OBJECT ORIENTED DEVICES:

DESCRIPTION OF REQUIREMENTS

1. Introduction

It is hoped that this paper will stimulate discussion on the topics of Network Attached Storage and Object Oriented Devices - and encourage work toward defining the right path along which to implement NAS.

1.1 Purpose

This document argues for the value of and the requirements for Object Oriented Devices in support of Network Attached Storage. Though there may not seem to be a necessary connection between the two, Seagate feels that the latter immeasurably improves the former. In fact OOD may be required to get the full advantages of storage manageability and single system image that are key to achieving all the benefits of shared storage among a cluster of systems.

Although an object abstraction traditionally has been ascribed to software constructs, benefits accrue from applying an analogous concept to the organization of data on a storage device. It can be argued that these should not be termed objects. A more appropriate name is welcome; in the absence of one, it is hoped that the using the term objects will not get in the way of the discussion.

1.2 Definitions

Block Oriented Devices (BOD). A storage device operating mode in which its space is managed as an ordered set of fixed length data blocks, or sectors. This is the mode used almost universally today. This is the term that will be used to refer to a traditional data storage architecture, as opposed to the proposed Object Oriented Device.

Device, or Storage Device. A secondary storage unit that can store or retrieve data. Discs, tapes, CD-ROM's and array subsystems are examples of storage devices.

Network Attached Storage (NAS). One or more storage devices having peer connections with two or more computer systems.

Network Attached Secure Discs (NASD). The term used by CMU for its research on network attached storage. The NSIC project sponsoring the CMU research.

Object. The smallest visible unit of capacity allocation on a device operating in Object Oriented Device mode. An object on such a storage device consists of an ordered set of sectors associated with a unique identifier. Data is referenced by the identifier and an offset into the object. Conceptually similar to a file, it is allocated and placed on the media by the device itself, while the operating system manages its files and metadata in these object constructs, instead of managing sectors of data, as it does in most current architectures.

Object File System (OFS or **FFS).** An Object Oriented Device (see below) manages its objects with a filing system consisting of a single level list of the objects for each partition on the device. This is also called a Flat File System (FFS).

Object Oriented Devices (OOD). A storage device operating in a mode in which data is organized and accessed as objects rather than as an ordered sequence of sectors. Device or devices operating in this mode.

Requester. A computer system. An element in a cluster or network of systems which submits a request for action on a storage device. The term Requester is used as a general description for systems including both clients and servers, as either could be directly connected to NAS and

imposing workloads on it. A client whose workload goes to a server which in turn submits I/O requests to storage is not a requester; its server is, as it is the element actually doing I/O with the NAS

Sector. The smallest unit of data exchange with the device. A device will always read or write multiples of the sector size even in OOD mode. The difference between sectors in an OOD and in a BOD is that sector addressing in an OOD is relative to the first sector of an object, while in a BOD addressing is relative to the first sector on the device.

1.3 Some References

Borowsky, E., et al, “Eliminating storage headaches through self-management”, www.hpl.hp.com/SSP/papers/IWQoS97.pdf
Gibson, G, et al, “Filesystems for Network-Attached Secure Disks”. March 1997. www.cs.cmu.edu/Groups/NASD/ftp/NASD/CMU-CS-97-118.ps
Golding, R et al, “Attribute-managed Storage”, October 1995. www.hpl.hp.com/SSP/papers/MSIO.pdf
Soltis, Steven R. “Proposed SCSI Device Locks”. December 18, 1996. www.lcse.umn.edu/GFS/pubs/nasa96.ps

2. NAS Environment

2.1 Why Clusters

This section lays the foundation for developing the OOD requirements by positing a system architecture in which network attached storage solves fundamental problems or offers significant functional advantages over traditional architectures.

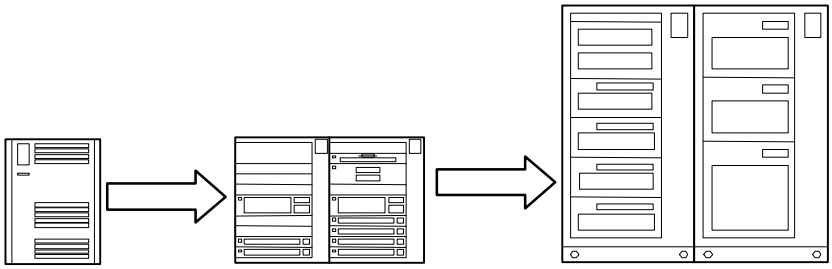

Figure 1: Mainframe Computing Model

The traditional progression for an end user is to purchase a system and - when additional processing capability is needed - to replace it with a bigger system. This has been the prevailing method for mainframes, in particular. At some points in this sequence traumatic discontinuities occur. If the user outgrows the architecture he started with, he may have to convert from one operating system to another or even from one vendor's proprietary architecture to another's. These entail huge costs for the organization in both dollars and employee time such that these conversions are avoided if at all possible. Another disadvantage with this model is the poor residual value of computer equipment. A system replacement often results in invested capital lost when the old system goes out the door. Moreover, larger systems tend to be sold in lower volume than smaller ones, which results in each new system having a higher cost of computing.

Consider the following alternative architecture.

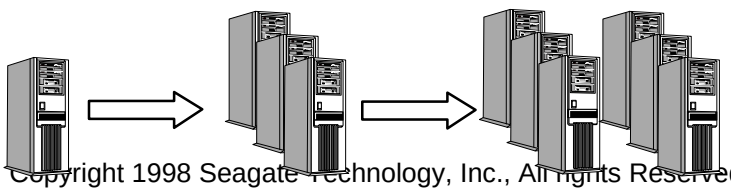

Figure 2: Cluster Computing Model

Replacing a mainframe computer with a cluster of smaller, standards based servers offers compelling advantages to the user. Since the cluster can start off as a single system the threshold to entry is lower. These smaller systems are sold in higher volume, making the cost of computing is less. With no dependence on proprietary architectures, the availability of equipment from multiple sources lets the user choose the best alternative with each purchase.

There are more advantages. The upgrade cost can be controlled more precisely by adding only the amount of additional resource required. If a different vendor has leapfrogged the original supplier in terms of offering what the user needs at the time of upgrade, he would be free to choose the best value without concern about migration or conversion to a new architecture. With the right architecture, there might never be a need for conversions from one OS to another, retraining of staff or developing new procedures - all of which have plagued mainframe customers in the past.

It would be so nice if the cluster computing model were as easy to achieve as it is to draw. There are fundamental architectural roadblocks to be overcome if the cluster model is to realize its potential.

While it is perhaps too simplistic to tie the continued importance of mainframes in the systems market to only two characteristics, there are two significant ones that prevent clusters of open architecture systems from pervasively replacing mainframes. One is the inability of clustered systems share data in a way that lets the cluster take on the workload that a single mainframe could. The other arises from the extensive experience the mainframe world has in managing storage and data. This experience has evolved into management software that simply has as yet no equivalent in the open systems markets. If clusters of systems are to replace mainframes, these two deficiencies will have to be remedied. This is where NAS comes in.

2.2 NAS Goal: Enable Scalable Clustering

NAS offers the prospect of solving both open system shortcomings mentioned above.

2.2.1 Scalable connectivity of storage to clients.

The ability to scale processors and storage independently and linearly is a fundamental goal of NAS. The NAS concept physically disassociates storage from processors. Devices are no longer peripheral components of a processor but a separate and equal architectural entity. Storage can be managed, changed and expanded with no impact on the processor configuration or operations - if NAS is done right. It should also liberate the processors in the same way. This is the easy part of scalability

In order for NAS to deliver fully on its scaling promise, there must be no overheads that increase appreciably faster than the rate of scaling. For instance, in loosely coupled shared processor systems today, this interprocessor communication activity goes up exponentially as the processors increase. This kind of overhead severely limits scalability, and NAS research aims to eliminate this barrier to scaling.

Making a cluster of servers process the same workload a mainframe accepts is easy unless it requires that all the servers process transactions on the same data. Unless a cluster can run these enterprise applications, like an Fortune 500 manufacturing system, an airline's reservation system, or a financial institution's complete inventory of transactions a cluster cannot replace a mainframe.

If, on the other hand, each processor in the cluster can accept any transaction that would have arrived at the mainframe and process it, there is hope for the cluster model. The challenge to

the cluster's ability to do this is giving the servers the ability to share data, including having concurrent update rights in a way that increasing the number of processors does not destroy performance. With this, any server in the cluster could process any transaction, and processors can be added as needed with no fall off in performance. This is what a cluster model must demonstrate if it is to replace the mainframe's.

Specifically, NAS aims to do the following:

1. Increase scalability of high volume servers by applying clusters of them to large, monolithic applications that traditionally have been serviced only by mainframes or tightly coupled processors. Network Attached Storage not only will provide sufficient connectivity for a large group of servers to have access to all the storage they require - in terms of both distance and number, but also provide a mechanism so that many servers sharing update access to common storage can do so without the excessive overhead that accompanies clusters today.

2. Enable heterogeneous computation. That is, let systems running different operating systems access a common pool of storage and share data. Perhaps an end user with a cluster of servers in a data mining application finds that a supercomputer doing statistical analyses on the data would be of value. If he could just wheel up such a system and have it act upon the data already in place, it would be much more cost effective than if he had to purchase a completely separate system, including a duplicate set of storage devices. Given the size of many data warehouses, the latter would not be possible; the data movement time alone would be prohibitive.

The fact that OOD make the storage data organization free for the operating system should make it possible for a server running any operating system to access the data. (There is still the problem of the internal data structures of a file, which must be reconciled at the application level.)

3. Increase the I/O bandwidth to Requesters. Making storage available to Requesters at storage network speeds without "channeling" it through a server can mean more performance for high data applications. (This is a principal objective of the work being done at CMU.)

4. Improve performance by eliminating the necessity for a server to translate a request into physical device accesses. When several servers are accessing the same OOD data, the metadata never travels over the Interconnect. Though there are a lot of factors affecting how caching is done and how it is affected by the object abstraction, there are some clear opportunities for benefit. A common cache of the metadata in the OOD serves all Requesters, with no coherency issues. The number of I/O operations is correspondingly reduced as a single metadata retrieval can service requests from multiple Requesters. What is more, with the OOD understanding quite precisely which objects are in use and which are not, the cache space can be more effectively utilized. Typically, even system caches are not tied closely to the file system and do not reflect what the OS knows about files being open or closed. (Of course, sometimes this can be misleading; a file can be closed and then be needed quite soon thereafter. A cache that is not as closely tied to the file system may happen to fortuitously retain data that otherwise would have been discarded. Even this condition can be accounted for in the OOD environment, with an indication that an object being closed will be soon accessed again.)

2.2.2 Storage Manageability

The second fundamental problem is that of storage management. The NAS view is to make all components of the storage architecture participate in the management. By breaking down management from a huge CPU task to many small activities assigned to the lowest possible level and directing those activities by means of simple attributes expressed by the user, storage takes on the responsibility for managing itself.

NAS should make it possible to:

1. Make storage be as much self-managed as possible. This will eliminate the associated drudgery now imposed on the operating system by such requirements as space management. It should also make scalability more linear by increasing storage management capability at the same rate as the number of storage devices increases. Today an operating system assumes the responsibility for allocating space on a disc drive, reclaiming space from deleted files, and - in some cases - deallocating bad sectors. Doing this for one drive is not difficult, but a server with dozens of drives could find it consuming quite good portion of its processing cycles. Object oriented devices would take over space management, eliminating any increase in OS overhead associated with the number of drives on a system. A server with dozens of disc drives would get the benefit of all those devices contributing horsepower for managing the storage resource.

2. Support attribute based management by having the devices take action based on the properties assigned to a given object or set of objects. The many engines (each storage device having some usable processing power to apply to this task) available would be put to additional use by helping with more than just space management. They could contribute to breaking the task of data management into many simple, small functions performed concurrently. For instance, an attribute could be set for an object that called for that object to be versioned. Every time the object was closed after an update, the storage device could automatically keep the old version of the object, while giving the new one a separate identifier.

Similarly, an attribute might be set to indicate that an object should be exported after it was updated. This could cause the device to start a sequence of actions independent of the application processor that would result in a copy of that object being sent to another device. Extrapolating this very simple process, an entire storage complex could participate in a more timely and less intrusive backup function. If the devices knew enough about what work was going on, they could make sure that an export operation only took place when an object was in a consistent state. This is a little more complicated for some data; complex data structures like data bases may require that multiple OOD take coordinated action to them up, but even this can be handled by a straightforward extension of the basic capability.

3. Support quality of service management by having the storage devices be as knowledgeable as possible about their own conditions and informing the appropriate service of those conditions or acting in response to those conditions as guided by a policy assignments. Suppose a disc drive has several levels of transfer rate performance to offer a Requester. The device could allocate an object to whatever zone was most appropriate given the users interest in cost versus performance. This could as easily apply to NAS that had combinations of mirrored, RAID 5, and unprotected storage, the user selecting the residence for particular data depending on his requirements for resiliency. A quality of service management system could interpret user choices into attributes associated with particular objects, leaving the device to act on those attributes and use its resources accordingly.

4. Facilitate the dynamic reconfiguring and expanding of storage. Whenever additional space was required, there would be a central authority to which any sever in the network could turn to find additional space, or to find all storage devices available to it. This could be the basis for operating systems being more dynamic and flexible as to what hardware they are operating with at any point. It need not be the peripheral set that was present at system generation time or even power up time.

5. Present as nearly as possible a single system view to users. If all Requesters accessing a NAS configuration - and all users contacting the Requesters - get a single view of the data regardless of which member of a cluster they are connected to, clusters of systems can look and feel much more like a single system than otherwise. If clusters of high volume servers are to replace large monolithic systems, this will be key.

6. Present as nearly as possible a single system view for management purposes. If clusters are to work as well as mainframes, the management tools must let the user control the configuration as if it was a single, coherent computing facility, even if there are multiple vendors and operating systems represented.

2.2.3 Inherent security

Object oriented storage offers new opportunities for improving system security. At the very least it could prevent random writing of sectors not associated with a file operation. The most common type of virus is one that changes the contents of a single sector on a PC's hard disc. Any software module has the ability to do this, as there is little or no security in most PC systems.

More than this, new security provisions could be architected into the OOD definition so that any spying on disc or tape traffic could not make available any usable information, either for espionage or sabotage.

2.3 Description of NAS

2.3.1 Elements of a NAS configuration

A NAS based system could be constructed of equipment and software from many different vendors. Its constituents collaborate in a way that should make it appear to the world outside as one large computer system. As NAS is used in this paper, four components constitute a NAS architecture: OOD, Requesters, File Server, and Interconnect.

The Object Oriented Devices are the storage components of the system. They include disc drives, RAID subsystems, tape drives, tape libraries, optical drives, jukeboxes, and any other storage device to be shared. They must have an I/O channel attachment to the Requesters that will access them.

The Requesters are the servers or clients sharing and directly accessing the OOD.

A File Server performs management and security functions such as request authentication and resource location. It is key to the self-management capability of NAS; The OOD depend on it for management direction, while the Requesters are relieved of storage management to the extent the File Server assumes it. In smaller systems, a dedicated File Server might not be feasible; a Requester may take on the responsibility for overseeing the operation of the NAS environment. Where the security and flexibility that the File Server brings are not desired, or where an overriding need for performance calls for the cluster of Requesters to talk directly with - and only with - the OOD, a File Server might not be present.
The Interconnect is the physical infrastructure via which all NAS components communicate. It must have properties of both networks and channels. It must have distance and addressability adequate to connect all components in networks; it must also have the low latency and flow control properties of a channel. It must have the manageability features of mainframe class peripherals.

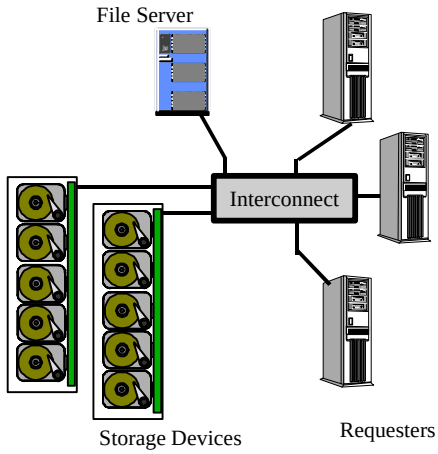

Figure 3: NAS Configuration

2.3.2 Overview of NAS Operation

2.3.2.1 Startup.

When the NAS is powered up all devices must identify themselves either to each other or to a common point of reference, such as the File Server or the Interconnect. The Interconnect offers network management techniques to be used for this. For instance, in a Fibre Channel based NAS, the OOD and Requesters would log onto the fabric. Any component wanting to determine the operating configuration could use fabric services to identify all other components. From the File Server, the Requesters learn of the existence of the storage devices they could have access to, while the OOD learn where to go when they need to locate another device or invoke a management service like backup. Similarly the File Server can learn of the existence of OOD from the fabric services.

Depending on the security practice of a particular installation, A Requester may be denied access to some equipment. From the set of accessible storage devices it can then identify the files, databases, and free space available.

At the same time each NAS component can identify to the File Server any special considerations it would like known. Any device level service attributes could be communicated once to the File Server, where all other components could learn of them. For instance a Requester may wish to be informed of the introduction of additional storage subsequent to startup, this being triggered by an attribute set when the Requester logs onto the File Server. The File Server could do this automatically whenever new OOD are added to the configuration, including conveying important characteristics, such as it being RAID 5, mirrored, etc.

2.3.2.2 Accessing NAS

When a Requester must open a file, it may be able to go directly to the OOD, or it may have to go to the File Server for permission and location information. To what extent (pardon the pun) the File Server controls access to storage is a function of the security requirements of the installation.

First, the case where the installation is physically secure. That is, there is not a requirement to protect the transmission of command and data between Requester and OOD. There might still

be a File Server present for management functions, but one that does not oversee the Requester I/O.

Here a Requester is in a position to access and create objects directly on an OOD. It can open, read, write and close objects just as if, they were natively attached to the Requester.

A typical sequence might go something like this. The Requester reads from an OOD one or more well-known objects that reveal the logical volumes or partitions of the device (more on this later) and how to start looking at objects. He then opens and reads an object, which might be a root directory. From this it is straightforward to find other objects, based on the contents of the directory. The Requester repeats the process until the desired data is located. This can be accessed just any file on a BOD could, the difference being that the data is referenced by the Object ID and a displacement within the object, not an LBA whose address is relative to the start of the storage device.

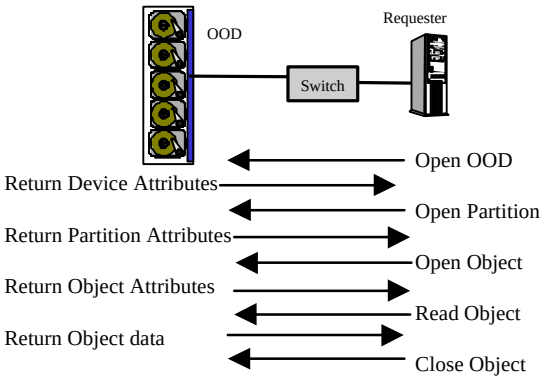

Figure 4: Read Object Sequence

In the second case, where security is required, the File Server interposes itself into the I/O chain to the degree necessary for the desired level of protection. A Requester must first go to the File Server for permission to perform a set of I/O operations. (The File Server may well also withhold OOD location information from the Requesters at initialization time in support of the security requirement.) The File Server accredits the request by returning sufficient information to allow the Requester to communicate directly with an OOD.

The OOD will also be informed of the installation security policy when they log into the File Server. Based on this they will not allow an I/O request unless it is properly constructed - including encoded with a valid permission. All NAS elements collaborate to enforce the security of the system.

Security requirements may demand both protecting data while in flight - using encryption - and validation of requests against security criteria at the OOD before servicing.

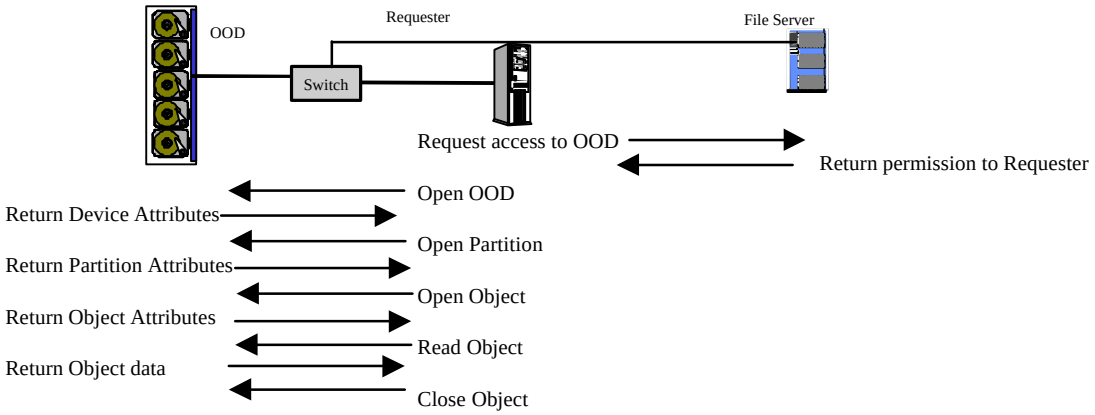

Figure 5: Read Object Sequence using File Server

Though the sequence of operations seems quite similar to the first case, they are quite different, especially in terms of the payloads associated with each command. In the secure case, both commands and data may be encrypted. In addition, the permission information must be added to the command parameters.

2.3.3 Data Sharing and Concurrent Update

NAS proposes solving the data-sharing problem by moving to an optimistic control scheme where Requesters can act as though there is no conflict unless there is actually simultaneous attempts to access the same records by multiple Requesters. They learn this from the OOD themselves when attempting to commit updates. An attribute can be set by a Requester to establish that it intends to update certain data. The attribute is reset after completion of the writes. Only if another Requester attempts to update data for which the attribute is set is there a need to resolve a conflict. This approach should greatly reduce the inter- system traffic database servers generate today to maintain data consistency.

When the cluster starts up, all Requesters can open the shared data base objects and are ready to update them. Attribute bits can be used to indicate the appropriate granularity of control. With something like VIA compliant host interfaces it will even be possible for applications to directly establish integrity control.

2.3.4 Differences with CMU

Carnegie Mellon University has published extensive descriptions of their view of NASD and their research on the subject. While Seagate is a supporter of that work and one of its principal sponsors, there are some differences in perspective, which should be made clear.

A primary motive for NAS at CMU is the elimination of the server bottleneck. For high data rate applications, if the clients can communicate directly with the storage devices, the data need not suffer the bandwidth loss associated traveling through a server. While this has some application in the marketplace, such as in collaborative video editing, the need we here about from our customers is for servers to share data.

CMU also advocates the prospect of any client anywhere talking with any storage device anywhere. As a disc and tape device manufacturer, we see this as a problem. It opens up the door for huge numbers of clients accessing a single disc drive simultaneously for arbitrarily long periods. While this might be possible, it is not something we hear from customers as needed. Moreover it flies in the face of the fundamental design philosophy of a drive. A disc drive is intended to be a limited resource device. The buyers of disc drives expect them to be as low

priced as possible. The electronics of a drive is architected with this requirement in mind. Essentially making a server of a drive leaves the expectation that the drive has the properties of a server, namely that it can support very large numbers of connections of infinite duration. Adding even small amounts of additional hardware - if not needed - to a drive adds significant unacceptable cost due to multiplicative effect of large numbers of drives on a system. Rather, the shared storage and storage management shortcomings of servers are what are cited as barriers to progress.

Fortunately, the differences in perspective do not result in substantively different views of the problems nor in different research topics.

2.4 Why Objects

It might seem that there is nothing in the problems of clustering or NAS that dictates the use of Object Oriented Storage to solve them. In fact object oriented storage does so much to improve the cluster architecture that Seagate feels it is essential to realize the full benefits of the clustered system architecture. Though this is far from doing the subject justice, here are several reasons for this position.

1. Objects really make the self-management of storage possible. Without the device having sufficient knowledge of the resident data, it cannot assume the responsibility for managing space. Devices could not participate in any attribute management without the knowledge of what constitutes a meaningful subset of its space or when it is appropriate to take action. More effective management will result from the devices able to participate intelligently.

2. The sharing of data can be controlled more intelligently when the device knows what constitutes an entity. If two systems were to share BOD, all the metadata activity would have to be controlled for concurrent access. In an OOD much of the metadata activity is opaque to the systems, which need only concern themselves with access conflicts to user data. Also space management being done by the device eliminates any contention or confusion that could arise from two systems trying to manage space on the same device at the same time.

3. In order for a cluster of systems to truly act as a single computing facility, a single system view of the data is essential. The object-based organization makes this easy.

4. Heterogeneous computing is made much easier by an object abstraction. There is essentially no commonality between OS's metadata structures. OOD make it possible to at least have an organization that any OS can interpret.

5. While there are a lot of issues revolving around performance in a clustered system architecture, some obvious performance benefits come with the Objects.

    a. The metadata never leaves the device eliminating a certain amount of I/O.
    b. The device knows which objects are closed or open, and is able to use that information to more effectively cache data.
    c. Prefetching can be much more effective as the device knows the layout of the object being read. The device can more effectively determine sequential access patterns.
    d. The cache in the device can hold metadata once for multiply systems accessing it.
    e. The device can participate in quality of service decisions, such as where to locate data most appropriately. It can only do this if it has responsibility for allocating storage. By comparison, almost no OS can allocate data by zone on a disc drive.

3. Description of Requirements

3.1 Functionality

Data will be organized into fully self-defined and self-contained objects, which the storage device will be responsible for managing. Putting security aside for a moment, the object oriented architecture makes it possible for any Requester to inspect a device to discover what files exist there and request one or more, without knowing anything about the device physical characteristics or even the operating system which created the object.

The object abstraction must not hide or otherwise make unavailable to a File Server or Requester information valuable to the optimal exploitation of the storage. For quality of service and self-manageability reasons the object semantics should make available details of the device's capabilities without infringing on the device's sovereign self-management. For instance, a typical disc drive today has several strata of transfer rates. The specifics of these should be known to a degree sufficient to allow a policy to take advantage of them without affecting the device's space management mechanism. This could take the form of service levels and capacity quotas for each.

Objects look and work very much like traditional files. They are not the same. Files are constructed in an operating system context. They are often collected in a hierarchical directory structure, which again is operating system specific. Objects will be operating system independent, with (almost) any OS able to overlay its file system on the object abstraction.

The following sections describe the functions that will be needed on a device to support objects. This is preceded by a short overview of how a storage device might organize the objects.

3.1.1 Device Organization

3.1.1.1 Space Organization

In order for object oriented devices to provide the same functionality as is delivered by an operating system with block oriented devices, the storage space must be manageable to a similar degree, and the OOD will have the constructs to do that. This necessitates having an organization layer on the OOD above objects. It is almost universal that operating systems provide for allocating disc space into one or more mutually exclusive regions, called partitions. The same should be available on an OOD. Within a partition a Requester can create objects. There is a single list of objects for each partition; any directory hierarchy is overlaid on the OOD OFS by the operating system. This makes it possible for any operating system to use this method, and makes much easier the use of one device by multiple operating systems.

The structure within a partition is envisioned to be a simple, flat organization. Onto this any operating system can map its OS structures. An unsigned integer of some specified size will be the permanent signature for a given object on a device. Associated with this will be a certain number of blocks.

The device will maintain an object list for each of its partitions and space management information for the entire device.

While there will likely always be a need to support the BOD and relative sector addressing, there is a definite security problem with allowing a device to switch from OOD to BOD and back. To insure data security the change from BOD to OOD must - and the change from OOD to BOD should - obliterate any object organization on the device.

3.1.1.2 Object Identifiers (ID's)

The identifier associated with an object is chosen by the device and returned to the Requester in response to the command to create an object. The ID will be an unsigned integer of a specified

length. The length could default to a size set by the drive or it could be set as a device attribute. A subset of ID's would be reserved for well known objects, special uses and special functions.

3.1.1.3 Well Known Objects

A well-known Object is one that always has a specific object ID. In some cases it must exist on every device or in every partition. Using these ID's starts the Requester traversing the OFS. The following list has possible Object identifiers associated with each for illustration purposes.

3.1.1.3.1 Device Control Object (DCO) - Object 1

This object contains the attributes the OOD must maintain that relate to the device itself or that relate to all objects on the device. The attributes are maintained by the SET ATTRIBUTE function. There will be one DCO per device.

Type	Name	Bytes	Semantics
Security	Clock	8	monotonic counter
	Master Key	8	master key, controlling device key
	Device Key	8	device key, controlling partition keys
	Protection Level	1	defines protection options
Partitions	Partition Count	1	Number of partitions on device
Device attr	Object Attributes	8	defines properties associated with all objects on device

Figure 6: Possible Device Control Object Attributes

3.1.1.3.2 Partition Control Object (PCO) Objects 2 - 255

This object contains the properties of a single partition. It describes not only the partition but also any object attributes that pertain to all in the partition. The OOD will have one PCO for each partition defined on the device.

Type	Name	Bytes	Semantics
	Master Key	8	Encryption keys
	Current Working Key	8	
	Previous Working Key	8	
Part. attr	Object Attributes	8	defines properties associated with all objects in partition

Figure 7: Possible Partition Control Object Attributes

3.1.1.3.3 Partition Object List (POL) Object 1 in Partition n

When a partition is created, a second well-known object will also be built - the point of departure for navigating the OFS. It will have the same identifier in every partition. This object consists of a list of the object ID's for all objects resident in this partition. The user will have the option to allocate user space associated with each ID.

Field	Bytes	
OBJECT ID	8	ID used for any OPEN, READ, WRITE, CLOSE on this OBJECT
User Data	N	POL Attribute sets this, use GET ATTRIBUTE to learn value

Figure 8: Possible Partition Control Object Attributes

3.1.1.4 Object

Associated with every object in a partition is a set of attributes. The following is the set of attributes posited by CMU. The length of each attribute is a Seagate estimate.

Type	Name	Set by	Bytes	Semantics
Access Control	Access_control_version	Set Attribute	4	Changing ACV revokes capabilities
Clustering	Nearby_Object	Set Attribute	4	Locate this Object near another
Cloning	Copied_Object	NASD	4	Object was created Copy Object
Size	Logical_Size	NASD, Set Attribute	4	Largest offset written
	Blocks Allocated	NASD, Set Attribute	4	Number of Blocks Allocated for Object
	Blocks_Used	NASD, write	4	Number of Blocks used to store data
	Block_Size	NASD, Set Attribute	2	Number of Bytes per Block
Time	Create_Time	NASD	8	Timestamp of Object creation
	Modify_Time	NASD	4	Timestamp of last data modification
	Attribute_Modify_Time	NASD	4	Timestamp of last attribute modification
File system	FS_Data_Modify_Time	NASD, Set Attribute	4	Timestamp of last data modification
	FS_Attribute_Modify_Time	NASD, Set Attribute	4	Timestamp of last attribute modification
	Object_Attributes	NASD, Set Attribute	4	Bits of Object properties for self-mgmt
???	FS-Specific	Set Attribute	256	256 bytes uninterpreted by NASD

Figure 9: Possible Object Attributes

3.1.1.4.1 Device Association Memberships (MDA) Objects 256 - n

To adequately manage objects spanning multiple OOD, there will an object describing an association. An important facility is the ability to define cross OOD relationships. Devices already intercommunicate; there will likely be even more of this in an OOD environment. The MDA defines all OOD that are members of a particular association of OOD. This could be used for something like a mirror paired devices or a RAID set.

Name	Bytes	Semantics
Association Identifier	2	Unique ID of this set
Association Type	2	Kind of Association
Membership List	n	
Association Identifier	2	
Association type	2	
Membership List	n	

Figure 10: Possible Device Associations Object Attributes

3.1.1.5 Navigating the Object File System

Step	Command	Partition No. Parameter	Object No. Parameter	Comments & Data Returned
1	GET ATTRIBUTE	0	0	device attributes (DCO)
2	GET ATTRIBUTE	0	1	Attributes of Partition 1 (PCO)
3	GET ATTRIBUTE	1	1	Partition 1 POL (optional step)
4	OPEN	1	1	POL
5	READ	1	1	List of OBJECT's in Partition 1
6.	CLOSE	1	1	CLOSE POL
7	OPEN	1	n	OPEN object (ID from POL)

Figure 11: Steps to finding an Object in a Partition

This illustration uses the Object ID's posited in the above descriptions. The first access to the device should be to get the device attributes. Since the device itself is not within a partition of the device, the partition ID used in the GET ATTRIBUTE command is zero (0). The Object ID is also zero (0). The returned information identifies the number of partitions, which can be each inspected with GET ATTRIBUTE commands for each partition of interest, using the appropriate partition ID and an Object ID of zero ((0).

When Objects in a specific partition are of interest, the POL is used to get a list of those Objects. An OPEN on the ID from the POL will start the Requester using the OBJECT.

3.1.2 Device Functions

The OOD must support requests to provide or store data for a Requester. Moreover, it must assume responsibility for other functions that would have been done somewhere else - most likely in the operating system - in a BOD architecture. Space management, as well as the maintenance of attributes associated with the objects, will have to be done by the device.

The following are the operations that an OOD will have to perform as its contribution to the NAS environment. Each description lists the service provided by that function and the information that must be passed to the drive in order for the drive to perform that function. The information that could be returned by the device to the Requester is also listed to develop a clearer idea of what the function entails.

3.1.2.1 OPEN

This will communicate to the device the fact that a certain Object is to be accessed or a new one created. It will also indicate the kind of operations to be performed on the object. The OPEN offers the opportunity to deliver to the OOD valuable information. It introduces the fact that a new object will be created, for which the device must take some action. It can set quality of service attributes or thresholds, which lets the device inform the Requester if the service cannot be provided. For instance if a new object minimum have a minimum degree of contiguity, an OPEN parameter could direct the device to allocate space in units of that size

3.1.2.2 Open-Create

The OPEN-CREATE will cause the device to allocate another Object ID, which will be returned. This the Requester will use to issue writes to the new object. In addition, the Requester can specify several options, among which might be:

1. Make this file password protected.
2. Quality of service thresholds
3. Encrypt this object.
4. Specify lock level - i.e. sector as well as object lock
5. Specify versioning
6. Mirror support - cause all updates to be mirrored onto another object.
7. Allocate space in units of a specified minimum size
8. Set rights (as in UNIX)

The needed information includes:

1. Permission information
2. Partition of the device in which the object is to be created
3. Options

The information returned could include:

1. Quota or capacity available on the device
2. Status of request
3. ID of new object

A special instance of this command includes all data associated with an Object, so that in one command an object can be created, written and closed.

3.1.2.2.1 OPEN-Update - includes extend

The OPEN-UPDATE will allow the Requester to read and write the specified Object. This also provides for extending the length of the object.

The needed information includes:

1. Permission information
2. Partition ID
3. Identifier of the object to be accessed
4. Type of action to be taken: Update
5. Options

The information returned includes:

1. Status of request
2. Length of Object
3. Quota or available capacity

3.1.2.2.2 OPEN-Read only

The OPEN-READ is obvious. It allows no writing to the file.

The needed information includes:

1. Permission information
2. Partition ID
3. Object ID
4. Options

The information returned includes:

1. Status of request
2. Length of Object

3.1.2.3 READ

The device is requested to return data from the specified object.

The needed information includes:

1. Permission information
2. Object identifier
3. Options
4. Partition ID
5. Starting location of blocks to be read
6. Number of blocks to be read

The information returned includes:

1. Status of request

2. Length of data
3. Data

3.1.2.4 WRITE

This will cause the specified number of blocks to be written to the designated object at the relative location also specified. Information required is similar to that for a READ. A WRITE might also cause other events to occur. If parity support was called for on the target device, a write could automatically cause an XOR to be performed on the data and the parity data to be written to one or more previously specified parity devices (via a DA).

An option on this function would cause all outstanding writes that have not been physically put on the media to be written.

The needed information includes:

1. Permission information
2. Object identifier
3. Partition ID
4. Starting location of blocks to be written
5. Number of blocks to be written
6. Options
7. Data

The information returned includes:

1. Status of request

3.1.2.5 CLOSE

This will cause the Object to be identified as no longer in use by a given Requester. Any changes as a result of writing to the Object, if not already written to the media, will be so at this time. The attributes for a newly created object will be updated at this time. The CLOSE could also specify the residual cache status in the Requester. If the Requester can inform the OOD that data is still being cached for the closed Object or no longer being cached, the OS's can retain cache information for those applications where files will be closed and opened again in quick succession. At the same time the device will be able to keep track of who may have to be informed in the event of coherency collisions, should another Requester request access to this Object.

The needed information includes:

1. Permission information
2. Object identifier
3. Options

The information returned includes:

1. Status of request

3.1.2.6 REMOVE

Delete the Object. The ID becomes usable again.

The needed information includes:

1. Permission information
2. Partition ID
3. Object identifier

4. Options

The information returned includes:
1. Status of request

3.1.2.7 CREATE PARTITION

Partition the drive into one or more regions. All space need not be accounted for. Regions can also span zones. The operation would implicitly establish an Object list for the partition.

There are two orthogonal uses for partitions. The first is a tiling arrangement with partitions true divisions of the storage space on a device. This would be used to divide the space by service levels, such as data rate. They could not be resized, but could be removed and recreated. The second is a logical partitioning where the intent is more to organize the objects than manage the space. These partitions could be resized dynamically.

Regardless of the type of partitioning, this function will also create POL, which cannot be removed. This object will serve as the starting point for navigating the objects in the partition.

The needed information includes:
1. Permission information
2. Options
3. Partition ID
4. Initial space allocation

The information returned includes:
1. Status of request
2. Partition map

3.1.2.8 REMOVE PARTITION

This is the function that will remove a partition and implicitly deletes the POL. Any Objects listed in this POL will be deleted.
The needed information includes:
1. Permission information
2. Options
3. Partition ID

The information returned includes:
1. Status of request
2. Partition map

3.2 Manageability

3.2.1 Overview

If management policies can be communicated to the device so that it can act independently to execute them, the result will be not only less human intervention required but also tighter and more timely control.

Consider the case of weekly backup. Systems are usually backed up during an idle period on weekends, so that the system availability is not interrupted during the business week. This also produces a backup that is guaranteed to be consistent. This window has been gradually shrinking at the same time system capacities have been exploding. It has become an almost impossible problem trying to find time to interrupt a system long enough to backup possibly terabytes of data.

By taking action on an object based on attributes assigned to it, The OOD could inform a backup function whenever an object has reached the correct state for its backup to be taken. The backup of all files could be spread over a longer period - during which others are still being updated -without affecting data integrity.

Other attributes that could invoke action by the OOD include encryption, compression, versioning, parity redundancy and HSM style migration. In each of these, the device would only have to be informed of the policy with respect to a specific object or set of objects. It could then perform the function itself or inform an agent designated to provide that service.

Compression and encryption could be done right on the drive, so that only the fact that one is required for an object need be communicated to the device. For a management function that must go off the drive, such as HSM, not only the policy is needed but also the identification of an agent to perform the function.

To make this most efficient it will probably be necessary for there to be associations established among objects, so that those with the same attributes or with dependencies can be identified. Consider a database consisting of 6 files, none of which can be backed up until either all have been closed or until one designated as the object on which all of the others are dependent has been closed. A File Server may be needed to manage this kind of relationship between Objects.

In addition it will be necessary to establish inter-device dependencies, as in the case of a RAID parity set. By making it possible to establish groups in which one device makes certain that the rest of the group has the same essential properties, management of the group will be more efficient and effective.

3.2.2 Manageability Operations

3.2.2.1 EXPORT Object

The EXPORT function will be needed for the device to support storage self-management and attributes-based management. It will enable the device to take action based on rules expressed in attributes associated with a given object. It could be used to initiate a backup or support versioning of objects to other devices.

This function will copy an object to another device by issuing OPEN-CREATE, WRITE's and a CLOSE to that device, transferring the object.

The needed information includes:

1. Permission information
2. Object identifier
3. Options
4. Target Device ID
5. Target Partition ID

The information returned includes:

1. Status of request
2. New Object ID

3.2.2.2 GET OBJECT ATTRIBUTES

The function retrieves for the specified objects the information on permissions, etc. It is also used to interrogate device wide policies. It should be possible to retrieve attributes for multiple Objects with a single GET OBJECT ATTRIBUTE operation. For instance, a single request could return the Device Object and all Partition Object Attributes.

The needed information includes:
1. Permission information
2. Object ID or ID list
3. Options

The information returned includes:
1. Status of request
2. Attributes of object

3.2.2.3 SET OBJECT ATTRIBUTES

The function sets permissions, etc. for a specified Object.

The needed information includes:
1. Permission information
2. Object ID
3. Options

The information returned includes:
1. Status of request
2. Attributes of object

3.2.2.4 DEVICE LOCK OPERATIONS

Object locks could be supported at the block, object, partition and device level. The LOCK mechanism provides for inter-server access resolution. These can be used for scheduling concurrent updates as well as prohibiting access during maintenance functions. Though these are really just instances of the READ ATTRIBUTE and SET ATTRIBUTE operations, more detail is provided as they are critical to the sharing of data among a cluster of Requesters.

3.2.2.4.1 READ LOCK Attribute

This function will inspect the Object to see if a particular LOCK is set.
The needed information includes:
1. Permission information
2. Object, Partition or Device identifier
3. Lock parameters
4. Options

The information returned includes:
1. Status of request
2. ID of requester owning lock

3.2.2.4.2 SET LOCK Attribute

This function will attempt to atomically inspect a lock, and - if it is not set - set it with the Requester's ID. This function can also be used to UNLOCK.
The needed information includes:
1. Permission information
2. Object, Partition or Device identifier
3. Lock parameters
4. Options

The information returned includes:
1. Status of request
2. ID of server owning lock

3.2.2.4.3 RESET LOCK Attribute

This function will be used to reset a LOCK in the event a server owning a LOCK no longer is functioning.
The needed information includes:

1. Permission information
2. Object, Partition or Device identifier
3. Lock parameters
4. Options

The information returned includes:

1. Status of request
2. ID of server owning lock

3.2.2.5 GET, SET DEVICE ASSOCIATIONS

These functions define or interrogate relationships among devices. This is necessary for inter-device communications. (A possible implementation would be one of the devices being identified as the “master” or first of set, with the others being dependent members of the set. The first of set would be responsible for disseminating to the other members changes in set attributes. Other members would reject attribute settings if they were not from the first of set.)

If storage is to be as much as possible self-managed, it must be possible for the device to perform a self-inspection. This must extend to its membership in a larger device group.

The needed information includes:

1. Permission information
2. Options
3. List of members and attributes

Information returned includes:

1. Status of request

3.3 Security Topics

Unlike the previous sections, which attempt to describe what will be required in a device, especially a disc drive, to support an object abstraction, this section merely introduces some issues for consideration. As the security requirements, we feel, are not yet sufficiently understood or described; it is premature to suggest what will be needed in a device to satisfy the security needs of a full NAS implementation. It is hoped that this section will lead to more definition of what is needed, which will in turn stimulate progress in this area.

As system configurations grow more dispersed, the threat to security increases. Network attached storage makes it easier to configure the equivalent of a server with elements spread over a campus or city - or the Internet. The Interconnect becomes correspondingly more susceptible to compromise. NAS should do its part to prevent this increased exposure with more sophisticated security capabilities.

3.3.1 Permissions

The File Server can gate access to storage by controlling to which Requesters it gives the credentials needed to get a response from an OOD. The File Server also dictates to the OOD that they must only honor I/O requests that adhere to the installation security policy.

For the storage interface this entails protocols that protect against corruption or snooping of traffic. The functions described above provide for the ability to include sufficient protection to insure that the OOD can validate the request as a legitimate one.

The keys underlying the permissions security capability can be communicated to the OOD by the SET OBJECT ATTRIBUTES function. If the appropriate level of security is set for an OOD, the OOD will check every I/O command for security compliance. It is conceivable that some applications will use security and some not.

There is a great interest in remote redundancy. If a server cluster had some devices located in another facility, it may be desirable to define a higher level of security for communication with them, but not for local traffic - to avoid the performance loss that would accompany the more complex commands.

3.3.2 Cryptographic Support

If a device must perform secondary operations on an object - such as compression, encryption or decryption - there could be a huge performance penalty. There would also be some unavoidable cost for this. The device might have to write the original object to the media, then reread it pass it through the appropriate encryption logic and rewrite it. Managing several of these concurrently while also servicing a queue of I/O requests puts quite a burden on that logic and on the overall performance of a device that today's architectures would be hard pressed to carry.

The added cost could be a problem. I suspect the market will have to decide if it is willing bear that across the board. It is almost impossible to offer it as an optional feature on a drive. Either the industry will buy into the concept or it will not.

3.3.3 Audit Trails

The necessity of logging an audit trail of NASD activity to itself could also pose a problem. It could be ameliorated by having the option to log the audit trail on another network device dedicated to that function. Even so, the bandwidth consumed by a sizable number of devices logging records would obviously reduce the application bandwidth.

It might seem that a single extra message for each I/O is not a big deal, but in a transaction environment it is a BIG deal! All I/O is essentially a message and the I/O subsystem is typically message bound. We already have some experience with this kind of thing on a drive where the SMART feature causes logging of environmental data. It can have a significant impact on performance. And the SMART data is only summary information.

3.3.4 Clocks

Each device will have a readable monotonically incrementing clock to be used for timestamping secure messages and objects. Either this clock must be synchronized system wide, or the server will have to accommodate the discrepancies in values from device to device.