Object-Based Storage

Mike Mesnier, Carnegie Mellon and Intel
Gregory R. Ganger, Carnegie Mellon
Erik Riedel, Seagate Research

ABSTRACT

Storage technology has enjoyed considerable growth since the first disk drive was introduced nearly 50 years ago, in part facilitated by the slow and steady evolution of storage interfaces (SCSI and ATA/IDE). The stability of these interfaces has allowed continual advances in both storage devices and applications, without frequent changes to the standards. However, the interface ultimately determines the functionality supported by the devices, and current interfaces are holding system designers back. Storage technology has progressed to the point that a change in the device interface is needed. Object-based storage is an emerging standard designed to address this problem. In this article we describe object-based storage, stressing how it improves data sharing, security, and device intelligence. We also discuss some industry applications of object-based storage and academic research using objects as a foundation for building even more intelligent storage systems.

INTRODUCTION

Industry has begun to place pressure on the interface to storage, demanding that it do more. Since the first disk drive in 1956,[1] disks have grown by over six orders of magnitude in density and over four orders in performance, yet the storage interface (i.e., blocks) has remained largely unchanged. Although the stability of the block-based interfaces of SCSI and ATA/IDE has benefited systems, it is now becoming a limiting factor for many storage architectures. As storage infrastructures increase in both size and complexity, the functions system designers want to perform are fundamentally limited by the block interface.

Recent industry and academic research suggests a shift in storage technology, in which devices evolve from relatively unintelligent and externally managed to intelligent, self-managed, and aware of the storage applications they serve. However, creating such an intelligent device requires a more expressive interface. Many in the industry believe that an interface based on storage *objects* can be the answer.

A storage object is a logical collection of bytes on a storage device, with well-known methods for access, attributes describing characteristics of the data, and security policies that prevent unauthorized access. Unlike blocks, objects are of variable size and can be used to store entire data structures, such as files, database tables, medical images, or multimedia.

Objects can be regarded as the convergence of two technologies: *files* and *blocks*. Files provide user applications with a higher-level storage abstraction that enables secure data sharing across different operating system platforms, but often at the cost of limited performance due to file server contention. Blocks offer fast, scalable access to shared data; but without a file server to authorize the I/O and maintain the metadata, this direct access comes at the cost of limited security and data sharing.

Objects can provide the advantages of both files and blocks. Like blocks, objects are a primitive unit of storage that can be directly accessed on a storage device (i.e., without going through a server); this direct access offers performance advantages similar to blocks. Like files, objects are accessed using an interface that abstracts storage applications from the metadata necessary to store the object, thus making the object easily accessible across different platforms. Providing direct, file-like access to storage devices is therefore the key contribution of object-based storage.

The remainder of this article is organized as follows. We discuss today's prominent storage architectures, the trade-offs involved, and the fundamental limitations of block-based interfaces. We describe object-based storage as an architecture that will remove these limitations. We conclude with a discussion of the industry activity around object-based storage, in particular the standards efforts in the Storage Networking Industry Association (SNIA) and the flurry of activity around object-based file systems.

STORAGE TODAY AND TRADE-OFFS

An ideal storage architecture would provide strong security, data sharing across platforms (i.e., operating systems), high performance, and

[1] *IBM introduced the Random Access Method for Accounting and Control (RAMAC) in 1956, with a density 2000 b/in^2 and throughput of 8 kbytes/s.*

0163-6804/03/$17.00 © 2003 IEEE

scalability in terms of the number of devices and clients. Today's architectures force system designers to decide which of these features is most important, as choosing an architecture involves a trade-off. The three storage architectures in common use today are direct-attached storage (DAS), storage area networks (SANs), and network-attached storage (NAS). A fourth architecture, often called a SAN file system, has recently been introduced in an attempt to capture the features of both NAS and SANs.

DAS connects block-based storage devices directly to the I/O bus of a host machine (e.g., via SCSI or ATA/IDE). While DAS offers high performance and minimal security concerns, there are limits on connectivity. SCSI, for example, is limited by the width of the bus (a 16-bit bus can have at most 16 hosts or devices). To address the connectivity limits of DAS , and consequently enable the consolidation and sharing of storage devices, the SAN was introduced. A SAN is a switched fabric that provides a fast, scalable interconnect for large numbers of hosts and storage devices. With this added connectivity, however, came the need for better security. SANs therefore introduced concepts such as zoning (like a virtual private network) and host-device authentication to keep the fabric secure.

DAS and SAN are both block-based. The storage application (e.g., file system) is responsible for mapping its data structures (files and directories) to blocks on the storage devices. The extra data required to do this mapping is commonly referred to as *metadata*. For multiple hosts to share data blocks, they must also share metadata, and do so in a manner that guarantees metadata consistency among the hosts. The complexity of this process has resulted in block sharing only among tightly coupled performance-sensitive storage applications such as clustered file systems and databases. Most other infrastructures only allow hosts to share data indirectly through files by using NAS.

NAS is just another name for file serving, which was introduced to enable data sharing across platforms. With NAS, the metadata describing how files are stored on devices is managed completely on the file server. This level of indirection enables cross-platform data sharing but comes at the cost of directing all I/O through the single file server. NAS may be implemented on top of a SAN or with DAS, the former often referred to as a *NAS head*. In either case, clients will be limited by the performance of the file server and will rarely see the aggregate performance of the storage devices (Fig. 1).

To address the performance limitations of NAS, SAN file systems have recently appeared. In a SAN file system, the file server and clients are all connected to a SAN on which the file system is stored. Given this connectivity, the file server can share file metadata with the clients, thus allowing the clients to directly access the storage devices. Examples include EMC's HighRoad, IBM's StorageTank, and Veritas' SANPoint Direct. Because the devices have no mechanism for authorizing I/O, increasing file serving performance in this manner reduces security; the SAN mechanisms for device security only protect the entire device, not data within the device. A SAN file system is illustrated in Fig. 2.

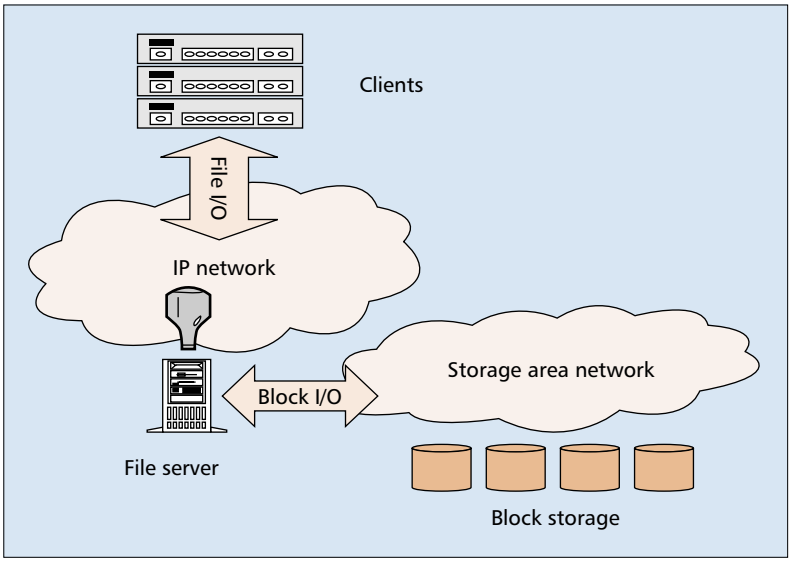

Figure 1. *This figure illustrates NAS being used to share files among a number of clients. The files themselves may be stored on a fast SAN. However, because the clients often suffer from queuing delays at the server, they rarely see the full performance of the SAN.*

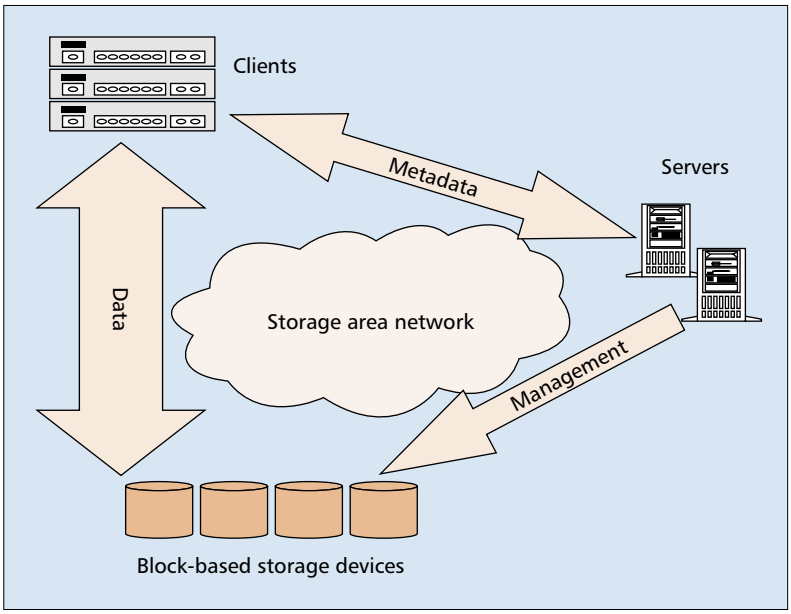

Figure 2. *This figure illustrates a SAN file system being used to share files among a number of clients. The files themselves are stored on a fast storage area network (e.g., iSCSI) to which the clients are also attached. File server queuing delays are avoided by having the file server share metadata with the clients who can directly access the storage devices; but, because the devices cannot authorize I/O, the file server must assume that the clients are trusted.*

The trade-off in today's architectures is therefore security and cross-platform data sharing (files) vs. high performance (blocks). While files allow one to securely share data between systems, the overhead imposed by a file server can limit performance. Yet, increasing file serving performance by allowing direct client access comes at the cost of security. Building a scalable, high-performance, cross-platform, secure data sharing architecture requires a new interface

Unlike block I/O, creating objects on a storage device is accomplished through a rich interface similar to a file system. And, because objects can grow and shrink dynamically, the storage device is responsible for all internal space management of the object.

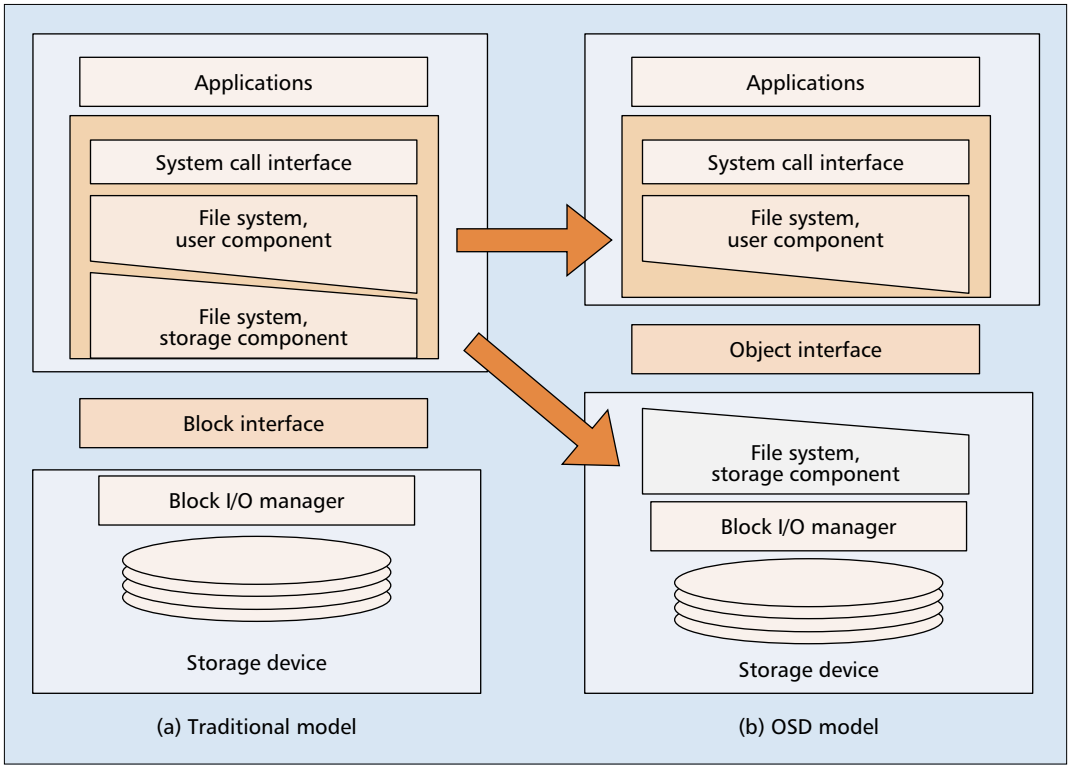

Figure 3. *Offloading of storage management from the file system.*

that provides both the direct access nature of SANs and the data sharing and security capabilities of NAS.

Object-Based Storage

Overview

Objects are storage containers with a file-like interface, effectively representing a convergence of the NAS and SAN architectures. Objects capture the benefits of both NAS (high-level abstraction that enables cross-platform data sharing as well as policy-based security) and SAN (direct access and scalability of a switched fabric of devices). Although objects do behave like files in terms of space allocation and data access, they are only intended to serve as containers for storage applications (e.g., file systems and databases), which implement any desired additional interfaces (e.g., locking) and lookup mechanisms (e.g., directories).

An object is variable-length and can be used to store any type of data, such as files, database records, medical images, or multimedia. A single object could even be used to store an entire file system or database. The storage application decides what gets stored in an object. Unlike block I/O, creating objects on a storage device is accomplished through a rich interface similar to a file system. And, because objects can grow and shrink dynamically, the storage device is responsible for all internal space management of the object (i.e., the storage device maintains the allocation and free-space metadata structures, such as UNIX index nodes, or inodes, and free-block bitmaps).

Objects are composed of data, user-accessible attributes, and device-managed metadata. The data stored in an object is opaque to the object-based storage device and is simply stored in the data portion of the object. The user-accessible attributes describe characteristics of the object, some of which will be opaque and others not. For example, a quality of service (QoS) attribute may describe latency and throughput requirements for a multimedia object. Lastly, the device-managed metadata is any extra information (e.g., an inode) maintained by the storage device for the purposes of managing the physical storage of the object.

We refer to a device that stores objects as an *object-based storage device* (OSD). OSDs can come in many forms, ranging from a single disk drive to a storage controller with an array of drives. OSDs are not limited to random access or even writeable devices; tape drives and optical media could also be used to store objects. The difference between an OSD and a block-based device is the interface, not the physical media.

The most immediate effect of object-based storage is the offloading of space management (i.e., allocation and tracking of used and free blocks) from storage applications. To illustrate this offloading, consider the traditional file system architecture (Fig. 3a). Block-based file systems can roughly be divided into two sections: a user component and a storage component. The user component is responsible for presenting user applications with logical data structures, such as files and directories, and an interface for accessing these data structures; and the storage component maps the data structures to the physical storage. This separation of responsibilities makes it easy to offload management to the storage device, which is the intended effect of object-based storage. In Fig. 3b, the user component of

the file system is unchanged, the storage management component offloaded (and therefore the metadata) to the storage device, and the device interface changed from blocks to objects.

The management of block metadata (the storage component in Fig. 3a) is completely determined by the storage application (e.g., file systems have unique ways of laying out data and maintaining on-disk metadata structures). These dependencies make directly sharing data blocks between hosts difficult, as a priori knowledge of both the metadata structures and on-disk layout is necessary before accessing the storage device. Furthermore, sharing the devices requires special coordination among the hosts in order to distribute the tasks associated with space allocation (e.g., by sharing a free-block bitmap). In offloading metadata to the storage device, objects remove the dependency between the metadata and storage application, making data sharing between different storage applications feasible. Even more, cluster scalability improves considerably when the hosts no longer need to coordinate metadata updates.

Storage applications may still maintain their own indexing information (e.g., directory metadata) to resolve an object id from a higher-level name. But, given this id, the object can then be accessed in a platform-independent manner. This makes sharing data considerably easier. For example, a backup application could be handed a list of object ids, allowing for a more efficient physical backup of the device.

Furthermore, with all metadata offloaded, storage applications can now store their structures as single objects as opposed to collections of blocks. And, because the device can treat objects individually, it is easy to set security policies on a per-object basis, similar to the manner in which files can be protected by a file server. Objects allow storage applications to set flexible security policies that will result in authorization for an entire device, a collection of objects on the device, a single object, or even bytes within an object.

The immediate benefits of object-based storage are therefore cross-platform data sharing and application-level security. These benefits are most relevant for SAN-based storage devices and would be of limited value for DAS-based storage, which is already under the protection and management of a single host. An additional benefit results from the devices managing the physical layout of the data, as new opportunities arise within the storage device for self-management. Self-management benefits both DAS and SAN devices equally, and includes actions such as reorganizing data to improve performance, scheduling regular backups, and recovering from failures. For example, file-level prefetching within a block-based device is precluded by the fact that the device does not know about files. In contrast, an object-based device could easily prefetch files (stored as objects) on behalf of storage applications, or organize files according to the order in which they are mostly commonly accessed.

Operating systems must support objects if they are to be widely adopted. Fortunately, the clean separation between the user and storage OS components (Fig. 3) will facilitate the change. In particular, object-based file systems will be relatively straightforward to implement (a file system need only give up control over block management), and an OS's I/O subsystem can be introduced to objects by virtue of a new class driver, similar to those that already exist for disk and tape. Reference code from Intel Labs shows how both can be done in Linux [1].

The remainder of this section describes the object-based storage architecture in greater depth, particularly the interface to an OSD, the attributes associated with an object, the security architecture used for object-based storage, and some opportunities objects present for more intelligent storage devices.

Data Sharing

The improved data sharing of objects is a result of both the higher-level interface and the attributes describing the data being stored.

Interface — The interface to object-based storage is very similar to that of a file system. Objects can be created or deleted, read or written, and even queried for certain attributes — just like files are today. File interfaces have proven easy to understand, straightforward to standardize (e.g., CIFS, NFS), and therefore possible to share between different platforms.

The interface can also be easily extended with application-specific methods for manipulating data within an object, a technique referred to as active disks [2]. For example, a database filter could be associated with an object, the output of which could be returned on subsequent read operations. Furthermore, an OSD could allow storage applications to establish sessions with the device to encapsulate application-specific parameters such as QoS or security guarantees. In short, objects introduce a mechanism in the storage device that allows the device treat storage applications, and even clients, individually.

Attributes — Attributes improve data sharing by allowing storage applications to share a common set of information describing the data (e.g., access times). They are also the key to giving storage devices an awareness of how objects are being accessed.

In most deployments, attributes will at least contain information analogous to that contained in an index node (inode), the primary data structure used in many UNIX file systems. An inode contains file attributes such as access time, size, and group and user information, all of which can be efficiently stored with the object data and, in certain cases, interpreted by the storage device. For example, a write operation that updates the size attribute would be reflected on subsequent attribute requests, making the update visible to other storage applications accessing the object. Clustered applications could therefore depend on the storage device to maintain this metadata, rather than delegate this responsibility to an in-band (i.e., on the data path) metadata server that may hinder performance.

Beyond these file-like attributes, additional information can be made available such as likely patterns of access to the object (e.g., sequentially

The interface to object-based storage is very similar to that of a file system. Objects can be created or deleted, read or written, and even queried for certain attributes — just like files are today.

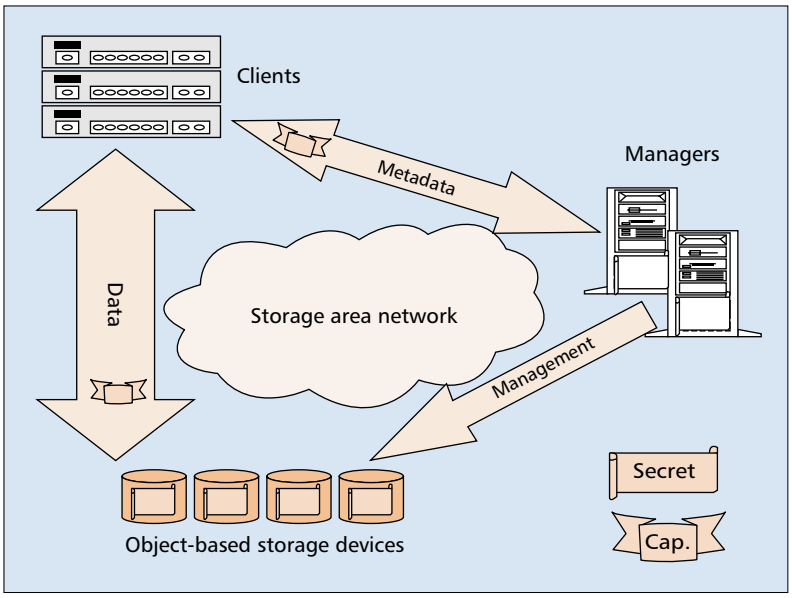

Figure 4. *The object-based storage security architecture. Object* managers *grant capabilities to clients; clients present these capabilities to the devices on every I/O. Secrets shared between the manager and the storage devices are used to generate a* keyed hash *of the capability, thereby protecting it from modification.*

or randomly accessed), or even relationships to other objects. For example, multimedia files on a storage device may have similar attributes that will cause them to be organized and efficiently managed as a group.

SECURITY

Security is perhaps the single most important feature of object-based storage that distinguishes it from block-based storage. Although security does exist at the device and fabric level for SANs (e.g., devices may require a secure login and switches may implement zoning), objects make it possible to partition large devices or fabrics into multiple security domains whose access policies are individually determined by the storage applications. In the object-based security architecture [3], every access is authorized, and the authorization is done without communicating with a central authority that may slow the data path (Fig. 4).

The storage application in Fig. 4 could be a file server that stores each file as an individual object on a storage device. This architecture is identical to the SAN file system shown in Fig. 2, with one important distinction: the clients no longer have to be trusted, nor does the network. This means that the clients and storage devices can be anywhere on the network, without the risk of unauthorized clients accessing the storage, or authorized clients accessing the storage in an unauthorized manner.

The fundamental building block in an object-based security system is a cryptographically strong *capability* that contains a tamper-proof description of the rights of a client. This capability represents the security policy, and is created out-of-band (i.e., off the main data path) by an *object manager* that manages the storage devices and grants access to clients. While in possession of this capability, the client can access the storage device, and it is the job of the storage device to validate the integrity of the capability to ensure that neither it nor the request has been modified. The OSD therefore provides only the mechanism for enforcing security, rather than the policy, which is set by the storage application. Separating policy from mechanism is key to building a scalable security architecture. In particular, not having to maintain client-specific authentication information on the device means the storage device will scale independently from the number and types of clients in the system.

While an OSD does not question the authenticity of the client, it does need some mechanism to validate the integrity of the capability, or proof that the object manager granted access to the client. Providing this guarantee requires that the object manager and device share a secret that can be used in creating a secure hash of the contents of the capability. Before granting a client its capability, the manager will first create a *keyed hash* of the capability, using the secret as the key. It will then return both the secure hash, referred to as a *capability key*, amf the capability to the client. The client is expected to use this capability key in creating its own keyed hash of every request sent to the OSD. This hash protects the command from undetected modification, similar to how the hash of the capability protects the capability from modification.

The request sent to an OSD includes the command, the client capability, and a signature (or digest) of the entire request. Upon receipt of a new request, the OSD must first validate the client digest. The OSD will create its own digest of the request and compare this with the digest sent by the client.[2] If they match, the OSD is guaranteed that neither the capability nor any of the arguments in the request were modified. Had either of these changed, the digest generated by the OSD would differ from that sent by the client, and the OSD would reject the request; all responses sent from the OSD to the client can be protected using a digest similar to that sent by the client.

In some environments, object-based storage must also ensure the privacy of data transfers and guard against delay and replay attacks. In the absence of a trusted channel (e.g., IPSec), the object-based storage security architecture allows the capability key to be used as an encryption key, thereby safeguarding the client and storage devices from snooping attacks. Delay and replay attacks are prevented by adding client timestamps and sequence numbers to each I/O, respectively. The timestamp will establish a small window in which the command is valid. If the command is received by the storage device outside of this window, it will not be authorized. Similarly, the storage device can check the sequence number of each command and reject those that have already been executed.

To avoid a trip to the object manager on every I/O, clients may cache and reuse capabilities. The object manager revokes cached capabilities by embedding expiration times in the capability or establishing a new secret with the storage device.

Although the object manager has been taken off the data path, it is still a single point of fail-

[2] *It is not immediately obvious how this is possible when the storage device does not possess the capability key used to generate the digest. However, recall that the capability key is just a hash of the client's capability (which was sent in the request), and the secret shared between the object manager and the OSD. Thus, the OSD can simply regenerate the capability key and use this key to generate its digest of the request.*

ure or attack. If the object manager is compromised, the system is compromised. Clustering can be used to improve availability, but at the expense of more attack points. These issues are not endemic to object-based storage, and are identical to what traditional file servers struggle with today. The goal of object-based storage is not to improve the availability of file servers but rather to improve the performance and scalability of the entire system by taking the servers off the main data path and allowing secure direct access to the storage devices.

Intelligence

With the emergence of object-based storage comes the potential for storage devices to actively *learn* important characteristics of the environments in which they operate. Storage devices today are largely unaware of the users and storage applications actually using the storage, because block-based storage devices manage opaque data blocks. With objects, storage devices can understand some of the relationships between the blocks on the device, and can use this information to better organize the data and anticipate needs.

Object attributes can contain static information about the object (e.g., creation time), dynamic information that is updated on each access (e.g., last access time), information specific to a storage application and uninterpreted by the device (e.g., filename, group, or user ids), and information specific to a current user (e.g., QoS agreement). Attributes may also contain hints about the object's behavior such as the expected read/write ratio, the most likely patterns of access (e.g., sequential or random), or the expected lifetime of the object. Having access to such attributes enables a storage system to better organize and serve the data.

Using objects to better manage storage is an active area of academic research. One of the largest questions to be addressed relates to the attributes themselves, specifically which attributes are most useful in classifying the behavior of objects. Past research has already shown that file attributes play an integral role in determining file behavior [4, 5]. For example, the name of a file can be used to predict how the file may be accessed. Parallels exist for object-based storage.

In general, objects enable attribute-based learning environments in which storage devices can become aware of the environments in which they operate, and thereby better allocate and provision resources. Furthermore, with increased knowledge of storage and user applications, storage devices can perform application-specific functions, thereby making the SAN a computational resource. Indeed, storage devices are themselves computers, with processors, network connections, and memory. Through more expressive interfaces, these resources can be more effectively exploited.

Related Work

Primitive forms of object-based storage can be found in the early work (circa 1980) on object-oriented operating systems, including the Hydra OS from Carnegie Mellon [6] and the iMAX-432 OS from Intel [7]. These operating systems used variable-size objects on disk to store not just files, but all pageable entities within an OS, including process state, instructions, and data. Although these operations systems never took off, they were instrumental in establishing the fundamentals of capability-based security.

The SWALLOW project (circa 1980) from Massachusetts Institute of Technology [8] was among the first systems to implement a distributed object store, and was a precursor to early file serving architectures.

The seminal work on object-based storage occurred at Carnegie Mellon University's Parallel Data Lab (PDL) with the Network-Attached Secure Disks (NASD) project [9]. The architecture focused on cost-effectively adding processing power to individual disks, specifically for networking, security, and basic space management functionality. This research led to a larger industry-sponsored project under the auspices of the National Storage Industry Consortium (NSIC). Several storage companies joined the collaboration, and NASD was generalized to network-attached *storage devices*, where individual drives, array controllers, and appliances could take advantage of the interface change. This work yielded a standard extension to the SCSI protocol for object-based storage [10].

The NSIC draft continues to be defined in the Object-Based Storage Devices working group of the Storage Networking Industry Association (SNIA) [11]. The SNIA plans to submit a completed SCSI draft standard to T10 in 2003, and is also exploring mappings to transports other than SCSI, including direct mappings onto IP.

Even as standards develop, the industry is already implementing systems using object-based storage technology. IBM is exploring object-based storage for their next generation of StorageTank [12]; the National Laboratories and Hewlett-Packard are building the highly scalable Lustre file system [13], with object-based storage as their basic storage interface; and smaller companies and startups (BlueArc, Data Direct, and Panasas) are building devices that make use of object-based storage. The Venti project at Bell Laboratories and Centera from EMC have also used object-based storage concepts to implement write-once media for the archival of reference data. Both systems employ the concept of content addressable storage (CAS) in which the id of an object (Venti actually uses a variable length *block*) is a unique hash of the data con-tents.

Intelligent storage is a hot area of academic research. Carnegie Mellon's PDL continues to explore the use of more expressive interfaces between host operating systems and storage devices [2, 14]. As one such interface, objects enable information exchange between the device and the OS in order to achieve better functionality and performance in the storage system. Researchers at the University of Wisconsin, Madison are exploring an alternative path, semantically smart disk systems that attempt to learn file system structures behind existing block-based interfaces [15]. Many other research

With objects, storage devices can understand some of the relationships between the blocks on the device, and can use this information to better organize the data and anticipate needs.

Although block-based interfaces have enabled significant advances both in storage devices and storage applications, we are now at a point where continued progress requires a change in the device interface.

groups are beginning to explore storage-level intelligence as well.

SUMMARY

Although block-based interfaces have enabled significant advances in both storage devices and storage applications, we are now at a point where continued progress requires a change in the device interface.

The object interface offers storage that is secure and easy to share across platforms, but also high-performance, thereby eliminating the common trade-off between files and blocks. Furthermore, objects provide the storage device with an awareness of the storage application and enable more intelligence in the device.

Object-based storage was designed to exploit the increasing capabilities of storage devices. Characteristics of the future storage device may include self-configuration, self-protection, self-optimization, self-healing, and self-management. Replacing block interfaces with objects is a major step in this evolution.

ACKNOWLEDGMENTS

The authors would like to thank the IEEE reviewers for their comments and suggestions, and Chenxi Wang (CMU) for her initial review. We would also like to thank the members of the SNIA OSD technical work group for their regular meetings and conference calls in which many of the arguments presented in this article were formed.

REFERENCES

[1] Intel, Internet SCSI (iSCSI) Reference Implementation, http://www.intel.com/labs/storage

[2] E. Riedel, G. Gibson, and C. Faloutsos, "Active Storage for Large-Scale Data Mining and Multimedia Applications," *Int'l. Conf. Very Large DBs*, New York, NY, Aug. 24–27, 1998, pp. 62–73.

[3] H. Gobioff, "Security for a High Performance Commodity Storage Subsystem." Ph.D. thesis, TR CMU-CS-99-160. Carnegie-Mellon Univ., July 1999.

[4] D. Ellard *et al.*, "Passive NFS Tracing of An Email and Research Workload," *Conf. File and Storage Tech.*, San Francisco, CA, Mar. 31–Apr. 2, 2003, pp. 203–17.

[5] D. Roselli, J. R. Lorch, and T. E. Anderson, "A Comparison of File System Workloads," *USENIX Annual Tech. Conf.*, San Diego, CA, June 18–23, 2000, pp. 41–54.

[6] G. Almes and G. Robertson, "An Extensible File System for HY-DRA," *3rd Int'l. Conf. Software Eng.*, Atlanta, GA, May 1978.

[7] F. J. Pollack, K. C. Kahn, and R. M. Wilkinson, "The iMAX-432 Object Filing System," ACM Symp. OS Principles, Asilomar, CA, published in *OS Rev.*, vol. 15, no. 5, Dec. 1981, pp. 137–47.

[8] D. P. Reed and L. Svobodova, "SWALLOW: A Distributed Data Storage System for a Local Network," *Int'l. Wksp. Local Networks*, Zurich, Switzerland, Aug. 1980.

[9] G. A. Gibson *et al.*, "A Cost-effective, High-bandwidth Storage Architecture," Architectural Support for Prog. Languages and OS, San Jose, CA, 3–7 Oct. 1998, published in *SIGPLAN Notices*, vol. 33, no. 11, Nov. 1998, pp. 92–103.

[10] R. Weber, "Object-Based Storage Devices (OSD)," http://www.t10.org

[11] SNIA, Object-Based Storage Devices (OSD) workgroup, http://www.snia.org/osd

[12] IBM, Storage Tank, http://www.almaden.ibm.com

[13] P. Braam, The Lustre Project, http://projects.clusterfs.com/lustre

[14] G. R. Ganger, "Blurring the Line Between OSs and Storage Devices," Tech. rep. CMU-CS-01-166, Carnegie Mellon Univ., Dec. 2001.

[15] M. Sivathanu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Evolving RPC for Active Storage," Architectural Support for Prog. Languages and OS, San Jose, CA, 05–09 Oct. 2002, published in *OS Rev.*, vol. 36, no. 5, 2002, pp. 264–76.

BIOGRAPHIES

MIKE MESNIER (mmesnier@ece.cmu.edu) is a storage architect with Intel Labs, a Ph.D. researcher at Carnegie Mellon University, and co-chair of the Object-Based Storage Devices workgroup of SNIA. He received his Master's in computer science from the University for Illinois, Urbana-Champaign, and has been with Intel since 1997. His current activities include file systems and storage technologies, in particular intelligent storage devices. Prior to joining Intel, he was a research scientist on the NEOS project at Argonne National Laboratory.

GREG GANGER [M] (greg.ganger@cmu.edu) is director of the CMU Parallel Data Lab (PDL), academia's premier storage systems research center, and an associate professor in the ECE department at Carnegie Mellon University. He has broad research interests in computer systems, including storage systems, operating systems, security, networking, and distributed systems. He received his Ph.D. in computer science and engineering from the University of Michigan. He is a member of the ACM.

ERIK RIEDEL (erik.riedel@seagate.com) leads the Interfaces and Architecture Department at Seagate Research, Pittsburgh, Pennsylvania. His group focuses on novel storage systems with increased intelligence for optimized performance, automated management, and content-specific optimizations. Before joining Seagate Research, he was a researcher at Hewlett-Packard Laboratories. He received his Ph.D. in computer engineering from Carnegie Mellon University for his work on Active Disks, an extension to NASD. His interests include I/O in a number of areas, including parallel applications, data mining, databases, file systems, and scientific data processing.