panasas®

OBJECT STORAGE ARCHITECTURE

WHITE PAPER | APRIL 2007

TABLE OF CONTENTS

ABSTRACT

This white paper describes the issues confronting Linux compute clusters when using today's storage architectures. These issues suggests that a new storage architecture based on the Object-based Storage Device (OSD) can provide the file sharing capability needed for scientific and technical applications while delivering the performance and scalability needed to make the Linux cluster architecture effective. This white paper also reviews the components of a storage system based on objects and the data flow through the system in typical storage transactions. Next it summarizes the advantages of the Object-based Storage Architecture in the areas of performance, scalability, manageability and security. Finally it concludes with a survey of the history of the project and the current efforts to create a standard around the Object-based Storage Architecture..

BACKUP OVERVIEW

The Object-based Architecture is based on data Objects, which encapsulate user data (a file) and attributes of that data. The combination of data and attributes allows an Object-based storage system to make decisions on data layout or quality of service on a per-file basis, improving flexibility and manageability. The device that stores, retrieves and interprets these objects is an Object-based Storage Device (OSD). The unique design of the OSD differs substantially from standard storage devices such as Fibre Channel (FC) or Integrated Drive Electronics (IDE), with their traditional block-based interface. This is accomplished by moving low-level storage functions into the storage device and accessing the device through a standard object interface. Object-based Storage Device enables:

- Intelligent space management in the storage layer
- Data-aware pre-fetching, and caching

Ultimately, OSD-based storage systems can be created with the following characteristics:

- Robust, shared access by many clients
- Scalable performance via an offloaded data path
- Strong fine-grained end-to-end security

These capabilities are highly desirable across a wide range of typical IT storage applications. They are particularly valuable for scientific, technical and database applications that are increasingly hosted on Linux cluster compute systems which generate high levels of concurrent I/O demand for secure, shared files. The Object-based Storage Architecture is uniquely suited to meet the demands of these applications and the workloads generated by large Linux clusters.

Each organization's backup requirements are different; the type of backup scheme should be chosen to suit the needs of the organization. All of these backup schemes are a compromise between minimizing the backup window and minimizing the time to restore. However the backup is performed, it will have an impact not only on the storage, but also on the systems using that storage. Backing up data generated by active applications is not without risks. Although the backup and restore software may be able to read the data that the application is working on, that data may not be in a consistent state. This makes that backup data useless for restore purposes because it will put the application in an inconsistent state. A number of backup/restore software vendor address these issues by providing agents that cooperate with the applications that run on the customer's server to make sure that the backups that are being made are consistent and can be used for restores.

The Linux Cluster Story

The high-performance computing (HPC) sector has often driven the development of new computing architectures, and has given impetus to the development of the Object Storage Architecture. Some history can provide an understanding of the importance of the Linux cluster systems, which are revolutionizing scientific, technical, and commercial computing. HPC architectures took a fundamental turn with the invention of Beowulf clustering by NASA scientists at the Goddard Space Flight Center in 1994. Development of the Message Passing Interface (MPI), Beowulf allowed racks of commodity Intel PC-based systems to emulate the functionality of monolithic Symmetric Multi-Processing (SMP) systems. Since this can be done at less than 1/10th the cost of the highly specialized, shared memory systems, the cost of scientific research dropped dramatically. Beowulf clusters, now more commonly referred to as Linux clusters, are the dominant computing architecture for technical computing, and are quickly gaining traction in commercial industries as well.

Unfortunately, storage architectures have not kept pace, causing systems administrators to perform arduous data movement and staging tasks to get stored data into the Linux clusters. There are two main problems that the storage systems for clusters must solve. First, they must provide shared access to the data so that the applications are easier to write and the storage is easier to balance with the compute requirements. Second, the storage system must provide high levels of performance, in both I/O rates and data throughput, to meet the aggregated requirements of 100's and in some cases up to 1000's of servers in the Linux cluster.

Linux cluster administrators have attempted several approaches to meet the need for shared files and high performance. Common approaches involve supporting multiple NFS servers or copying data to the local disks in the cluster. But to date there has not been a solution that effectively stages and balances the data so that the power of the Linux compute cluster can be brought to bear on the large data sets typically found in scientific and technical computing.

Aggregate Throughput and I/O

Due to the number of nodes in the cluster, the size of the data sets, and the concurrency of their access patterns, Linux clusters demand high performance from their storage system. As Linux clusters have matured, the scale of the clusters has increased from 10's of compute nodes to 1000's of nodes. This creates a high aggregate I/O demand on the storage subsystem even if the demand of any single node is relatively modest. Applications, such as bioinformatics similarity searching, create demands on the system for very high random I/O access patterns while the compute nodes search through hundreds of thousands of small files. Alternatively, high-energy physics modeling typically uses datasets containing files that are gigabytes in size, creating demand on the storage system for very high data throughput. In either case, application codes running on many nodes across the Linux cluster creates a demand for highly concurrent access in both random I/O and high data throughput. This level of performance is rarely seen in typical enterprise infrastructures and cause huge burdens on the storage systems.

Shared Files

Unlike the monolithic supercomputers that preceded Linux clusters, the data used in the compute process must be available to a large number of the nodes across the cluster simultaneously. Shared access to storage lowers the complexity for the programmer by making data uniformly accessible rather than forcing the programmer to write the compute job for the specific node that has direct access to the relevant portion of the dataset. Similarly, shared data eliminates the need for the system administrator to load the data to specific locations for access to the compute nodes or to balance the storage traffic in the infrastructure.

In many applications the project is not a single analysis of a single dataset, but rather a series of analyses combining multiple datasets, where the results of one process provide inputs to the next. For example, geologic information in the oil and gas industry typically takes the raw seismic traces from time and depth migration analysis, and combines them with well-head information to create 4-Dimensional (4D) visualizations. Given the size of the data and results sets, simply moving the information between local and/or network storage systems could add days to the completion of the project and increase the likelihood of error and data loss.

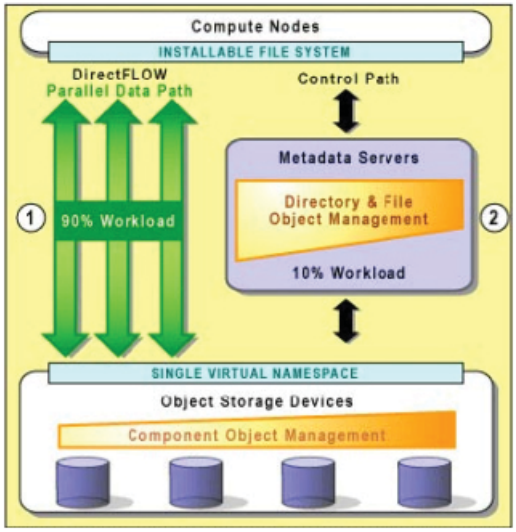
Sharing files across the Linux cluster substantially decreases the burden on the scientist writing the programs and the system administrator trying to optimize the performance of the system. However, providing shared access to files requires that there be a central repository for the file locations, known as the storage system metadata server, to track where each block of every file is

stored on disk and which node of the cluster is allowed to access that file. If the metadata server also sits on the data path between the cluster nodes and the disk arrays, as nearly all file servers today are designed, it becomes a major bottleneck for scaling in both capacity and performance.

CURRENT STORAGE ARCHITECTURES

There are two types of network storage systems, each distinguished by their command sets. First is the SCSI block I/O command set, used by Storage Area Networks (SANs), which provides high random I/O and data throughput performance via direct access to the data at the level of the disk drive or fibre channel. Second, the Network Attached Storage (NAS) systems that use NFS or CIFS command sets for accessing data with the benefit that multiple nodes can access the data as the metadata on the media is shared. Linux clusters require both excellent performance and data sharing from their storage systems. In order to get the benefits of both high performance and data sharing, a new storage design is required that provides both the performance benefits of direct access to disk and the ease of administration provided by shared files and metadata. That new storage system design is the Object-based Storage Architecture.

Architectural Breakthrough



The Object Storage Architecture combines the two key advantages of today's storage systems, performance and file sharing. When combined, these advantages eliminate the drawbacks that have made their previous solutions unsuitable for Linux cluster deployments.
First, the Object Storage Architecture provides a method for allowing compute nodes to access storage devices directly and in parallel providing very high performance. Second, it distributes the system metadata allowing shared file access without a central bottleneck. The Object Storage Architecture offers a complete storage solution for Linux clusters without the compromises that today's storage systems require in either performance or manageability.

Parallel Data Access

The Object Storage Architecture defines a new, more intelligent disk interface called the Object-based Storage Device (OSD). The OSD is a network-attached device containing the storage media, disk or tape, and sufficient intelligence to manage the data that is locally stored. The compute nodes communicate directly to the OSD to store and retrieve data. Since the OSD has intelligence built in there is no need for a file server to intermediate the transaction. Further, if the file system stripes the data across a number of OSDs, the aggregate I/O rates and data throughput rates scale linearly. For example, a single OSD attached to Gigabit Ethernet may be capable of delivering 400 Mbps of data to the network and 1000 storage I/O operations, but if the data is striped across 10 OSDs and accessed in parallel, the aggregate data rates achieve 4,000 Mbps and 10,000 I/O operations. These peak rates are important, but for most Linux cluster applications, the aggregate sustained I/O and throughput rates from storage to large numbers of compute nodes are even more important. The level of performance offered by the Object Storage Architecture is not achievable by any other storage architecture.

Distributed Metadata

Current storage architectures are designed with a single monolithic metadata server that serves two primary functions. First, it provides the compute node with a logical view of the stored data (the Virtual File System or VFS layer), the list of file names, and typically the directory structure in which they are organized. Second, it organizes the data layout in the physical storage media (the inode layer).

The Object Storage Architecture divides the logical view of the stored data (VFS layer) from the physical view (the inode layer) and distributes the workload allowing the performance potential of the OSD to avoid the metadata server bottlenecks found in today's NAS systems. The VFS portion of the metadata typically represents approximately 10% of the workload of a typical NFS server, while the remaining 90% of the work is done at the inode layer with the physical distribution of data into storage media blocks.

In the Object Storage Architecture, the inode work is distributed to each intelligent OSD. Each OSD manages the layout and retrieval of the data that is presented to it. It maintains the metadata that associates the objects (files or portions of files) with the actual blocks on the storage media. Thus 90% of the metadata management is distributed among the intelligent storage devices that actually store the data. If a file is striped across ten OSDs, no single device has to do more than 10% of the work that a conventional metadata server must perform in a today's NAS systems or file servers. This provides an order of magnitude improvement in the performance potential for the system's metadata management. In addition, because the metadata management is distributed, adding more OSDs to the system increases the metadata performance potential in parallel with the increased capacity of the system.

OBJECT STORAGE COMPONENTS

There are five major components to the Object Storage Architecture.

- Object - Contains the data and enough additional information to allow the data to be autonomous and self-managing.
- Object-based Storage Device (OSD) - An intelligent evolution of today's disk drive that can store and serve objects rather than simply putting data on tracks and sectors.
- Installable File System (IFS) - Integrates with compute nodes, accepts POSIX file system commands and data from the Operating System, address the OSDs directly and stripes the objects across multiple OSDs.
- Metadata Server - Intermediates throughout multiple compute nodes in the environment, allowing them to share data while maintaining cache consistency on all nodes.
- Network Fabric - Ties the compute nodes to the OSDs and Metadata Servers.

Objects

The Object is the fundamental unit of data storage in this system. Unlike files or blocks, which are used as the basic components in conventional storage systems, an object is a combination of file data plus a set of attributes that define various aspects of the data. These attributes can define on a per file basis the RAID levels, data layouts, and quality of service. Unlike conventional block storage where the storage system must track all of the attributes for each block in the system, the object maintains its own attributes to communicate with the storage system how to manage each particular piece of data. This simplifies the task of the storage system and increases its flexibility

by distributing the management of the data with the data itself.

Within the storage device, all objects are accessed via a 96-bit object ID. The object is accessed with a simple interface based on the object ID, the beginning of the range of bytes inside the object and the length of the byte range that is of interest (<object, offset, length>). There are three different types of objects. The "Root" object on the storage device identifies the storage device and various attributes of the device itself, including it's total size and available capacity. A "Group" object provides a "directory" to logical subset of the objects on the storage device. A "User" object carries the actual application data to be stored.

The user object is a container for data and two types of attributes.

- Application Data - The application data is essentially the equivalent to the data that a file would normally have in a conventional system. It is accessed with file-like commands such as Open, Close, Read, and Write.
- Storage Attributes – These attributes are used by the storage device to manage the block allocation for the data. This includes the object ID, block pointers, logical length, and capacity used. This is similar to the inode-level attributes inside a traditional file system. There is also a capability version number used when enforcing access control to objects.
- User Attributes – These attributes are opaque to the storage device and are used by applications and metadata managers to store higher-level information about the object. These attributes can include file system attributes like ownership and access control lists (ACLs), which are not directly interpreted by the storage device as described later. Attributes can describe Quality of Service requirements that apply specifically to a given object. These attributes can tell the storage system how to treat an object, for instance what type of RAID to apply, the size of the capacity quota or the performance characteristics required for that data.

Object-based Storage Device

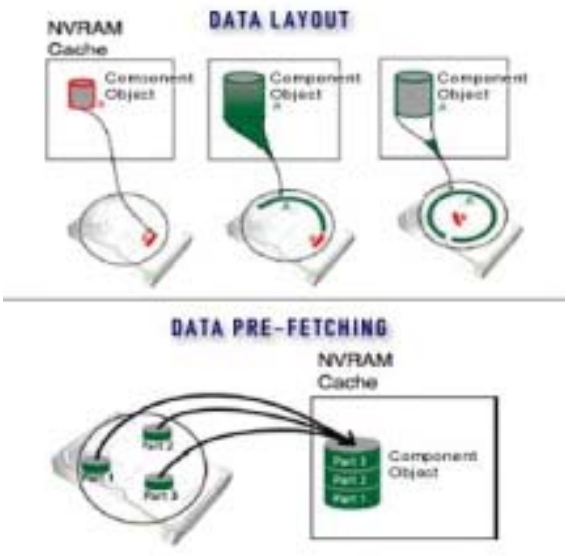
The Object-based Storage Device represents the next generation of disk drives for network storage. The OSD is an intelligent device that contains the disk, a processor, RAM memory and a network interface that allows it to manage the local object store, and autonomously serve and store data from the network. It is the foundation of the Object Storage Architecture, providing the equivalent of the SAN fabric in conventional storage systems. In the "Object SAN," the network interface is gigabit Ethernet instead of fibre channel and the protocol is iSCSI, the encapsulation of the SCSI protocol transported over TCP/IP. SCSI supports several command sets, including block I/O, tape drive control, and printer control. The new OSD command set describes the operations available on Object-based Storage Devices. The result is a group of intelligent disks (OSDs) attached to a switched network fabric (iSCSI over Ethernet) providing storage that is directly accessible by the compute nodes. Unlike conventional SAN configurations, the Object Storage Devices can be directly addressed in parallel, without an intervening RAID controller, allowing extremely high aggregate data throughput rates.

The OSD provides four major functions for the data storage architecture:

- Data Storage – The primary function in any storage device is to reliably store and retrieve data from physical media. Like any conventional storage device; it must manage the data as it is laid out into standard tracks and sectors. The data is not accessible outside the OSD in block format, only via their object IDs. The compute node requests

a particular object ID, an offset to start reading or writing data within that object and the length of the data block requested.

- Intelligent Layout – The OSD uses its memory and processor to optimize the data layout on the disk and pre-fetching of data from the disk. The object and its protocol provide additional information about the data that is used to help make layout decisions. For example, the object metadata provides the length of data to be written, allowing a contiguous set of tracks to be selected. Using a write-behind cache, a large amount of the write data can then be cached and written in a small number of efficient passes across the disk platter. Similarly the OSD can do intelligent read-ahead or pre-fetching, of the blocks for an object and have them available in buffers for maximum access performance.

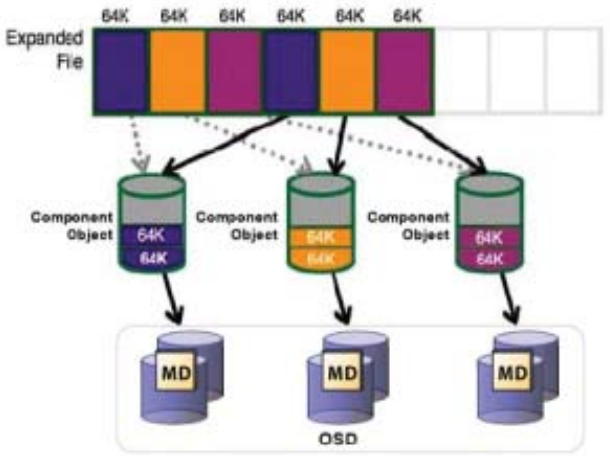


- Metadata Management – The OSD manages the metadata associated with the objects it stores. This metadata is similar to conventional inode data including the blocks associated with an object and the length of the object. In a traditional system, this data is managed by the file server (for NAS) or by the host operating system (for direct-attached or SAN). The Object Storage Architecture distributes the work of managing the majority of the metadata in the storage system to the OSDs and lowers the overhead on the host compute nodes. The OSD also reduces the metadata management burden on the Metadata Server by maintaining one component object per OSD, regardless of how much data that component object contains. Unlike traditional systems where the Metadata Server must track each block in every stripe, on every drive; successive object stripe units are simply added to the initial component object. The component objects grow in size, but for each object in the system, the Metadata Server continues to track only one component object per OSD reducing the burden on the Metadata server, and increasing its scalability.
- Security – There are two ways the object protocol improves security over SAN or NAS network storage systems. First, object storage is a network protocol and like any network transaction (SAN or NAS) it is potentially vulnerable to an external attack. In addition, it allows distributed access to the storage array from the host nodes (similar to a SAN), which can allow a node to intentionally or unintentionally (via an operating system failure) attempt to write bad data or into bad locations. Implementing the OSD architecture takes security to a new level by eliminating the need to trust clients external to the system. Each command or data transmission must be accompanied by a capability that authorizes both the sender and the action. The capability is a secure, cryptographic token provided to the compute node. The token describes to the OSD, which object that the compute node is allowed to access, with what privileges, and for what length of time. The OSD inspects each incoming transmission for the proper authorization capabilities and rejects any that are missing, invalid or expired.

Distributed File System

In order for the compute nodes to read and write objects directly to the OSDs, an installable file system must be deployed. The distributed file system provides four key functions in the Object Storage Architecture.

- POSIX File System Interface – The distributed file system must provide a transparent interface to the applications above it. The distributed file system will provide a POSIX interface to the application layer which allows the application to perform standard file system operations such as Open, Close, Read and Write files to the underlying storage system. In addition, it must support a full set of permissions and access controls expected by Linux applications, allowing it to have exclusive or shared access to any given file.
- Caching – The distributed file system must provide caching in the compute node for incoming data complementing the cache in the OSD. There will also be a cache for write data that aggregates multiple writes for efficient transmission and data layout at the OSDs. A third cache must be maintained for metadata and security tokens, so that the client can quickly generate secure commands to access data on the OSDs for which they have been given permission.
- Striping/RAID – The distributed file system must handle the striping of objects across multiple OSDs on a per object basis. Unlike standard RAID arrays, an object distributed file system can apply a different data layout and RAID level to each object. The distributed file system takes an object and breaks it down to component objects, which are the subset of an object sent to each OSD. The size of each component object (stripe unit size) is specified as an attribute of the object. The stripe width, or the number of OSDs that the object is striped across, is also specified as an attribute of the object. Because the object is read or written in parallel, the width of the stripe will correlate directly to the bandwidth of the object. If RAID is specified, the parity unit will be calculated by the client and applied to the object stripe.



- iSCSI – The distributed file system must implement an iSCSI driver which encapsulates the SCSI command set, the Object extensions to the command set and the data payload across a TCP network in order to transmit and receive data from the OSDs. TCP/IP accelerators (called TCP Offload Engines or TOE) can provide the iSCSI and TCP protocol processing which offloads TCP and iSCSI processing from the compute node to the TOE adapter.
- Mount – All clients mount the file system at the root, using access controls to determine access to different portions of the file tree. Authentication mechanisms are required such as Kerberos, Windows NTLM, and Active Directory. The identity of the compute nodes must be maintained via these authentication mechanisms, UID/GIDs for Unix and SIDs for Windows systems.
- Additional file system interfaces – Beyond a POSIX interface, other application interfaces such as Message Passing Interface for I/O (MPI-IO) may be useful. This interface allows parallel application writers to more efficiently control the layout of the data across

the OSDs via low-level ioctls in the file system. This can be useful for creating very wide stripes for massive bandwidth or to allow cluster checkpointing to a single file for maximum restart flexibility. Two MPI-IO implementations are widely used, MPICH/ROMIO from Argonne Labs and MPI Pro from MSTI, which will also support the MPI-2 standard.

Metadata Server

The Metadata Server (MDS) controls the interaction of the compute nodes with the objects on the OSDs by coordinating access to nodes that are properly authorized. The MDS is also responsible for maintaining cache consistency for users of the same file. In NAS systems, the metadata server (filer head) is an integral part of the data path, causing significant bottlenecks as traffic increases. The Object Storage approach removes the Metadata Server from the data path allowing high throughput and more linear scalability that are typically associated with SAN topologies that allow clients to interact directly with the storage devices. The Metadata Server provides the following services for the Storage Cluster:

- Authentication – The first role of the MDS is to identify and authenticate Object-based Storage Devices wishing to join the storage system. The MDS provides credentials to new storage system members and checks/renews those credentials periodically to assure that they are valid members. Similarly when a compute node wants access to the storage system, the MDS assures its identity and provides authorization. In the case of a compute node, it must turn the work of authentication over to an external service, which provides this service for the organization at large.
- File and Directory Access Management – The MDS provides the compute node with the file structure of the storage system. When the node requests to perform an operation on a particular file, the MDS examines the permissions and access controls associated with the file and provides a map and a capability to the requesting node. The map consists of the list of OSDs and their IP addresses, containing the components of the object in question. The capability is a secure, cryptographic token provided to the compute node, which is examined by the OSD with each transaction. The token describes to the OSD, which object that the compute node is allowed to access, with what privileges, and for what length of time.
- Cache Coherency – In order to achieve maximum performance, compute nodes will normally request the relevant object, and then work out of locally cached data. If there are multiple nodes using the same file, steps must be taken to assure that the local caches are updated if the file is changed by any of the nodes. The MDS provides this service with distributed object locking or callbacks. When a compute node asks for Read or Write privileges to a file or a portion of a file from the MDS, a callback is registered with the MDS. If the file privileges allow multiple writers, and is modified by another node, the MDS generates a callback to all of the nodes that have the file open, which invalidates their local cache. Thus, if the node has Read access to a file that has been updated, it must go back to the OSDs to refresh its locally cached copy of that data, thereby assuring that all nodes are operating with identical data.
- Capacity Management – The MDS must also track the balance of capacity and utilization of the OSDs across the system to make sure that the overall system makes optimum use of the available disk resources. When a compute node wants to create an object, the MDS must decide how to optimize the placement of the new file as it authorizes the node to write the new data. Since the node does not know how large the file will be at the time of file creation, the MDS provides the node with an escrow, or quota, of space. This allows the node to maximize the performance of the write operation by creating

and writing data in one step. Any excess quota is recovered once the file is closed maintaining maximum performance during the critical write operations.

- Scaling - Metadata management is the key architectural issue for storage systems attempting to scale in capacity and performance. Because the Object Storage Architecture separates file/directory management from block/sector management, it can scale to levels greater than any other storage architecture. It distributes the block/sector management to the OSDs (which is approximately 90% of the workload) and maintains the file/directory metadata management (10% of the workload) in a separate server that can also be implemented as a scalable cluster. The scalability of the MDS is the key to allowing the entire object storage system to scale, balanced in both capacity and performance.

Network Fabric

The network is a key element of the Object Storage Architecture. It provides the connectivity infrastructure that binds the Object-based Storage Devices, Metadata Server and compute nodes in a single fabric. With the advent of inexpensive gigabit Ethernet, it became possible to run storage traffic at speeds that meet or exceed specialized storage transports like fibre channel. This gives the Object Storage Architecture two advantages, the lower component costs implied by the commodity status of Ethernet, and more importantly, the lower management costs associated widespread knowledge of building reliable Ethernet fabrics. However, the Object Storage Architecture is wedded only to TCP/IP, rather than Ethernet. It is possible to build an Object Storage system on other transports such as Myrinet and InfiniBand using their support for TCP/IP. The Object Storage Architecture has three key network components:

- iSCSI – As described above, the iSCSI protocol is used as the basic transport for commands and data to the OSDs, encapsulating SCSI inside of a TCP/IP packet. The SCSI Command Data Blocks (CDB) deliver the commands to the storage device to read and write data, as well as the data payload itself. The iSCSI protocol has been extended to support the object command set, yet remain within the iSCSI protocol definitions.
- RPC Command Transport – The Object Storage Architecture has a separate protocol for communication between compute nodes and Metadata Servers. This is a lightweight Remote Procedure Call (RPC) that facilitates fast communication with the Metadata Server.
- Other Services – Many standard TCP/IP services are also needed to build Object Storage systems. For instance, NTP is used to synchronize compute nodes with the storage system, DNS is needed to simplify address translation and maintenance, and the various routing protocols allow the compute nodes to be separated from the storage system. Fortunately such services are well established in the TCP/IP world and by taking advantage of them, the Object Storage Architecture can benefit from their wide availability and interoperability.

OBJECT STORAGE OPERATION

Below is a description of the data flow and how a transaction in an object storage system moves through the system.

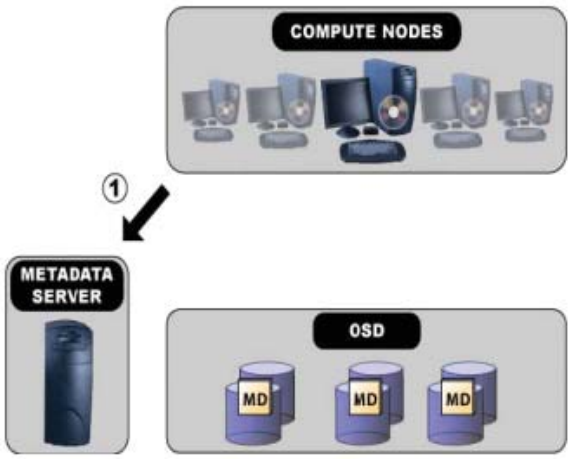
READ Operations

An Object READ transaction consists of the following:

1. Client Initially Contacts Metadata Server

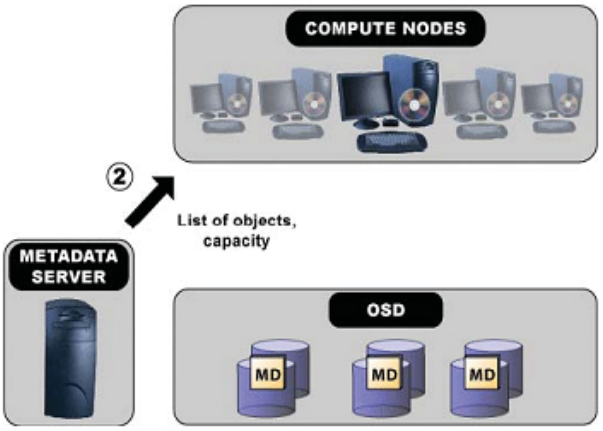
   Once the Distributed File System software is installed on the compute node and the file system is mounted, the compute node contacts the Metadata Server via an RPC (Remote Procedure Call) to a DNS name or IP address, configured in the compute node's etc/fstab file during installation. At this point, the Metadata Server validates and authenticates the identity of the node via NTLM, Kerberos or similar. The compute node reads the root directory object for the list of directories in the storage system and gets back a list of directory names.

   The compute node now understands the topology of the storage system and is able to contact the Metadata Server in order to acquire a Capability and Map. This is the first step of the process by which the Metadata Server (or set of Metadata Servers) manages the access policies for the entire storage cluster.



2a. Metadata Server Returns List of Objects

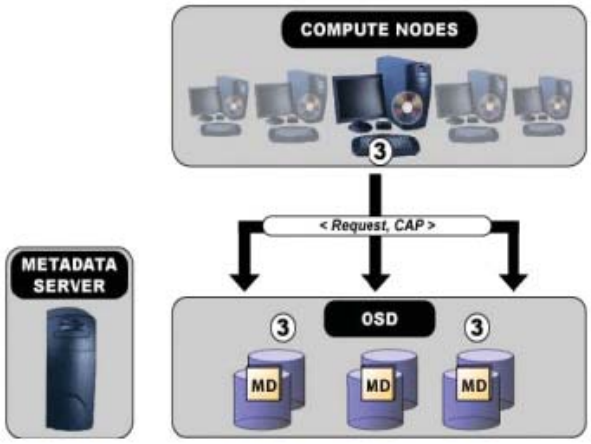
   Having received a request for access to an object within a Directory that it manages, the Metadata Server consults its Object Map and returns to the compute node a list of OSDs, by IP address, and the name of the component object that resides on each OSD. Assuming that the file is large enough, the number of OSDs will be related to the stripe width of the file. This gives the client the autonomy to communicate directly with the OSDs.



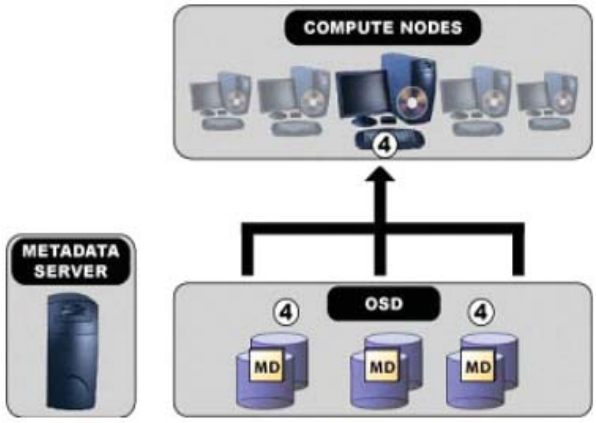
2b. Metadata Server Also Returns a security Capability

   The Metadata Server also sends a Capability, or security token, which authorizes the node to access the specific component objects, at specific offsets, with a specific set of permissions, for a specific length of time. This properly authenticates an untrusted node and restricts their ability to interact with the data stored on the OSDs. The node cannot maliciously or unintentionally (via a bug for instance) cause harm to data other than that allowed by the Metadata Server. The Metadata Server also sets a callback with the node that allows the Metadata Server to notify the node if another node is attempting to change the data in that file or portion of the file. If the node receives a callback, it must go back to the Metadata Server to renew its access to the file and then update its local cache from the OSDs.

This allows all nodes across the system to maintain cache consistency without resorting to a central lock manager that becomes a performance bottleneck in large systems.



3. Client Sends Read Requests Directly to OSD
Along with Capability
The node then packages the request in an iSCSI transaction to read all or some of the data that it is authorized for and sends it to the OSD. It also includes the Capability in the Read request, which is inspected by the OSD to insure that the requester is properly authorized by the Metadata Server to access the data.



4. Direct-data Transfer Between Client and OSD
In this example, the node has sent its read request to ten OSDs simultaneously. At that point, all ten of the OSDs have outstanding requests for component objects from the compute node. All of the OSDs begin transmitting the requested component objects in parallel, generating approximately 40 MB/s per OSD or a potential peak rate of 400 MB/s for this file (if the client could accept this data rate). Unlike a traditional system requesting a series of blocks, the Object Architecture allows very efficient communication about the desired transaction and allows the OSD to be intelligent about how it responds to the request. Where a traditional system may read ahead some of the adjacent blocks hoping to have the right data in cache, the OSD knows where the complete component object is, regardless of the physical location of the blocks on the disk, and can read ahead with a much higher likelihood that the data it puts in cache will be required for the next transaction.

WRITE Operations

An Object WRITE transaction is similar to the READ transaction described above and consists of the following:

1. Client Initially Contacts Metadata Server
As in the READ request, for the first step in the data flow, the compute node contacts the Metadata Server, is authenticated and makes a request to WRITE an object.

2a. Metadata Server Returns a List of Objects
The Metadata Server keeps a table of least utilized OSDs and returns a reference to two OSDs, by IP address, allowing the node to write some data and create a RAID 1 mirror.

2b. Metadata Server Also Returns a Security Capability
The Metadata Server sends a Capability, or security token, to the node authorizing the node to write a certain amount of data, with a specific set of permissions, for a specific length of time.

The Metadata Server again sets a callback with the node that allows the Metadata Server to notify the node if another node is attempting to change the data in that file or portion of the file.

3. Client Sends Write Requests Directly to OSD with Capability
   The node then packages the WRITE request, the Capability and up to 64K of data, in an iSCSI transaction to the two OSDs simultaneously.

4. Direct-Data Transfer Between Client and OSD
   If the node acquires more data to write (or if the buffer was more than 64K to start with) the node will request additional space from the Metadata Server and continue writing. For example, if there is a megabyte to write with a stripe width of ten OSDs, the Metadata Server will provide the map to the OSDs, and the Capability authorizing the write and escrowing the quota size for the write. The node will then write to all ten OSDs simultaneously with nine data component objects and one parity component for RAID 5 data protection. If the unit size of each component write is 64K, the first stripe will be 576K, requiring a second stripe. Instead of creating a new set of components for the second stripe, the existing components are expanded with additional data, reducing the number of transactions with the Metadata Server and providing the OSD greater autonomy to efficiently manage data layout.

OBJECT STORAGE ARCHITECTURE ADVANTAGES

While the Object Storage Architecture offers significant benefits for a wide variety of storage applications, it is uniquely qualified to solve the scaling issues that plague cluster computing environments. The primary advantage of the Object Storage Architecture comes from its shared-nothing cluster-based design, which allows the storage system to scale in performance and capacity just like the cluster compute system.

Performance

The Object Storage Architecture removes the central metadata manager bottleneck found in all other shared storage systems. NAS systems use a central file server as the metadata manager, while some SAN file systems use a central lock manager, but ultimately metadata management becomes a bottleneck. The Object Storage Architecture is fundamentally similar to a SAN, where each node has direct access to its storage devices allowing access at aggregate disk stripe rates. The Object Storage Architecture improves on the basic SAN design by getting rid of the RAID controller bottleneck allowing RAID stripes to be read in parallel and thus the aggregate RAID stripe throughput from a single file to be large. The benefit is magnified as you scale the compute cluster to large numbers of nodes. The total throughput for all of the nodes is eventually limited only by the size of the storage system and the performance of the network fabric.

This kind of very high throughput is particularly valuable in cluster computing applications that synchronize their reads or when the large number of compute nodes creates a high peak throughput demand to the storage sub-system. In a traditional system the WRITE commands generate a series of blocks written out to sectors and tracks on disk all of which need to be managed by the NAS filer head in order to optimize the placement of the data onto the media creating a significant burden on the filer head. The Object Storage node is sending data to the OSDs, which can autonomously

worry about optimizing placement of the data on the media. This minimizes the burden on the compute node and allows it to write to multiple OSDs in parallel, maximizing the throughput of a single client, but also allowing the storage system to handle the aggregated writes of a large cluster. This can be especially important in applications that use checkpointing where all compute nodes run to a barrier in the application and then write their physical memory out to storage.

Scalability

The Object Storage Architecture leverages the same fundamental principles as the original Beowulf compute cluster model. By distributing the workload to many intelligent subsystems, using the network fabric and sophisticated software to tie them together, scalability boundaries are eliminated and new types of problems and applications can be addressed. An Object Storage system has intelligent OSDs that have memory and processor that allow them to add storage processing power independently of the rest of the system. If an Object Storage system does not have sufficient storage processing power, adding OSDs will increase it almost linearly. Assuming the files are reasonably well distributed across the storage system, the additional OSDs add component object handling capacity independent of all other factors save two: network capacity and MDS capacity.

OSD Offloads 90% of Workload from Metadata Servers

Metadata management capacity is the typical bottleneck for any shared storage system, since it is the one place where all compute nodes and storage nodes must be coordinated. This is a key innovation in the Object Storage Architecture. As described in the Metadata Server section, there are two components to metadata: inode metadata - managing block layout on the media which accounts for approximately 90+% of the work in the system; and the file metadata – the hierarchy of file and directory names that the compute nodes need to navigate the file system, which accounts for the remaining 10%. The key benefit of the Object Storage Architecture is that it increases MDS scalability is to make the OSD responsible for its own inode metadata so that adding an OSD adds 90% of the metadata management resources needed to support the additional storage capacity of that OSD. Unlike a typical NAS server that gets slower as you add more disks, the Object Storage systems maintain a consistent level of throughput even as additional capacity is added.

The table below shows the results of a key experiment performed at Carnegie Mellon University (CMU) as part of the NASD program at the Parallel Data Lab, which provided the architectural underpinnings of today's Object Storage Architecture. This table compares two NFS servers; the first is a standard NAS server and the second, an Object-based server. It compares the number of MDS processor cycles required to support the most frequent NFS operations in a standard workload. As compared to a standard NFS server the Object-based MDS required only 7% of the processor cycles to accomplish a typical file sharing workload containing these key NFS operations. These results represent an order of magnitude improvement in MDS efficiency. This was accomplished by distributing the majority of the metadata management operations to the OSDs rather than executing them at the MDS where they represent a bottleneck to performance and scalability.

NFS Operation	Ops Count in top 2% by work (K)	NAS Server		Object (NASD)	
		CPU Cycles	% of CPU	CPU Cycles	% of CPU
Attr Read	792.7	26.4	11.8 %	0.0	0.0 %
Attr Write	10.0	0.6	0.3 %	0.6	0.3 %
Data Read	803.2	70.4	31.6 %	0.0	0.0 %
Data Write	228.4	43.2	19.4 %	0.0	0.0 %
Dir Read	1577.2	79.1	35.5 %	0.0	0.0 %
Dir RW	28.7	2.3	1.0 %	2.3	1.0 %
Delete Write	7.0	0.9	0.4 %	0.9	0.4 %
Open	95.2	0.0	0.0 %	12.2	5.5 %
Total	3542.4	223.1	100 %	16.1	7.2 %

This experiment showed how a single metadata server could be 10 times more efficient with the Object Storage Architecture. In order to completely eliminate barriers to scaling, multiple metadata servers can be clustered in the Object Storage Architecture, dividing the file system namespace among themselves and cooperate to provide service atomically, where there is no shared data with other MDSs, allowing the "shared-nothing" approach to scale.

Management

The distributed intelligence of the Object Storage Architecture provides many opportunities to minimize or eliminate time-consuming and error-prone management tasks often found in other high performance storage systems. The core tasks associated with managing and optimizing the data layout of the system are eliminated. For instance, new capacity is automatically incorporated into the storage system since an OSD that is ready to receive component objects when asked by the compute nodes. No LUNs need to be created, no partitions resized, no volumes re-balanced, and no file server to upgrade. Also note that RAID stripes are automatically expanded for new objects, to take advantage of the additional OSD without intervention by the System Administrator.

Security

Storage has typically relied on authentication of the clients and private networks to guarantee the security of the system, whether it's a fibre channel SAN or a SCSI array inside a file server. The Object Storage Architecture provides security at every level:

- Authentication of the storage devices to the storage system
- Authentication of compute nodes to the storage system
- Authorization for compute node commands to the storage system
- Integrity checking of all commands via CRC checks
- Privacy of data and commands in flight via IPsec

This level of security can give customers confidence that they can use more cost effective, manageable and easily accessible networks, such as Ethernet for storage traffic while improving overall storage system security.

Standards

The Object-based Storage Device interface standardization effort can be traced directly to DARPA sponsored research "Network Attached Secure Disks" conducted between 1995 and 1999 at CMU by Panasas Founder, Dr. Garth Gibson. Building on CMU's RAID research at the Parallel Data Lab (www.pdl.cmu.edu) and the Data Storage Systems Center (www.dssc.ece.cmu.edu), NASD was tasked to "enable commodity storage components to be the building blocks of high-bandwidth, low-latency, secure scalable storage systems."

From Dr. Gibson's prior experience defining the RAID taxonomy at UC-Berkeley in 1988, there was an understanding that it was industry adoption of revolutionary ideas that yields impact on technology. In 1997 CMU initiated an industry working group in the National Storage Industry Consortium (now www.insic.org). This group, including representatives from CMU, HP, IBM, Seagate, StorageTek and Quantum, worked on the initial transformation of CMU NASD research into what became, in 1999, the founding document of "Object-based Storage Device" working groups in the Storage Networking Industry Association (www.snia.org/osd) and the ANSI X3 T10 (SCSI) standards body (www.t10.org). Since that time, the OSD working group in SNIA

has guided the evolution of Object Storage interfaces, as member companies experiment with the technology in their R&D labs. Today the SNIA OSD working group is co-led by Intel and IBM with participation from across the spectrum of storage technology companies.
Panasas has made a commitment to continue to help drive the development of standards based on the Object Storage Architecture. Rather than the traditional approach of trying to build a business based on proprietary, closed systems, Panasas believes that customers will benefit if an industry is built around the Object Storage Architecture. Therefore, we are also working with key members of the file system community, some of who are already involved with Object Storage through the ANSI X3 T10 work. Panasas has developed a standard file system that can fully exploit the capabilities of the Object-based Storage Device with the vision that an Object Storage Architecture-based parallel file system eventually becoming as ubiquitous as NFS v3 is today. To that end, Panasas has committed to working with its partners to make sure that open-source, reference implementations of the file system are available and that the file system is driven through an open standards process. This is intended to lay the foundation for an industry around the Object Storage Architecture that has interoperable Object-based Storage Devices, Metadata Controllers and a parallel, object-based file system.

CONCLUSION

The Object Storage Architecture provides a single-system-image file system with the traditional sharing and management features of NAS systems and improves on the resource consolidation and scalable performance of SAN systems. This combination of performance, scalability, manageability and security could only be accomplished by creating a major revolution in storage architectures. The first product supporting the Object Storage Architecture was released as the Panasas ActiveScale Storage Cluster. It has since been deployed at multiple National Laboratories, key seismic processing organizations and well as biotech organizations. All are using Linux clusters to solve key scientific problems that were unattainable with monolithic supercomputers. They all expect to be able to address new types of problems that required high performance, scalable, shared storage that was not available to the market prior to the Panasas system. The Panasas Parallel Storage Cluster and the Object-based Storage Architecture have demonstrated that they can meet the challenge posed by the Beowulf/Linux cluster compute architectures, where traditional SAN, NAS and DAS-based products fall short.

panasas®

Accelerating Time to Results™

6520 Kaiser Drive Fremont, California 94555 Phone: 1-888-PANASAS Fax: 510-608-4798 www.panasas.com
1-888-PANASAS (US & Canada) 00 (800) PANASAS2 (UK & France) 00 (800) 787-702 (Italy) +001 (510) 608-7790 (All Other Countries)

©2005, 2007 Panasas Incorporated. All rights reserved. Panasas, the Panasas logo, and ActiveScale are trademarks of Panasas in the United States and other countries. All other trademarks are the property of their respective owners..

Information supplied by Panasas, Inc. is believed to be accurate and reliable at the time of publication, but Panasas, Inc. assumes no responsibility for any errors that may appear in this document. Panasas, Inc. reserves the right, without notice, to make changes in product design, specifications and prices. Information is subject to change without notice.