics
Vol.19, No.3, July 2010

Access Control Lists for Object-Based Storage Systems*

NIU Zhongying, ZHOU Ke, FENG Dan and YANG Tianming

(College of Computer Science and Technology, Huazhong University of Science and Technology; Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China)

Abstract — We developed an Access control list (ACL) mechanism for object-based storage systems. The proposed ACL mechanism affords Object-based storage devices (OSDs) much more flexibility in managing who can and how she accesses the object and makes it possible to implement completely distributed security for object-based storage systems. By enabling the ACL capability of inheritance and sharing, the mechanism reduces the number of ACLs needing to be stored and maintained. And by allowing the use of public key certificates as identifications of remote users, our ACL mechanism allows access control to extend beyond the local machine's realm to across-organizational users. Experimental results show that the overhead of access control is sufficiently small, with a bandwidth overhead of no more than 5%.

Key words — Object storage, Access control lists, Object-based file systems, Distributed security.

I. Introduction

Since object storage was first proposed in the NASD (Network-attached storage devices) project[1] at CMU, it has received more and more attention as one of the most promising technological solutions to next-generation storage systems in the past few years. Many systems have been designed using object-based storage technology. The two commercial products, Panasas[2] and Lustre[3], have been released in the commercial market. Meanwhile, the U.S. SNIA OSD working group and T10 Technical Committee of INCITS have made continued efforts for the standardization of object interface and the first version of the object-based storage interface standard (also referred to as T10 OSD standard)[4] was ratified by ANSI in January 2005. The latest draft of the proposed revision of OSD-3 revision (r00) is now available.

Object-based storage repartitions the file system functionalities and offloads the storage management functions to intelligent Object-based storage devices (OSDs). An object-based storage device manages its own storage space, provides persistent storage, and exports an object interface that is completely different from traditional file and block interfaces. With abundant on-board computing power, an OSD can independently provide storage services to clients without needing a file server. By distributing objects across many devices, object-based storage systems can provide high capability, throughput, reliability, availability and scalability. However, object-based storage creates a security problem not present in a centralized file system, because storage devices are now exposed directly to potential network attacks instead of being hidden behind a file server that can protect them. Most existing object storage systems, such as NASD[5], T10 OSD standard and Ceph[6], use capabilities to authenticate and authorize I/Os. A client wishing to access to storage devices must acquire necessary metadata information, such as locations of objects, and a capability authorizing the I/O from a Metadata server (MDS), and then presents the capability to the devices with the I/O request. The storage devices can thus use the capability to authorize access to object data. Fig.1 depicts a typical capability-based security architecture for object-based storage systems.

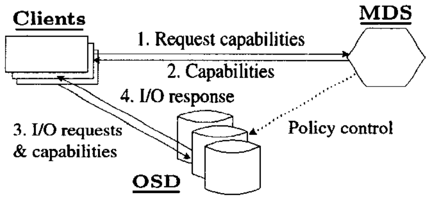

Fig. 1. Capability-based security architecture for object-based storage systemd. Clients request capabilities from MDS and use them to request I/O from OSDs

However, an object-based storage system may service millions of clients and holds tens or even hundreds of thousands of storage devices since object storage is designed for future or next-generation storage systems. Consequently, capability-based security schemes impose a lot of overhead on MDS, which has to be online and presents a central point of failure. Especially in HPC systems, it is impractical for servers to generate and return that many capabilities for an instant. Typically, MDS is replicated to improve the performance and prevent the central point of failure of the server. However, replicating the server is difficult and requires complex management task.

This paper describes the design and implementation of an alternative access control mechanism for object-based storage systems, which is based on Access control lists (ACLs). The ACL mechanism affords each Object-based storage device (OSD) much more flexibility in managing who can and how she accesses the object according to local ACLs along with each object stored in it, thus making it possible to implement completely distributed security for object-based storage systems. By enabling the ACL capability of in-

*Manuscript Received June 2009; Accepted Aug. 2009. This work is supported by the National Basic Research Program of China (973 Program) (N0.2004CB318201), National High-tech R&D Program (863 Program) (No.2009AA01A402), the Natioanl Natural Science Foundation of China (No.60703046, No.60873028), Changjiang Innovative Group of Education of China (No.IRT0725), the Program for New Century Excellent Talents in University (No.NCET-04-0693 and No.NCET-06-0650).

heritance and sharing, the mechanism reduces the number of ACLs needing to be stored and maintained. And by allowing the use of public key certificates as identifications of remote users, our ACL mechanism allows access control to extend beyond the local machine's realm to across-organizational users. We implemented the ACL mechanism in the HUST OSD project[7]. Experimental results show that the overhead of access control is sufficiently small, with a bandwidth overhead of no more than 5%.

II. Related Work

There have been numerous efforts to secure parallel and distributed storage systems[5,8−11], most of which are based on capabilities. Capability-based security maintains authorization information on a centralized authorization server, such as a security manager. The initial capability-based schemes are based on fine-grained capabilities, such as NASD[5], Azagury, *et al.*[12], Snapdragon[9], and the T10 OSD security protocol[4], which authorize access privileges at the granularity of a block or object, requiring the generation of tens of or even hundreds of millions of capabilities. This imposes a substantial overhead on the security manager, which has to be online and thus presents a central point of failure. If the security manager is down the entire system comes to a halt. LWFS[10] employs coarse-grained capabilities to grant access to a group of objects, but constrains the granularity of access control. Moreover, the LWFS capabilities can only be verified by the authorization server that generated them. As a result, the additional overhead can quickly overload the authorization server.

Leung *et al.*[6] propose an extended capability that authorizes I/Os for any number of clients to any number of files that span any number of devices by using public-key cryptography. Extended capabilities identify a group of users having the same access permissions to a set of files and the files themselves via the root hash of a Merkle hash tree[13] constructed from the user IDs and file identifiers, and store an access control matrix on MDS. However, the MDS's access control matrix with large-scale storage systems can become too large and complex to maintain and operate efficiently and economically, and the heavy MAC computation and signature can quickly exhaust MDS.

SCARED[11] extends NASD to provide mutual authentication between clients and devices. It supports authentication based on capabilities (as in NASD) as well as identity keys. In the case of identity keys, the SCARED device stores ACLs along with objects.

Kher and Kim[14] exploit RBAC in their system. In most of the cases, a client needs to acquire an identity key from the file manager, which can be used by the client to further derive role keys. Unlike SCARED, which stores identity-based access control lists along with each object, Kher and Kim store role-based access control lists with each object. It reduces the complexity of security administration in a large-scale system. However, both SCARED's and Kher and Kim's schemes do not address the design and implementation of access control lists in storage devices.

III. ACL Design

Our ACL mechanism allows fine-grained access control, giving the security administrator the flexibility to assign different access permissions to different users or groups. The defined ACLs are compatible with the current T10 OSD standard[4]. This section details the design of ACLs and discusses the concepts behind them.

1. ACL-based security architecture

Different from capability-based security architecture, which centrally maintains authorization information at a centralized MDS, an ACL-based security architecture for object-based storage systems, shown in Fig.2, stores ACLs at each OSD. Therefore, OSDs can authenticate and authorize the clients according to their certificates and local ACLs concurrently and in a distributed manner. Moreover, storing ACLs at OSDs makes it possible to implement completely distributed security for object-based storage systems. However, distributed security gives rise to the issue of ACL consistency, since the client's data in an object-based storage system is typically striped across multiple devices to achieve a high parallelism. Keeping ACLs consistent for all striped objects thus becomes a critical concern for the design of decentralized security mechanisms. In the following subsection, we discuss two potential approaches, namely, callback and centralized management, that we believe work well for maintaining ACL consistency in object-based storage systems.

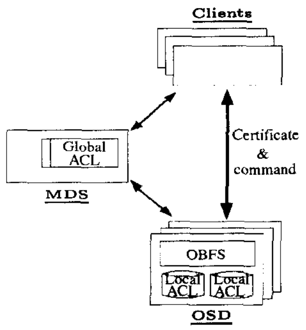

Fig. 2. ACL-based security architecture for object-based storage systems. OSDs authenticate and authorize the clients according to their certificates and local ACLs in a distributed manner

2. Keeping ACL consistent

The wide-area file system AFS[15] employs a callback mechanism to keep the cache consistent. It is similar to the invalidation mechanism used in the Web cache. The file server keeps track of all the clients that cache a particular document and, when the document is changed, the server sends out invalidation messages to all the clients. Cache consistency makes sure that a requested document is up-to-date while ACL consistency ensures that the ACLs associated to multiple stripes or replicas of an object make the same authorization decision. Though the cache consistency problem is different from the ACL consistency problem, the notion behind cache consistency solutions can be used to solve the ACL consistency problem. With callback, one can achieve strong consistency among multiple ACLs. For example, one can specify an OSD containing a striped object in a file as a master and other OSDs containing the data of this file as slaves. When an ACL in the master is modified, the master sends a notification to all slaves. Since the callback mechanism requires the storage device to maintain the map information of objects in a file. Thus an implementation of the callback approach may be similar to that in the Panasas parallel file system[16], which stores the file-level metadata, such as the file's storage map, in object attributes on two of the N objects used to store the file's data.

With callback, the system can be deployed to completely distributed security architecture. However, completely distributed access control gives rise to a complex privilege management. A simple review to a user's or file's permissions will require the administrator to contact multiple or even all storage devices. Since object-based storage systems typically maintain global metadata at a centralized MDS, one natural way to keep ACLs consistent is to store a global replica of all local ACLs at the MDS and then all ACL changes will first go to the MDS that will propagate it to other OSDs. Centrally storing a global replica of ACLs can achieve centralized security management, but it should be noted that this approach is completely different from the capability-based security system because the centralized authorization has been removed from the critical I/O path. In a practical system, global ACLs can be stored as metadata along with other common metadata. Since many researches have addressed the store and maintenance issues of metadata, this paper mainly concentrates on the design and implementation of local ACLs for Object-based file systems (OBFS) at OSDs.

3. ACL structure

The T10 OSD standard stores and maintains object attributes in the form of attribute page. To be compatible with the current standard[4], we propose an extended ACL attributes page that defines the fundamental ACL attributes and values. An object that allows to be accessed should have its ACL attributes page. Table 1 shows the proposed ACL attributes and associated parameters.

Table 1. The extended ACL attributes page

Attribute number	Length (bytes)	Attribute	Client settable	OSD logical unit provided
0h	40	Page identification	No	Yes
1h	1	List flag	Yes	No
2h to FFh		Reserved	No	
0100h to BFFF FFFFh	13	User ACE	Yes	No
C000 0000h to FFFF FF00h	13	Group ACE	Yes	No
FFFF FF01h to FFFF FFFEh		Reserved	No	

The page identification attribute identifies the organization that has defined the contents of the attributes page and the attributes page in which it appears. The format of the page identification has been specified by the standard. In an ACL attributes page, for example, the attribute value with attribute number 0h can be "HUST ACL".

Table 2. List flag values

Value	Name	Description
0h	ACCESS	ACL cannot be used for inheritance; it is only applied to this object.
01h	SUPER	ACL is only used for inheritance; it is not applied to this object.
02h to FFh		Reserved

The list flag attribute controls the inheritance of the ACL. Its attribute values are defined in Table 2. Access ACL defines the usual access permissions of protected objects and super ACL defines the permissions capable of inheritance by a child object from its parent object. ACL inheritance means that an object can automatically obtain the corresponding permissions granted to its parent without having to manually assign these permissions to it. It is also necessary for high-performance storage systems to enable the ACL capability of inheritance because in such systems a large number of objects can be generated in a timely manner and each object must be assigned the corresponding permissions.

An access control list includes a list of Access control entries (ACEs). Each user ACE attribute (0100h to BFFF FFFFh) defines access privileges of a user and each group ACE attribute (C0000000h to FFFF FF00h) defines access privileges of a group of users. The former grants only one user access to the object with that the access control entry is associated, while the latter authorizes a group of users.

4. ACL deployment

Each ACL attributes page is associated with an object. An access ACL attributes page can be associated with both objects and container objects that contain child objects, such as directories. A super ACL attributes page can be associated with only container objects. The super ACL attributes page for non-containers would be of no use, because no other system objects can be created inside non-containers. Our ACL mechanism can support multiple-level inheritance models through extending the current ACL definition in the future. For example, multiform ACLs with different inheritance properties can be defined through specifying different values for the list flag attribute. However, multiple-level inheritance makes determining object capabilities an even more complex operation that is prone to demanding potentially expensive traversals to determine what objects should be inherited. In this paper, we propose a simple but effective inheritance model by enabling multiple child objects to inherit a single parent object.

According to the T10 OSD standard, an OSD storage device contains one or more OSD logical units, each with four types of stored data objects: root object, partition, collection and user object. Each OSD logical unit has one and only one root object, as well as a plurality of partitions. Each partition contains a set of collections and user objects. The user object contains end-user data (*e.g.*, file or database data), while the collection is used for fast indexing of user objects. A user object may be a member of zero or more collections concurrently, while each collection can reference one or more user objects. Fig.3 shows the ACL deployment in an OSD logical unit. We propose an extended ACL collection shown in Table 3. Access ACLs are associated with user objects and super ACLs are associated with ACL collections. Thus the user objects referenced by an ACL collection can inherit the super ACL associated with that ACL collection. To avoid confusion, a user object can be a member of one and only one ACL collection.

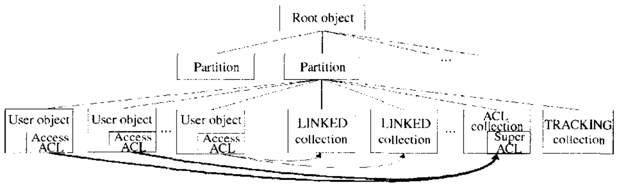

Fig. 3. The ACL deployment in an OSD logical unit

Table 3. Collection type codes

Code	Name	Description
00h	LINKED	T10 specific
01h	TRACKING	T10 specific
02h	ACL	User objects may be added to or removed from the collection using the Collections attributes page.
03h to EEh		Reserved
EFh	SPONTANEOUS	T10 specific
F0h to FFh		Reserved

IV. Reference Implementation

The proposed ACL-based security mechanism has been implemented in the HUST OSD project[7]. Our new OSD implementation supports not only the ACL-based security scheme but also the capability-based security scheme that has been defined in the T10 OSD standard.

ACL layout

Since attributes in an OSD are organized in pages, and the same number and kind of pages are usually associated with the same kind of objects, we store these relatively fixed attribute pages in the forepart of objects to get fast access to attributes. To meet the requirement of storing a variable-length attribute page, such as the ACL attributes page, we use extended attribute region to store an overrunning attribute.

As shown in Fig.4, the OBFS implementation separates an OSD logical unit into different regions for different functions. The super block records the characteristics of OBFS, including its size, the data block size, the empty and the filled data blocks and their respective counts, the size and location of the different regions, etc. The B+ tree region stores all B+ trees in OBFS. OBFS uses B+ trees to retrieve three types of data: (1) objects' onodes in data regions; (2) data blocks of large objects; (3) free blocks in data regions. The extended attribute region stores the overrunning part of a variable length attribute that cannot be completely filled in a fixed-length attribute page. The data region stores objects' onodes and data. All of the blocks in a region have the same size, but the block sizes in different regions may be different. Regions are initialized when there are insufficient free blocks in any initialized region to satisfy a write request. In this case, OBFS allocates a free region and initializes all of its blocks to the desired block size. In the current OBFS implementation, the block size in the B+ tree and data regions is set to 4kB, but the block size in the extended attribute region is set to 512 bytes. Thus a block in the extended attribute region can hold about 512/13=39 ACEs.

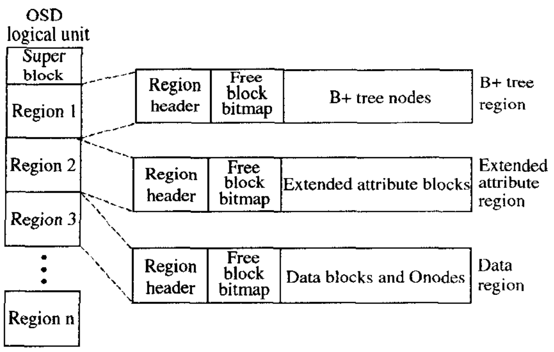

Fig. 4. OBFS structure

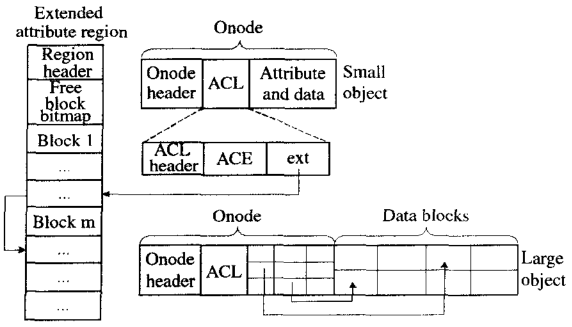

Fig. 5. Object structure and ACL layout

OBFS stores an ACL attributes page in an object' onode, which occupies a block in data regions. The onode, *i.e.*, object metadata, is used to track the status of the object, similar to the inodes in a Linux file system, such as ext2 and ext3 file systems. Fig.5 shows the object structure and ACL layout. The data and attributes of a small object are directly stored in the onode, where the ACL attribute comes after the onode header. The data and attributes (except ACLs) of a large object are stored in data blocks, which are referenced by the extended pointers in the object's onode. Each extended pointer is a two-tuple (address, length), which can specify any number of sequential blocks. The ACL in an onode is limited to 32 ACEs. Thus an ACL including the ACL header and extended pointer occupies $9 + 32 \times 13 + 8 = 433$ bytes in an onode, where the ACL header consists of the page number, the list flag and the number of ACEs in the ACL. The overrunning ACEs will be stored in the extended attribute region. A rational limit, such as 32 ACEs, enables an onode to hold all ACEs in most ACLs, each of which can thus be read into memory with the onode in a lump.

V. Performance Evaluation

We compare the security overhead of the ACL-based security implementation with that of the non-secure and capability-based implementations to evaluate how much performance degradation is incurred when the ACL-based and capability-based security schemes are added to an unsecured object-based storage system respectively.

1. Experimental setup

The experiments were conducted on 3 Linux hosts, each with one Intel Xeon 3.0 GHz processor and a total of 512MB physical memory. Each node was connected to a Highpoint Rocket 2240 Raid controller attached to 15 SATA disks (7200RPM, 300GB each). All machines ran Fedora Core 4 (kernel version 2.6.12) and were connected by a 1-Gbits Ethernet. In each experiment, one machine acted as MDS and the other two acted as OSD and the client respectively.

For the capability-security evaluation, the CMDRSP security method, one of the three security methods defined in the T10 OSD standard, was used in our experiments because this method provides the same security as the ACL-based security mechanism does. For the ACL-based security evaluation, we assumed the client and OSD had obtained an identity certificate from a Trusted authority (TA) that certifies the user identifier and user-to-group association. In addition, we also assumed that the client had authenticated herself to OSD and negotiated a shared key with the latter via an identity-based authenticated key agreement protocol[17]. The shared key is used to protect the OSD command from various network attacks, similar to the use of the capability key for securing an OSD command with the CMDRSP security method.

2. Latency breakdown

We ran a set of micro benchmarks on the ACL-based and capability-based security implementations and measured the latency of each operation in order to evaluate the various overheads associated with our OSD implementation. All the latency benchmarks were run on a collection of 512 files, each of size 4kB. In each benchmark, a fixed filesystem operation (*e.g.*, create, read, or write) was performed on each of the files in a randomized order. For each operation, the client and OSD drivers were instrumented to report the time spent in fine-grained sub-operations shown in Table 4.

Table 4. Operation breakdown

Sub-operation	Overhead
Command encapsulation (CmdEncap)	OSD command encapsulation
Communication to OSD (CommToOsd)	Latency that an OSD command is transmitted from clients to OSD; and Latency that results are returned from OSD to clients; and Latency that data is transmitted
Security	MAC computation (to compute the integrity check value of the command) and privilege verification for ACL-based security; or MAC computation, privilege verification and capability reconstruction for capability-based security
Disk access	

Table 5 summarizes the latency breakdown of the various operations for the ACL-based and capability-based security implementations, indicated by the ACL and CAP columns respectively, and the percentage of the latencies for the sub operations in relation to that for the OSD command. As observed in the table, the security overhead of the ACL-based security implementation, including MAC computation and privilege verification, is slightly lower than that of the capability-based security implementation. This is because the latter includes not only MAC computation and privilege verification, but also capability reconstruction that undoubtedly costs more than a simple privilege matching in the former.

3. Administration overhead

We measured the latency of the chmod operation to evaluate the overhead for security administration in ACL-based and capability-based security systems respectively. For the chmod operation, ACL-based security systems require two round trips from clients to MDS and then from the MDS to OSD respectively, while capability-based security systems require only one round trip from clients to MDS, because in the capability-based systems the chmod operation only needs to modify the privilege information in MDS, but in the ACL-based security systems the chmod operation needs to modify not only global ACLs in MDS but also local ACLs in OSD.

Table 6 shows the overhead for the chmod operation. The security overhead for the MDS command includes MDS command encryption and decryption, as well as privilege verification, which costs 86μs for all chmod operations. We store global ACLs along with other common metadata in an LDAP database. It costs 405μs to modify an LDAP database record and 180μs to write an ACE to the physical disk. As a result, a chmod operation costs a total of 756μs and 1467μs for capability-based and ACL-based security systems respectively. The latter is two times slower than the former. But it should be noted that permission operations is much less frequent than the demanding I/O operation, such as read and write.

Table 5. Latency breakdown of various operations

Latency(μs)	Create		Delete		Read		Write	
	ACL	CAP	ACL	CAP	ACL	CAP	ACL	CAP
CmdEncap	42(7%)	42(7%)	42(7%)	42(7%)	42(4%)	42(4%)	42(4%)	42(4%)
CommToOsd	258(43%)	258(40%)	258(44%)	258(41%)	571(56%)	571(53%)	329(35%)	329(33%)
Security	231(38%)	273(42%)	231(39%)	273(43%)	231(22%)	273(26%)	231(25%)	273(28%)
DiskAccess	73(12%)	73(11%)	62(10%)	62(10%)	184(18%)	184(17%)	340(36%)	340(35%)
TotolCost	604	646	593	635	1028	1070	942	984

Table 6. Overhead of chmod operations

Latency(μs)	MDS Command				OSD Command				TotalCost
	CmdEncap	CommToMds	Security	DBAccess	CmdEncap	CommToOsd	Security	DiskAccess	
Capability	38	227	86	405	–	–	–	–	756
ACL-based	38	227	86	405	42	258	231	180	1467

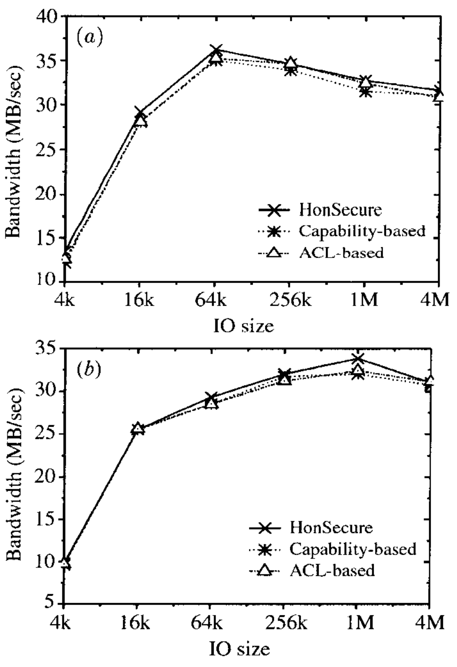

Fig. 6. Read/write bandwidth with 1 OSD for the non-secure, ACL-based and capability-based security systems. (*a*) Read bandwidth; (*b*) Write bandwidth

4. Throughput

In this experiment, we sequentially read and wrote a 512MB file with multiple transfer sizes (the transferred bytes per OSD command) and measured the raw read, write performance of the non-secure, ACL-based and capability-based security setups. Fig.6 shows the bandwidth as a function of transfer size for the write and read benchmarks. For both non-secure and secure setups, the throughput increases with the transfer size. This is because the overall overhead of OSD command building and validation is lesser for higher transfer sizes when compared to lower transfer sizes. Another important observation from the table is that both ACL-based and capability-based security setups have comparable bandwidth with the non-secure setup. Compared to the non-secure setup, the read and write performance with both the ACL-based and capability-based security setups decrease by less than 5%.

5. Discussion

The above experimental results in terms of latency and throughput show that the ACL-based and capability-based security implementations have comparable performance on OSD, though the former have actually incurred slightly lower latency. However, the performance benefits of the ACL-based security mechanism stem from the significantly reduced round trips from clients to MDS, because a client wishing to access OSD has to acquire a capability from MDS in a capability-based system, which may overhead the MDS. On the contrary, the ACL-based security mechanism completely removes the bottleneck caused by security overhead by avoiding capabilities for the frequent read and write operations.

VI. Conclusions

This paper described the design and implementation of an access control list mechanism for object-based storage systems. The proposed ACL mechanism allows fine-grained access control, giving the security administrator the flexibility to assign different access permissions to different users or groups. The defined ACLs are compatible with the current T10 OSD standard and allow to be inherited automatically, and thus are easy to implement and maintain. Moreover, the ACL mechanism affords OSDs much more flexibility in managing who can and how she accesses the object and makes it possible to implement completely distributed security for object-based storage systems, thus removing MDS' performance bottleneck caused by security overhead through avoiding capability requests in traditional capability-based systems. And by allowing the use of public key certificates as identifications of remote users, our ACL mechanism allows access control to extend beyond the local machine's realm to across-organizational users. Experimental results show that the overhead of access control is sufficiently small, with a bandwidth overhead of no more than 5%.

References

[1] G.A. Gibson, D.F. Nagle, K. Amiri *et al.*, "A cost-effective, high-bandwidth storage architecture", *Proc. of 8th ASPLOS*, San Jose, CA, USA, pp.92–103, Oct. 1998.

[2] David Nagle, Denis Serenyi, Abbie Matthews, "The panasas activescale storage cluster-delivering scalable high bandwidth storage", *Proceedings of the ACM/IEEE SC2004 Conference*, Pittsburgh, PA, USA, 2004.

[3] P.J. Braam, The Lustre storage architecture. Cluster File Systems, Inc., *http://www.lustre.org/documentation.html*, Aug. 2004.

[4] T10 Technical Committee, NCITS. SCSI object-based storage device commands -2 (OSD-2), project t10/1729-d, revision 5 edition, Jan. 2009.

[5] H. Gobioff, "Security for a high performance commodity storage subsystem", *Ph.D. Thesis*, Carnegie Mellon University, USA, July 1999.

[6] A.W. Leung, E.L. Miller, S Jones, "Scalable security for petascale parallel file systems", *Proc. of SC07. Reno*, NV, USA, Nov. 2007.

[7] Zhongying Niu, Ke Zhou, Dan Feng *et al.*, "Implementing and evaluating security controls for an object-based storage system", *Proc. of MSST'07*, San Diego, CA, USA, pp.87–99, Sept. 2007.

[8] Ke Zhou, Yongfeng Huang, Jiangling Zhang *et al.*, "Disk tree — a case of parallel storage architecture to improve performance in random access pattern", *Chinese Journal of Electronics*, Vol.14, No.1, pp.39–44, 2005.

[9] M.K. Aguilera, M. Ji, M. Lillibridge *et al.*, "Block-level security for network-attached disks", *Proc. of FAST'03*, San Francisco, CA, pp.159–174, 2003.

[10] R.A. Oldfield, A.B. Maccabe, S. Arunagiri *et al.*, "Lightweight I/O for scientific applications", *Technical Report 2006-3057*, Sandia National Lab, May 2006.

[11] B.C. Reed, E.G. Chron, R.C. Burns *et al.*, "Authenticating network-attached storage", *IEEE MICRO*, Vol.20, No.1, pp.49–57, 2000.

[12] A. Azagury, R. Canetti, M. Factor *et al.*, "A two layered approach for securing an object store network", *Proc of IEEE Security in Storage Workshop*, Greenbelt, MD, pp.10–23, 2002.

[13] R.C. Merkle, "A digital signature based on a conventional encryption function", *Advances in Cryptology - Crypto'87*, pp.369–378, 1987.

[14] Vishal Kher, Yongdae Kim, "Decentralized authentication mechanisms for object-based storage devices", *Proc. of the Second IEEE International Security in Storage Workshop*, Washington, DC, pp.1–10, 2003.

[15] J.H. Howard, M.L. Kazar, S.G. Menees *et al.*, "Scale and performance in a distributed file system", *ACM Transactions on Computer Systems*, Vol.6, No.1, pp.51–81, 1988.

[16] Brent Welch, Marc Unangst, Zainul Abbasi *et al.*, "Scalable performance of the Panasas parallel file system", *Proceedings of 6th USENIX Conference on File and Storage Technologies*, San Jose, CA, pp.17–33, 2008.

[17] Quan Yuan, Songping Li, "A new efficient ID-Based authenticated key. agreement protocol", *Cryptography ePrint Archive, Report 2005/309*, Mar. 2005.

NIU Zhongying was born in 1978 in Nanyang, Henan, China. He received the B.S. degree from College of Computer Science and Technology, Huazhong University of Science and Technology (HUST) in 2003. Currently he is a Ph.D. candidate in College of Computer Science and Technology at HUST. His current interests include network storage, parallel file system and storage security. (Email: niuzhy@gmail.com)

ZHOU Ke was born in 1974 in Xiangtan, Hunan, China. He received the Ph.D. degree from College of Computer Science and Technology, Huazhong University of Science and Technology (HUST) in 2003. Currently he is a professor of College of Computer Science and Technology at HUST. His main research interests include computer architecture, network storage system, parallel I/O, storage security and network data behavior theory. (Email: k.zhou@hust.edu.cn)

FENG Dan was born in 1970 in Jingshan, Hubei, China. She received the Ph.D. degree from College of Computer Science and Technology, Huazhong University of Science and Technology (HUST) in 1997. Currently she is a professor of College of Computer Science and Technology at HUST. Her current interests include computer architecture, redundant array independent disks, mass storage, and information storage technology. (Email: dfeng@hust.edu.cn)

YANG Tianming was born in 1968 in Chongyang, Hubei, China. He received the B.S. degree from Department of Computer Science, Zhengzhou Institute of Technology in 1991. Currently he is a Ph.D. candidate in College of Computer Science and Technology at Huazhong University of Science and Technology. His current interests include data backup, networking storage, parallel file system and storage security. (Email: ytmzqyy@yahoo.cn)