A Unified Object Oriented Storage Architecture

Andy Hospodor, Ethan Miller, Rekha Pitchumani,Yangwook Kang, Darrell Long
Department of Computer Science, University of California, Santa Cruz, 95064, US

Ahmed Amer
Department of Computer Engineering, Santa Clara University, Santa Clara, CA 95035, US

Yulai Xie
Huazhong University of Science and Technology, China

Abstract

Object Oriented (OO) Storage has a history of false starts and high expectations. OO disk drives, in particular, have been described by industry standards, reference implementations and countless publications for over a decade. Today, computer science lacks a unified storage architecture. By reducing devices to storage objects that contain data and associated methods, such an architecture is not only possible, but may extend to include all varieties of storage devices, interfaces, operating systems and file systems.

1 Introduction

This paper introduces a unified storage architecture based upon object oriented storage devices (OOSD) that obfuscate the physical implementation issues of the underlying hardware. Within such an OO Storage Architecture the metadata is decoupled from the data to permit the management of objects rather than blocks. Storage objects are managed through associated methods over a wide range of storage devices, including Magnetic Disk, Optical Disk, Storage Class Memories, Tape, Flash, and anticipates new storage devices such as Shingled Disk.

2 History and background

In 1986, Quantum introduced the Q200, the first disk drive with an integrated SCSI (Small Computer Systems Interface) controller that managed its address space as a contiguous set of logical blocks [Quantum1986][SCSI1986]. Previously, access to disk storage occurred using Cylinder, Head and Sector addresses, or Cylinder, Head and Count/Key/Data addresses in the case of IBM Enterprise Storage. The obfuscation of physical geometry by devices following the Q200 led the industry down a path of virtualization where nearly all storage devices appear as their 1986 equivalents. While the consolidation of file system and device drivers offered distinct advantages in management, it made the addition of new capabilities difficult and extended the adoption cycle. New commands were first argued over by standards committees, a process that took years, then implemented in devices from select manufactures, and, hopefully, supported by a variety of operating system and storage management vendors.

The SCSI *Third Party Copy* command, for example, allows a tape drive to oversee the transfer of data between two devices, typically a disk drive and a tape drive. However, operating systems generally have different file system/device driver stacks for disk and tape, so, Third Party Copy required an entirely new, integrated stack. Sadly, this important feature was rarely implemented in storage architectures. Another, more timely example is the Trim command used to perform garbage collection in storage based upon flash memory. Different operating systems (OS) and even different versions of an OS may or may not support Trim. Apple supports Trim on Apple supplied SSDs in version 10.7 of their operating system (aka Lion), but no support is offered for Trim on non-Apple SSDs nor is Trim supported at all in version 10.6.6 (aka Snow Leopard). Reinstallation of the OS during a recovery/rebuild operation may also disable Trim and prevent garbage collection. [Trim2010]

[Reidel2001] and [ANSI2009] provided extensions of SCSI that include Object Oriented command sets. While this approach was groundbreaking, and generated a swell of research, it ultimately appeared as another attempt to extend the antiquated SCSI interface from 1986 to the present. Again, the perceived requirement that new storage devices use existing file systems and devices drivers had a similar effect: *no Object Oriented devices ever made it into mass production.*

3 Motivation

The authors believe that an OO Storage Architecture is best implemented by scrapping the existing infrastructure of file systems and devices drivers. While this may be considered blasphemy within the data storage industry, not doing it has the effect of locking us into storage architectures of the 1980s.

Modern storage devices have microprocessor based control systems that decode commands, position read/write elements, translate logical to physical addresses, transfer data and report status. However, today's disk drives are block level devices. Flash Translation Layers make flash memory look like a SCSI disk, while Hardware Abstraction Layers make USB memory sticks, look like SCSI disks and RAID subsystems make groups of SATA disks also appear as SCSI disks .

Modern microprocessors are capable of managing a storage address space as a collection of objects, rather than blocks. Such an OO storage device would map its objects and access them using specified methods, as shown in Table 1. This approach offers design flexibility and permits efficient allocation and placement policies based up the device's intimate knowledge of its own physical geometry. Legacy support could be as simple as using a Logical Block Address(LBAs) in place of an object name, so, storage.read(12345) would read LBA 12345 in the same manner a SCSI or ATA storage device would read it today.

Table 1: Basic Methods of an Object Oriented Storage Architecture

Method	***Description***
new storage = StorageObject(size)	Instantiate new Storage Objects
new moreStorage = StorageObject(size)	
metadata = storage.finger	Obtain metadata from a Storage Object
buffer = storage.read	Retreive data from Storage Object into memory
storage.write = buffer	Store data from memory onto Storage Object
storage.append = buffer	Extend Storage Object with additional data
storage.find(regExp)	Search a Storage Object
sortedStorage = storage.sort(keyValue)	Sort a Storage Object by key value pair
moreStorage = storage.replicate	Replicate a Storage Object

New storage devices are now free to implement new methods, rather than emulate old ones. An OO storage device would *publish* its capabilities and allow the operating system, file system and ultimately applications, to *subscribe* to their methods. Many modern, object oriented operating systems already contain a registry for precisely this purpose.

4 Effects on File Systems and Drivers

Within our Unified OO Storage Architecture, the OO storage devices manage their objects and the File System manages the name space. Traditional hierarchical File Systems can simply assign globally unique names to objects and allow users and applications to reference files in the usual manner. Non-hierachical File Systems can tap into the rich metadata and search devices for relevant objects.

Although device drivers would still be necessary to access each specific interface, the reader should be aware that the most popular serial interfaces: FibreChannel, Gigabit Ethernet, Serial SCSI, Serial ATA and IEEE 1394 all use similar signaling technologies. The electrons go on the wire in the same fashion, yet the main differences are the connectors and the protocols. Thus, each serial interface requires a different driver. In the future, common or generic device drivers will extend across families of similar interfaces such as wireless, serial, parallel, long haul. Once this happens, translation layers and abstraction layers become unnecessary. Operating systems, file systems and applications will no longer perform virtualization - this function will be absorbed within the Object Oriented Storage Architecture.

5 Conclusion

Object Oriented Storage Architectures have the promise of integrating new storage technologies and new features without the continuos modification of operating systems, file systems and device drivers. As the industry embraces Flash, Phase Change and other other non-volative memories, the ability to integrate and bring products to market quickly offers a distinct competitive advantage - one that Active Disk never had.

6 Acknowledgment

The authors received funding for this research from Center for Research in Intelligent Storage (CRIS).

7 References

[ANSI2009] ”The Object-Based Storage Device Commands - 2 (OSD-2) Rev 05a,” ANSI INCITS 458-2011, 16 January 2009.

[Intel2010] ”Intel High Performance Solid State Drive - Advantages of TRIM”. Intel Corporation, Santa Clara, CA,14 Sept 2010.

[Quantum1986] ”Q200 Series Intelligent Disk Drives”, Quantum Corporation, Milpitas, CA, Nov 1986.,

[Reidel2001] Riedel, E., Faloutsos, C., Gibson, G.A. and Nagle, D.F. ”Active Disks for Large-Scale Data Processing”. IEEE Computer, June 2001.

[SCSI1986] ”Small Computer System Interface (SCSI)”, ANSI X3.131-1986.