



The Evolution of File Systems

How the Digital World Manages its Data

Christian Bandulet, Principal Engineer, Sun Microsystems

SNIA Tutorial Basics



Table of Contents

Abstract	1
Basic Architecture of Open Systems	1
Operating System Taxonomy	2
File System Taxonomy	2
Local File Systems.....	2
SAN File Systems.....	5
Network File Systems	6
Wide Area File Systems.....	7
NAS Aggregation and Virtualization.....	9
NFS Version 4.1/pNFS.....	10
NAS Grid/NAS Cluster.....	12
File System Futures	13
Conclusion	20
About the SNIA	20



Table of Figures

Figure 1: The File Systems Evolution.....	
Figure 2: Architecture of an Open Systems Operating System	
Figure 3: Vnode Layers in File Systems.....	
Figure 4: Local File System Hierarchy.....	
Figure 5: File Attributes in the File System.....	
Figure 6: Local File Systems and Applications	
Figure 7: Scaling the File System.....	
Figure 8: Client I/O Requests in Shared File Systems	
Figure 9: The SAN File System.....	
Figure 10: Clustered SAN File System.....	
Figure 11: The Network File System	
Figure 12: Application Activity in the Network File System.....	
Figure 13: NFS File System Server and Clients.....	
Figure 14: Client-Server Distributed File System.....	
Figure 15: Aggregated Storage in File Systems.....	
Figure 16: Dedicated Storage in 4 Segments.....	
Figure 17: Scaling Out Storage	
Figure 18: NAS Aggregation.....	
Figure 19: NAS Virtualization	
Figure 20: Out of Band NAS.....	
Figure 21: pNFS Metadata Server in Redundant Configuration.....	
Figure 22: Referral Technology	
Figure 23: Global Namespace in Referral Configuration.....	
Figure 24: Federated File System: Global Name Space.....	
Figure 25: NAS Cluster	
Figure 26: NAS Cluster with Shared Storage.....	
Figure 27: NAS Cluster with Dedicated/Captive Storage.....	
Figure 29: The Data Matrix	5
Figure 28: The Storage Cloud.....	
Figure 30: Semi-Structured Data.....	
Figure 31: File System Metadata.....	
Figure 32: Object ID in an Object-Aware File System.....	
Figure 33: Binary Large Objects (BLOBs) in Relational Databases	
Figure 34: File Object Indexes	
Figure 35: Application Content Repositories.....	
Figure 36: File-Based Content Repositories.....	
Figure 37: Storage Clouds of the Future.....	
Figure 38: File Systems and Objects Complement Each Other	



Abstract

File Systems impose structure on the address space of one or more physical or virtual devices. Starting with local file systems over time additional file systems appeared focusing on specialized requirements such as data sharing, remote file access, distributed file access, parallel files access, HPC, archiving, security etc.. Due to the dramatic growth of unstructured data files as the basic units for data containers are morphing into file objects providing more semantics and feature-rich capabilities for content processing. This presentation will categorize and explain the basic principles of currently available file systems (e.g. Local FS, Shared FS, SAN FS, Clustered FS, Network FS, Distributed FS, Parallel FS, etc.). It will also explain technologies like Scale-Out NAS, NAS Aggregation, NAS Virtualization, NAS Clustering, Global Namespace, and Parallel NFS. All of these files system categories and technologies are complementary. They will be enhanced in parallel with additional value added functionality. New file system architectures will be developed and some of them will be blended in the future.

Basic Architecture of Open Systems

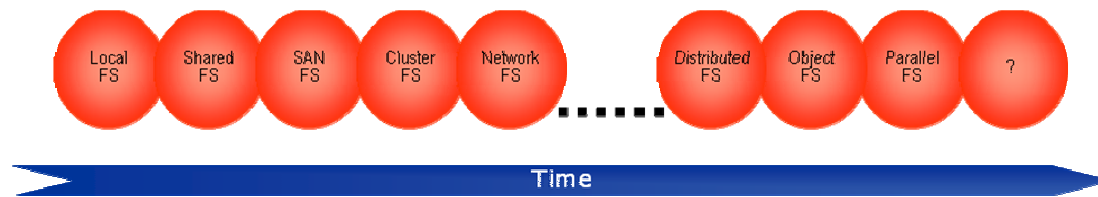


Figure 1: The File Systems Evolution

File systems have evolved over time. Starting with local file systems, over time additional file systems appeared focusing on specialized requirements such as data sharing, remote file access, distributed file access, parallel files access, HPC, archiving, etc. The diagram above (Figure 1) does not reflect the exact sequence in which the files system types appeared. Some of them actually appeared in parallel. It is also not the intention of this diagram to indicate that a new file system replaces its predecessors. Instead they are targeting complimentary objectives.

The diagram on the right (Figure 2) illustrates the basic architecture of an Open Systems operating system. The red horizontal bar in the middle represents the border between user and kernel address spaces. User space is the unprivileged area in an operating system where CLI interfaces and user applications run. File system related commands such as **ls** (list files), **mv** (move files), **rm** (remove files), **cp** (copy files) etc. actually use system calls, such as **open()**, **close()**, **ioctl()**, **read()** and **write()** etc.. These system calls transfer control of the computer from user space to kernel space, and place it in a privileged

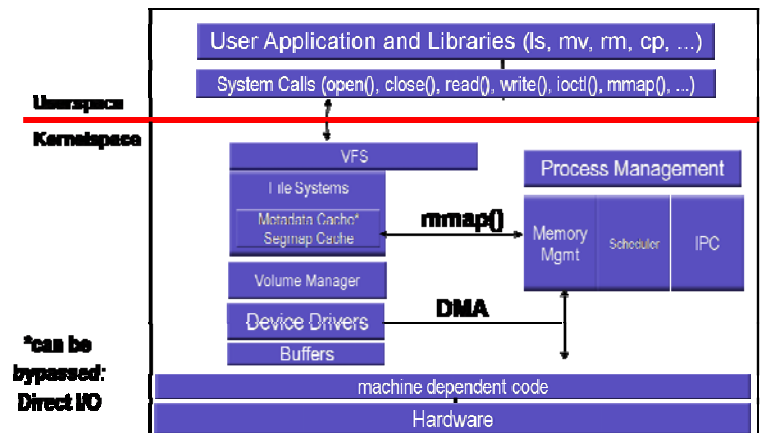


Figure 2: Architecture of an Open Systems Operating System



operating mode.

The first kernel space layer encountered in executing file system related commands is the vnode layer. A *vnode* is an object in kernel memory that speaks the UNIX file interface. Vnodes can represent files, directories, FIFOs, domain sockets, block devices, or character devices.

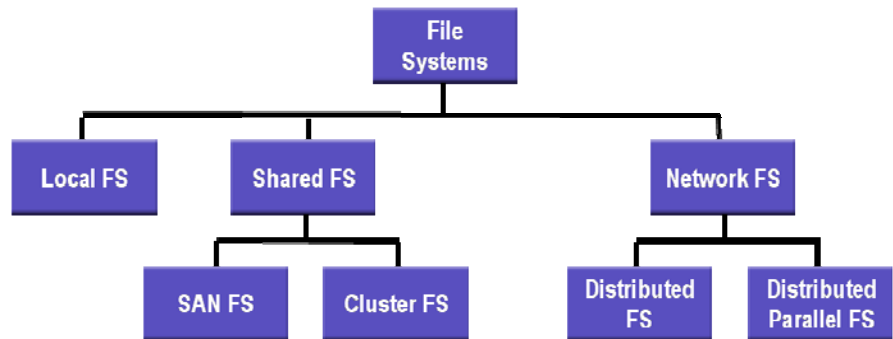


Figure 3: File System Taxonomy

A single operating system can run multiple file systems, for example, UFS and ZFS, concurrently. The vnode layer is the common interface that makes this possible.

Each file system has a mechanism to cache user data and metadata. If necessary the operating system's file system buffer cache can be bypassed (aka Direct I/O). Some modern file systems also incorporate a volume manager. In other cases, the volume manager is a separate component.

After traversing the I/O and SCSI driver stacks, requests are issued to hardware (network interface cards or host bus adapters) that connects to a storage area network or directly to storage devices.

On the right side of the diagram, three major operating system components are represented:

- The inter-process communication (IPC) mechanism enables kernel modules such as file systems to communicate with each other via special shared memory segments or global sections using mutual locks and semaphores and so forth.
- The scheduler is responsible for scheduling tasks and processes in a multi-tasking, multi-processing operating system.
- The memory management system maps between the virtual addresses used in applications and physical random access memory (RAM) addresses in the operating system. An alternative to accessing a file via a file descriptor, is to map it into virtual or physical memory and access it using the much faster pointer operations (`mmap()`). `mmap` is a POSIX-compliant Unix system call that maps files or devices into memory. It is a method of memory-mapped file I/O. It naturally implements demand paging, because initially file contents are *not* entirely read from disk and don't use physical RAM at all. The actual reads from disk are performed in "lazy" manner, after a specific location is accessed.

File System Taxonomy

The Wikipedia online encyclopedia lists over 70 different file systems. There are thought to be about a thousand different file systems on the market. Clearly, a taxonomy is needed to categorize file systems so that their properties can be studied. There are three different major families of file systems: local, shared, and network.

Shared file systems can be further sub-categorized as SAN file systems and cluster file systems. Network file



systems likewise can be categorized as distributed and distributed parallel.

Local File Systems

The typical local file system is co-located with applications on a single server. The file system's storage is either directly connected to this server or accessed via a Storage Area Network (SAN).

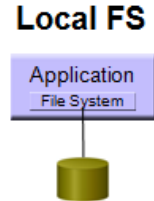


Figure 4: Local File System

The left part of Figure 5 illustrates that the local file system is organized as a hierarchy consisting of a root directory, sub-folders, sub-sub-folders etc.

The major task of the file system is to map each file's human readable name to the blocks of storage that hold its data. The right hand side of the diagram represents the storage. The blue rectangle in the middle represents the inode data structure that accomplishes this mapping.

The inode is a complex structure, but fundamentally it holds pointers to data. If a file becomes very large there may be indirect pointers to other pointers to data, and even triple indirect pointers (pointers to pointers to pointers to data).

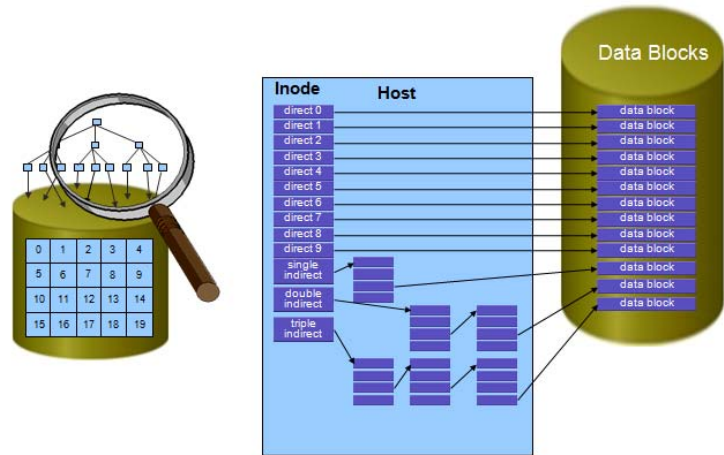
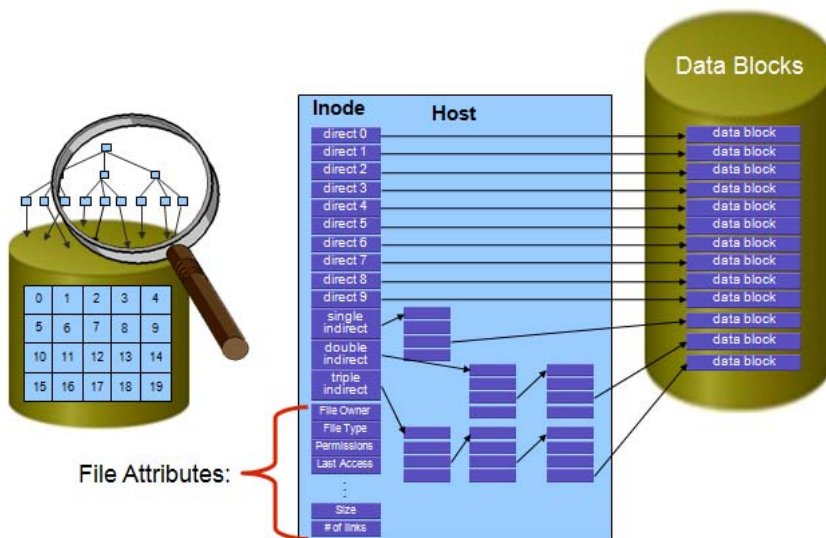


Figure 5: File System Inode Structure



The larger a file, the more pointers and indirect pointers are necessary to map it. The important point is that physical data locations are completely separate from its logical structure. For example, a storage device is not aware of which data blocks make up a particular file. This information is contained in the file system's inodes.

Figure 6: File Attributes in the File System



In addition to pointers to data blocks, inodes contain file attributes. In UNIX systems, the `ls -l` command displays file attributes including owner, group and world access permissions, time of most recent **access** and modification, and soft links associated with the file, etc.

The diagram below (Figure 7) illustrates the primary limitation of local file systems in enterprise applications — different servers running different applications, each with its own storage and own local file system. These “islands” of storage, with separate name spaces are an inherent property of local file systems.

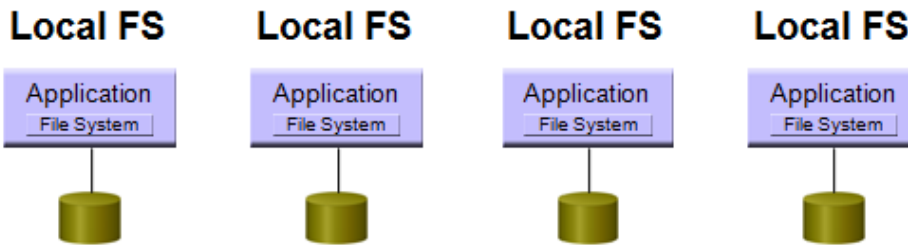


Figure 7: Local File Systems Islands

Local file systems can scale *up*. If additional storage, processing, or bandwidth capacity is required, the next larger disk array, server, or network switch can be installed. Scaling *out*, by means of additional processors, is not possible, however, because it creates additional islands of data.

Scaling out, also called horizontal scaling, is accomplished with the second category in the taxonomy, the shared file system, which is also called the global file system.

Shared file systems enable data sharing among different servers running different applications or possibly multiple instances of the same application. This requires *shared storage devices*, that is, physical storage devices able to connect to multiple servers simultaneously. It also requires a much more sophisticated technology — shared data, meaning not only is shared access to physical devices required, but also concurrent access to the data on them, as it is

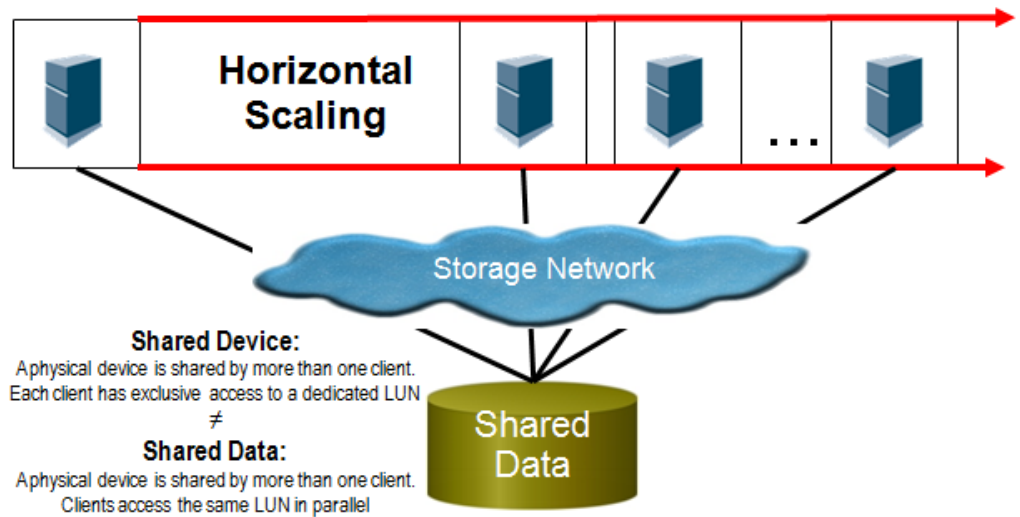


Figure 8: Shared Data vs. Shared Devices



structured by a file system.

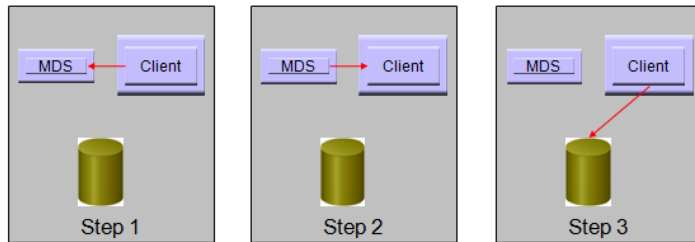


Figure 4: Client I/O Requests in Shared File Systems

With shared file system technology, client I/O requests are transactional, as they are with databases. Transactions have three steps. In the first step (Figure 9) on the left-hand side of the diagram, the client requests from the metadata server (MDS), permission to access a file, and information about its location. In the second step, the MDS responds to the client. In the third step, the client reads or writes file data from the storage device.

SAN File Systems

The scenario we are discussing is actually more complex because in a shared file system, several clients might access the same data concurrently. This requires additional locking and caching techniques to prevent data inconsistency. Moreover, for availability reasons, redundant metadata servers are often required, usually in active-passive mode. A metadata server might be a dedicated system, or alternatively, a process running on the system running the shared file system software.

The shared file system environment depicted in the diagram at right (Figure 10) is sometimes called a SAN file system. In this scenario, clients might be both widely dispersed as well as heterogeneous (not running the same operating system). This is essentially a master/slave architecture, with metadata servers playing the role of master. Clients and MDS are interconnected by a dedicated low-latency system area network.

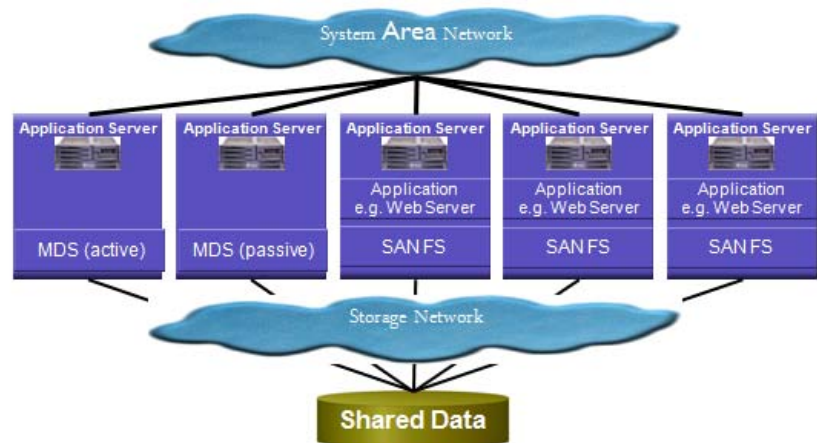


Figure 10: The SAN File System



Cluster File System

Another variation (Figure 11) of the shared file system is the cluster file system. The difference is that the metadata server is not a dedicated machine but the metadata service is distributed across all participants, all clients within this ecosystem. So it's a kind of distributed database and you need some cluster technology mechanisms in order to take care about cache coherency and locking. Typically in this environment, it's a peer-to-peer or symmetric architecture, because all clients are equal in this context and each client knows the logical structure of the data. But here, mostly you find products with homogeneous operating systems, so you cannot mix AIX, Linux or Solaris machines in this architecture and the distance between the nodes is pretty limited.

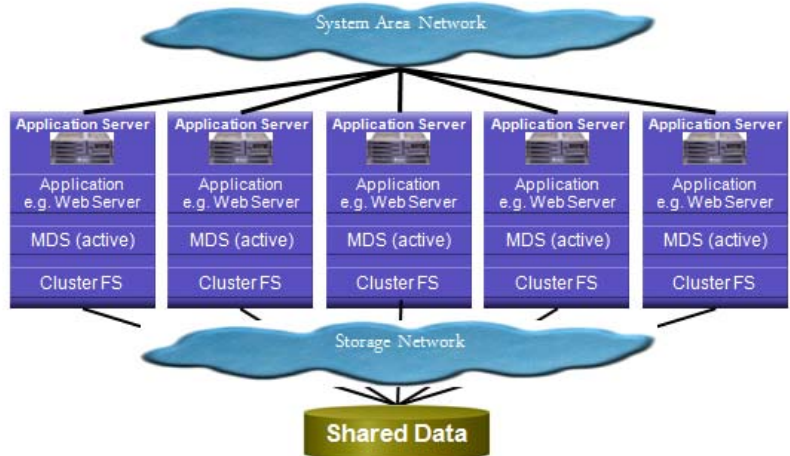


Figure 5: Clustered SAN File System

It is important to notice that although shared file systems enable scale-out of the file system the actual file system's access of the shared storage device itself does not scale. Scale-out storage devices will be explained later.

Network File Systems

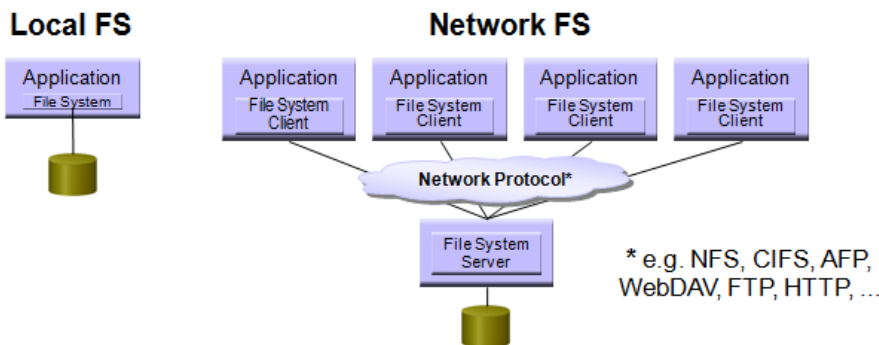


Figure 12: The Network File System

Next in the taxonomy is the family of network file systems. The left side of the diagram (Figure 12) depicts the local file system discussed earlier. The network file system represented on the right side is similar to the aforementioned SAN file system, the major difference being that in a SAN file system clients use a block based protocol such as ATA or SCSI to access storage devices. In a network file system,



all communication is between a client and a server, using a protocol layered on a local area network (typically TCP/IP), such as NFS, CIFS, WebDAV, HTTP or FTP. The network (also known as proxy) file system is in essence a computer network in which clients communicate with file servers.

Wide Area File Systems

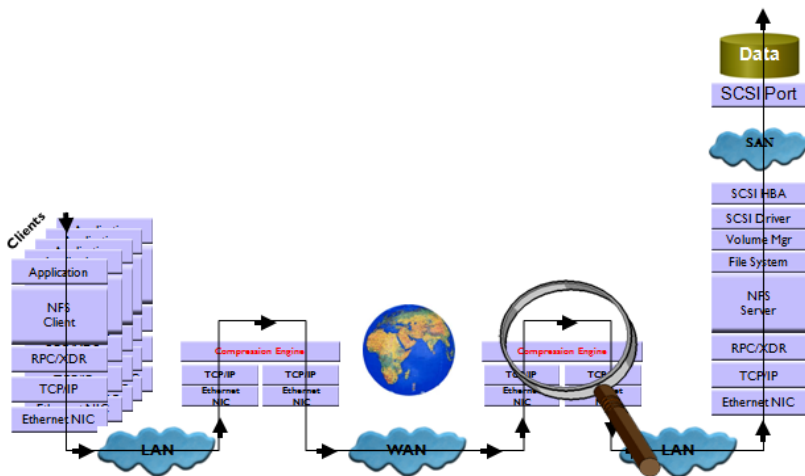


Figure 13: Application Activity in the Network File System

This diagram (Figure 13) “zooms” into the network file system architecture. On the left hand side, the client is running the application that uses software (e.g., NFS client) layered on a message-response and transport infrastructure (remote procedure calls or RPC and TCP/IP in the case illustrated in the diagram). Network interface cards (NICs) connect the client to the local area network (LAN). Similarly, the entry point to the server is the NIC. The server’s TCP/IP-RPC stack ultimately presents requests to the NFS server, which processes client read and write requests for any local file systems.

The local file system interacts with a volume manager, the SCSI stack, the hardware, whether SAN-attached or direct-attached storage to access the data blocks of the requested file.

NFS was developed in the mid-'80s by Sun Microsystems. Since that time, IT has changed. Enterprise IT has become globally distributed; it is common for organizations to have interconnected branches throughout the world.

NFS was originally developed for use within a local area network; not for wide area infrastructures. On a wide area network or on the internet, high latency is probable. The data storage industry has developed the wide area file system concept to solve this latency problem. The wide area file system is actually not a file system; it's a bundle of services that include network compression. Gateways at both ends of a connection compress the data that flows over the network.

Wide area file system technology might optimize TCP/IP operations or Ethernet packets; it could be application aware, or do write-behind or aggressive pre-fetching to give the impression of LAN-like latency over a wide area infrastructure.



NFSv4

The current version of NFS is NFSv4. NFSv4 improves the ability to deal with global namespaces and remote file access. For example, NFSv4 includes the concept of *pseudo-file systems*. The diagram (Figure 14) represents a file server running NFSv4, presenting three different local file systems. With earlier versions of

NFS, this would have required three mount points at each client wanting to access the three file systems. The pseudo-file system integrates multiple file systems with a kind of soft link, presenting just one single mount point (“share”) to the outside world. Client import a single share rather than three.

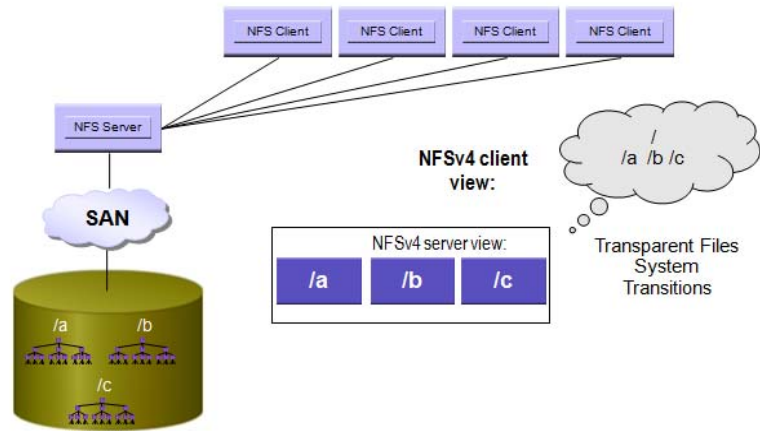


Figure 14: NFS Pseudo-File System

Client-Server Distributed File Systems

The client-server *distributed file system* (below, Figure 15) is one type of network file system. Clients read and write files from what appears to be a single file server. But in this case, the files and directories are distributed across several file servers. In the diagram, for example, three file servers, each of them having one single file, A, B and C. To clients, it looks like a single file system with one single mount point that provides access to files A, B and C distributed across file servers.

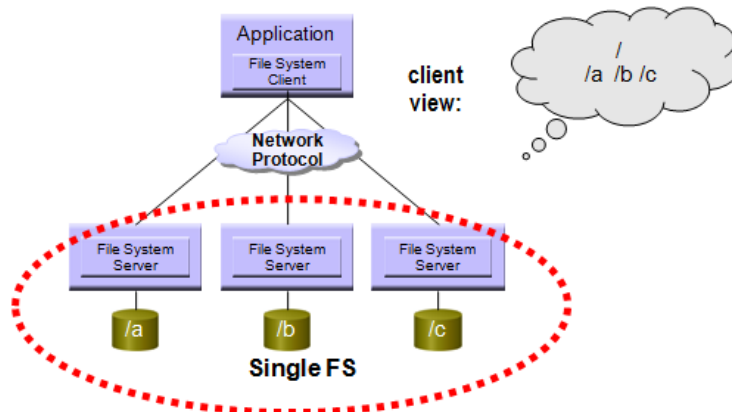


Figure 15: Client-Server Distributed File System



Distributed Parallel File System

Distributed and parallel is a special form of distributed file systems in which files are distributed across file servers and additionally, each file is segmented into small pieces distributed across the file servers. So the granularity is much smaller. The lower part of the diagram illustrates that the targets that hold pieces of these files run in conjunction. This is often called aggregate storage, redundant array of inexpensive nodes (RAIN), or network RAID technology. All file servers together provide a single or global namespace.

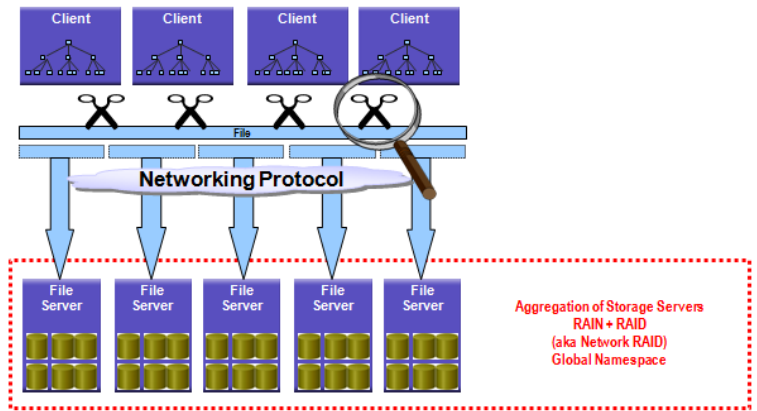


Figure 16: Aggregated Storage in File Systems

Figure 17 illustrates 16 nodes with dedicated storage and a single file stored in four segments, which are distributed across the nodes in what appears to be accidental order. Actually there are sophisticated algorithms that guarantee high availability of these parts, for example using enhanced parity protection. Furthermore I/O performance is enhanced by replication of files and load balancing mechanisms.

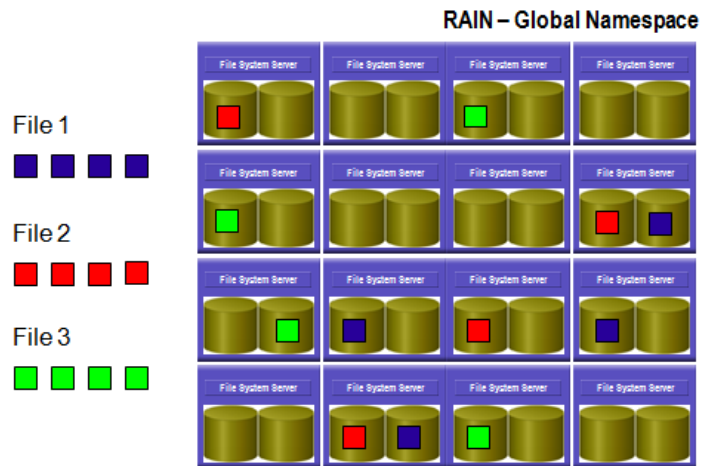


Figure 17: Data Distribution and RAIN

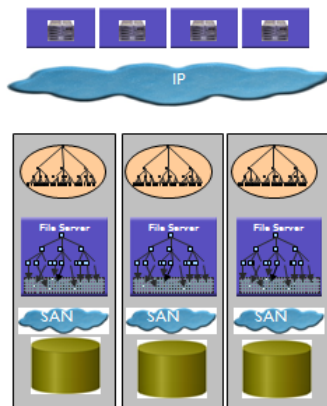


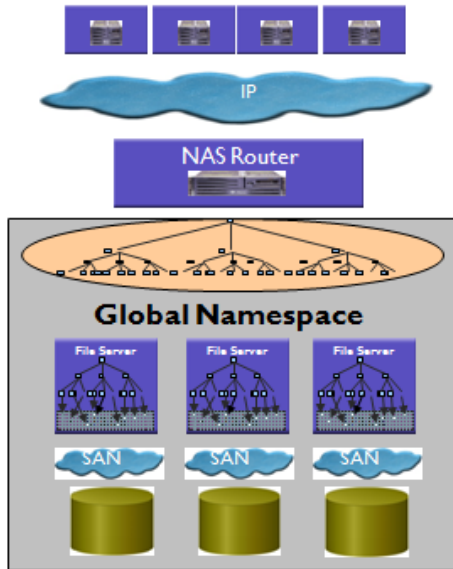
Figure 18: Scaling Out Storage

We turn next to *scale-out storage*. Scale-out storage combines network storage with distributed and parallel file systems. It is available in three major variations. The left-hand side of the diagram (figure 18) represents the IT ecosystem of a typical large enterprise. The enterprise starts with one file server, which eventually fills to capacity. The enterprise buys a second, a third and a fourth file server, each of



which becomes a separate name space, or island of storage.

It is possible to administer these NAS systems from a single point, but the namespaces are separate. Scale-out NAS addresses the problem of horizontal scaling. One begins with a NAS head, and adds a second, third, fourth, and so on. All NAS systems combine to create a single global namespace, or image of data. Clients that access the single data image are unaware that it comes from a federation of NAS systems.



NAS Aggregation and Virtualization

There are a variety of methodologies to abstract storage to clients, commonly known as storage virtualization. The first approach introduced is NAS aggregation. NAS aggregation adds a layer called a NAS router or gateway to the data path. Its task is to be a single point of entry for clients. At the back end of a NAS router there may be several (possibly heterogeneous) NAS systems. The NAS router is in-band. It presents a global namespace, and can serve as a load balancing mechanism by moving data transparently from one file server to another. To a client, the NAS router is a kind of yellow pages; clients are not aware of physical file locations.

Figure 19: NAS Aggregation

The next variation is NAS virtualization. This is an out-of-band solution (Figure 20). It resembles the SAN file system, because the client first goes to a metadata server to get authorization and file location information. A file need not be located on a single server; it might be striped, concatenated, replicated, or otherwise distributed across several servers. Each file server in this variation has dedicated storage; file servers do not share storage or data.

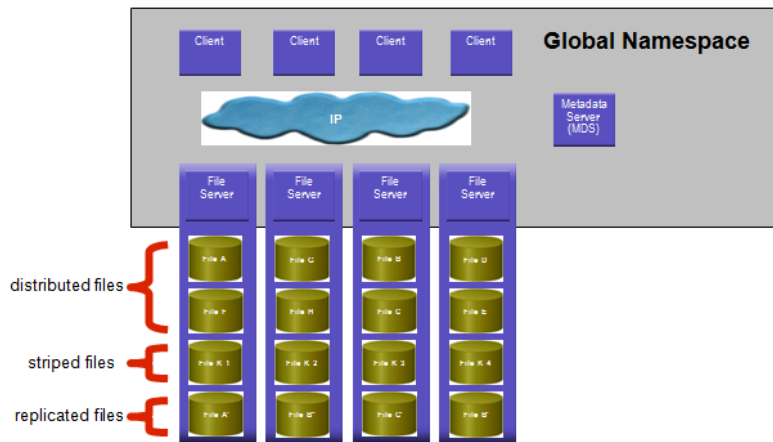


Figure 20: NAS Virtualization



NFS Version 4.1/pNFS

One instance of this technology might be the next minor version of NFS, version 4.1, which includes the so-called pNFS extensions. The left hand side of the diagram (Figure 21) represents traditional architecture, with clients, a NAS appliance, and a NAS file server connected to the storage area network.

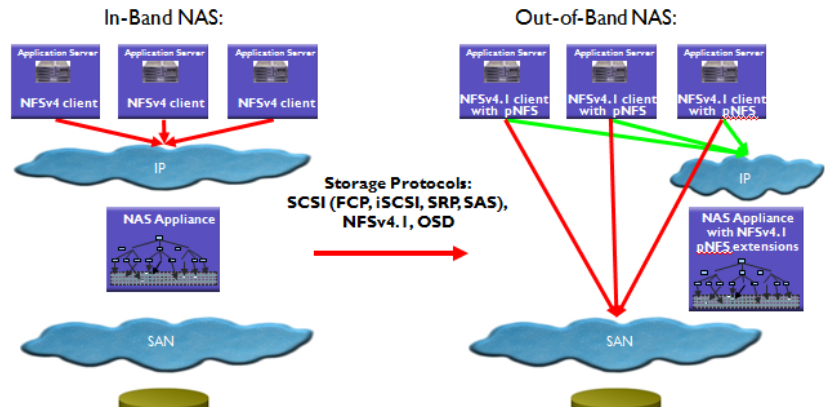
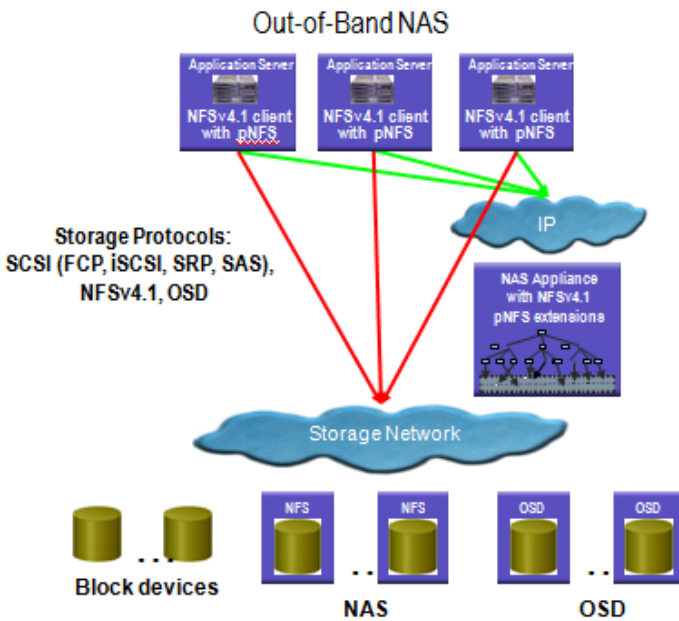


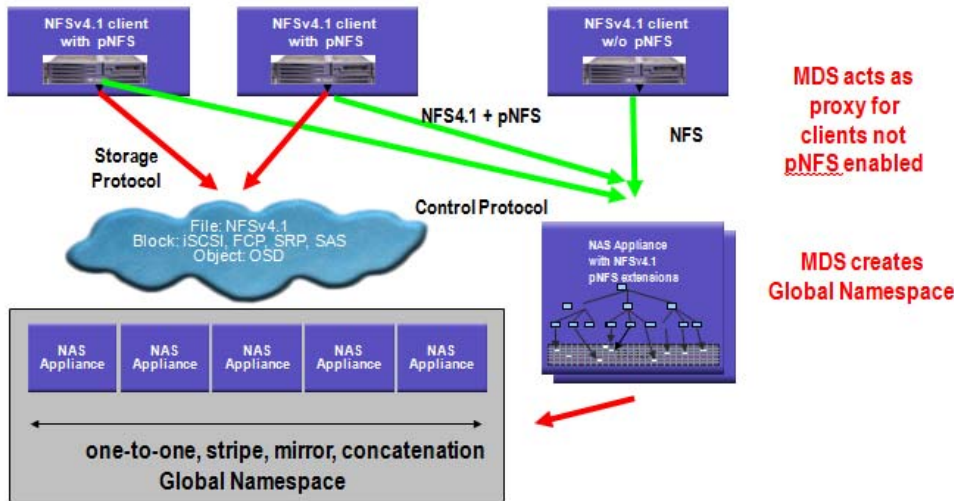
Figure 21: NFSv4.1. pNFS

Removing the NAS server from the data path transforms it into an out-of-band solution; pNFS-aware clients resemble the clients of a SAN file system. To access a file they start with the pNFS metadata server, requesting the file's layout. The pNFS server returns a map that describes where the file physically resides and whether it is striped or not. The red arrows represent pNFS clients directly communicating with the physical devices that hold the file data.



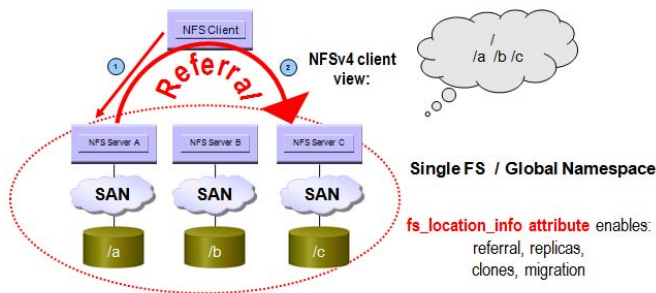
The pNFS protocol supports access to file data, block data and data objects. (Objects are described later).

Figure 22: Data Access Protocols for pNFS



The pNFS metadata server is a single point of failure, so it must be redundant, typically in an active-passive configuration. A pNFS metadata server may manage many file servers or NAS devices; these are represented by disk icons in the diagram (Figure 23). A file may be concatenated or striped across some or all of the physical devices. The MDS can also become a proxy for non-pNFS enabled clients.

Figure 23: pNFS Metadata Server in Redundant Configuration



The third means of creating a global namespace with NFS v4.1 is *referral technology*. In this variation, a client wishing to access file C (in the diagram), can connect to any file server in the environment. For example, the client connects to NFS server A, which is aware that it doesn't host the requested file, so it returns an error code to the client along with a link (`fs_location_info` attribute) to the server that does host the file. Then the client connects to server C (in the diagram) to access the requested file.

Figure 24: NFSv4.1 Referral Technology

This referral mechanism effectively creates a global namespace (aka multi-server namespace) that can be extended further by combining it with pNFS technology (Figure 25).

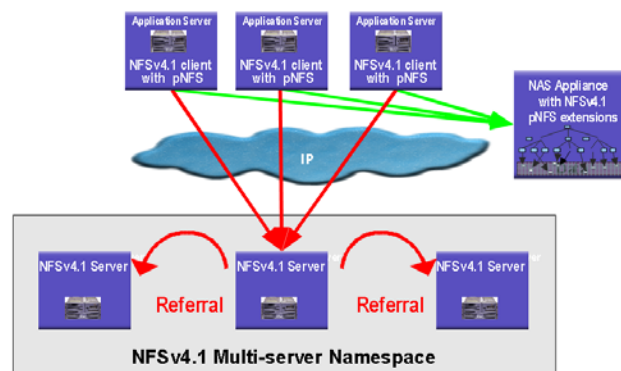


Figure 25: pNFS and Referrals

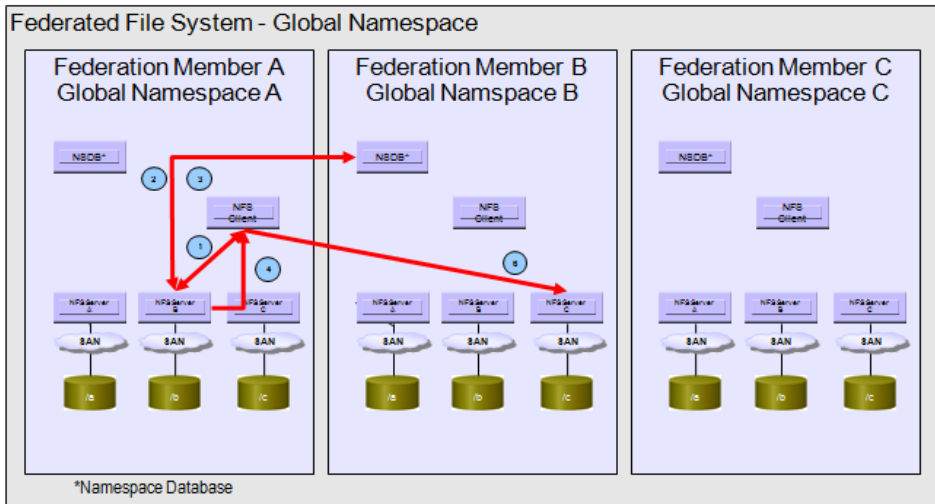


Figure 26: Federated File System: Global Name Space

Many enterprise IT environments have separate global namespaces. Ultimately, some kind of super-global-namespace that combines these global namespaces is desirable. An IETF technical working group is attacking this problem with technology called the Federated File System. A federated file system combines global namespaces that consist of collections of NAS servers. For example, the client on the left hand side of the diagram

(Figure 26) represents a member of global namespace A accessing a file that is not part of its global namespace. It connects to any file server in its global namespace. The file server is aware that the requested file is not part of its global namespace, so it refers to an application called the *namespace database* (NSDB) to determine the file's location. The namespace database returns a referral, which the client can use to connect to the correct server in a different global namespace.

Clustered NAS

Last, but not least, in the taxonomy of distributed file systems is the NAS cluster or NAS grid. This diagram (Figure 27) shows an older architecture, developed to circumvent the single point of failure represented by the NAS head. Most NAS systems that host mission critical data are actually clustered. Normally these NAS heads run in asymmetric active-active mode. Each NAS head actively serves files from a distinct set of LUNs. Thus both NAS heads can serve different LUNs simultaneously, but they are also a failover mechanism for each other.

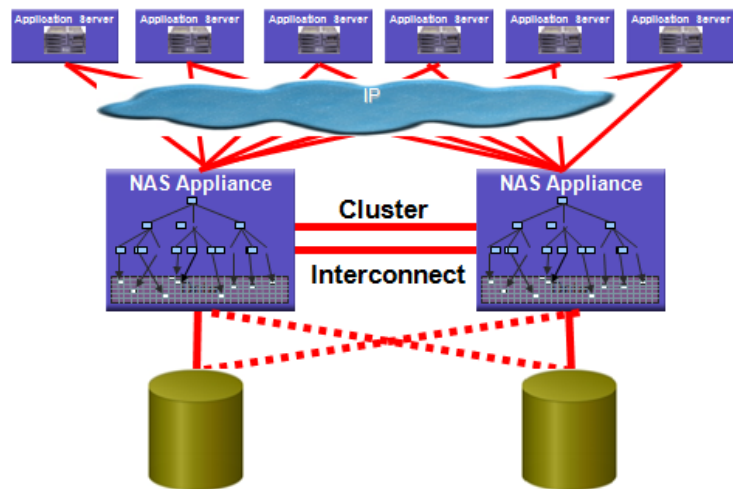


Figure 27: Clustered NAS



NAS Grid/NAS Cluster

There are two types of NAS grid or NAS cluster. This diagram (Figure 28) represents the first, using one single virtual IP address to access two or more NAS heads as one NAS grid or cluster. The client access protocol might be NFS, CIFS, WebDAV, ftp, and so on.

What's important here is that all nodes (possibly dozens) can serve any request, because all share the same storage (i.e. non-captive storage). Examining this architecture in more detail reveals an interesting aspect: the file servers are file servers in terms of the network file system protocol because they serve client requests, but at the same time they are clients to a SAN or a cluster file system. Whenever a client accesses any of these file servers, the file server first queries a metadata server (on the right hand side of the diagram) to determine the location of the requested file. The file is retrieved from physical storage and delivered to the client. In this architecture, each file server is capable of serving any request.

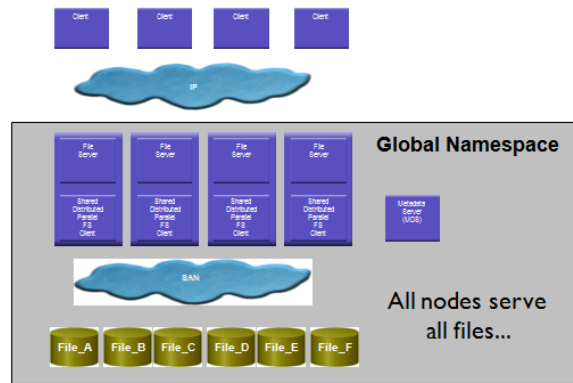


Figure 28: NAS Cluster with Shared/Non-Captive Storage

Whenever a client accesses any of these file servers, the file server first queries a metadata server (on the right hand side of the diagram) to determine the location of the requested file. The file is retrieved from physical storage and delivered to the client. In this architecture, each file server is capable of serving any request.

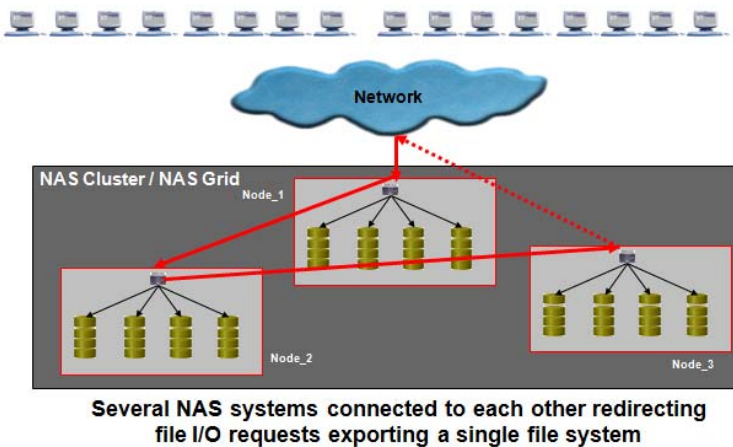


Figure 29: NAS Cluster with Dedicated/Captive Storage

In this architecture, which is reminiscent of the NFS referral mechanism, a client can connect to any head in the NAS grid. Any NAS head which does not host the requested file relays the request to its peer in the grid. If this NAS head also does not host the file, it refers it to a third one and so on. The file data is either transferred to the NAS head the clients has originally connected to or the client has to establish a new connection to the NAS head which hosts the requested data. Eventually, the client will access the file it requested no matter where it entered the NAS grid.

In the second type of NAS grid or NAS cluster (Figure 29), each NAS head includes dedicated storage (i.e. captive storage). Storage and files are not shared; each head can only serve requests for the files or file segments it hosts.

In this architecture, which is reminiscent of the NFS referral mechanism, a client can connect to any head in the NAS grid. Any NAS head which does not host the requested file relays the request to its peer in the grid. If this NAS head also does not host the file, it refers it to a third one and so on. The file data is either transferred to the NAS head the clients has originally connected to or the client has to establish a new connection to the NAS head which hosts the requested data. Eventually, the client will access the file it requested no matter where it entered the NAS grid.



Storage Cloud

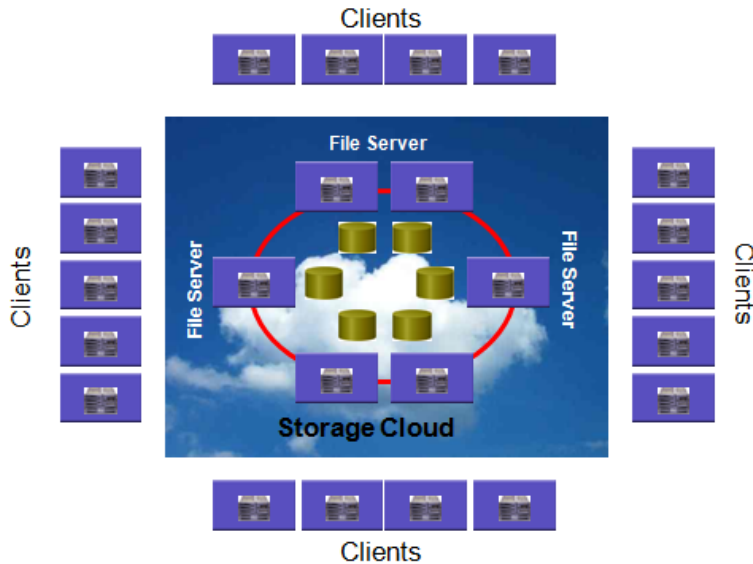


Figure 30: The Storage Cloud

This diagram (Figure 30) shows how a storage cloud, or Storage as a Service (SaaS) basically works. In the center (the inner circle) is the NAS grid of physical storage that holds persistent data. The clients, which are geographically dispersed can access this cloud from anywhere in the world, and data will be delivered to them, wherever it resides.

In this type of cloud, some kind of locality technology is desirable. For example, if a client in New York accesses data in Frankfurt, it would be useful to either move the data to the client or the client application to the data to minimize latency. Locality can also be accomplished by creating replicas of the file. The latter approach requires additional techniques in order to guarantee data coherency.

File System Futures

	Dynamic Data	Fixed Data
Unstructured	Media production, eCAD, mCAD, Office docs	Media archive, DAM, Broadcast, medical imaging, Media-Internet
Structured	Transactional systems, ERP, CRM	BI, data warehousing, scientific, transaction archive

Figure 31: The Data Matrix



Basically, data can be categorized along two orthogonal dimensions (Figure 31). It can either be dynamic or fixed content and it can be either structured or unstructured. An OLTP database is a typical application to manage structured dynamic data. Whereas structured fixed content data lives in specialized databases for reference-only data (i.e. data warehouse). File systems manage unstructured dynamic data with arbitrary records. Unstructured data with reference-only data can be hosted by archives (i.e. so-called Content Addressable Storage – CAS).

In addition to these four traditional data categories a new category gains ground: semi-structured data. It is by nature unstructured data which can be dynamic or fixed and it is managed by a database which is embedded in a file system providing database-like capabilities (e.g. indexing, virtual views etc.).

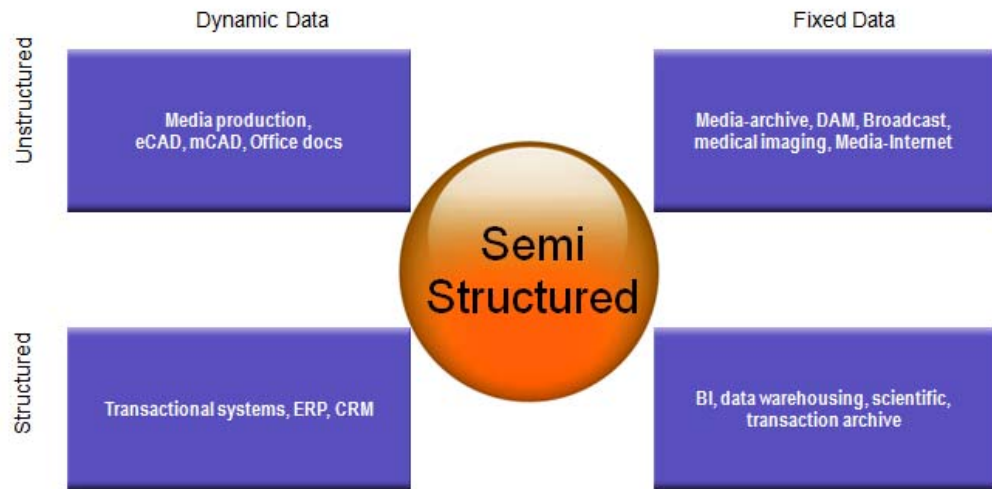


Figure 32: Semi-Structured Data

A lot of exciting file system research is occurring. File systems of the future may resemble databases more than the file systems of today. One major change that is occurring has to do with the structure of a file. The structure is very complicated, very sophisticated. Basically, today, files consist of datablocks and metadata. The industry is enhancing these structures with more metadata and semantic behavior (Figure 33). Common metadata such as owner, last modification date, size of the file, type of the file, permissions and so on, are being enhanced by user and application defined attributes, much like the columns in a database schema. It is also possible to attach policies and methods to files, just as the object oriented programming (e.g. C++ and Java) paradigm attaches processing methods to data.

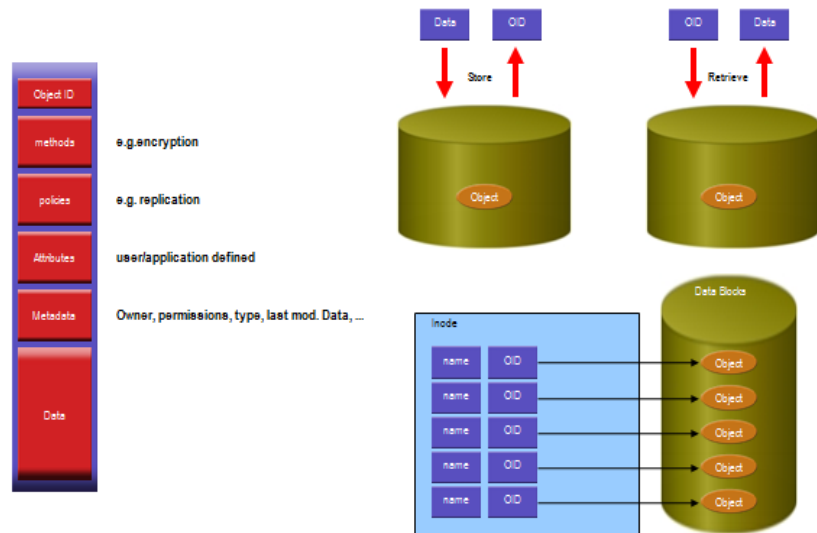
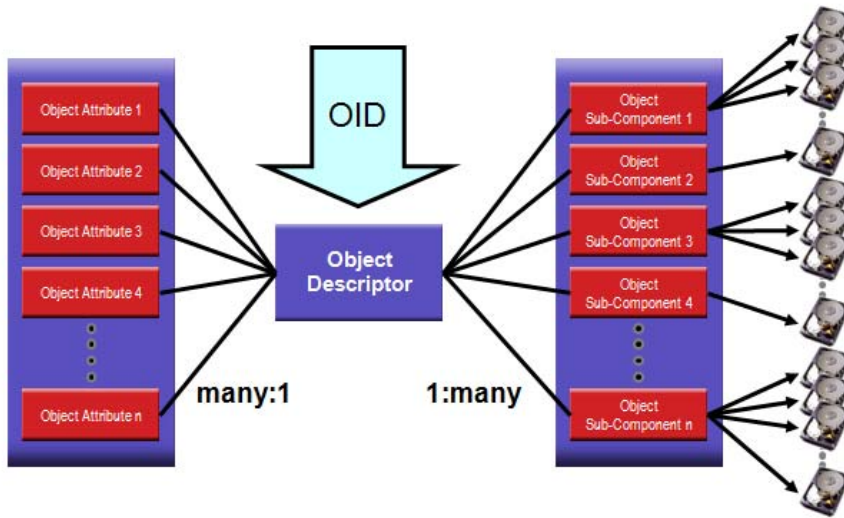


Figure 33: File System Metadata



Something similar is happening with file systems. Methods and policies, for example encryption, are attached to files or groups of files, making them much more than files in the traditional sense. Files are morphing into objects.



Objects in conjunction with object-aware devices have several unique advantages. Creating an object (i.e. storing data in an object-aware device) creates a unique object id. To retrieve the object (container), one supplies the unique object id to the object-aware device instead of complex sequences of logical block numbers. In consequence, the file system becomes much leaner. Space management and block mapping disappear; the file system just maps human readable names to object ids. Object management — block allocation and space management — is done by the object-aware devices, not by the overarching file system.

Figure 34: Object ID in an Object-Aware File System

Objects are identified by a unique OID (Object ID) which points to an object descriptor (Figure 34) which references the related metadata and the subcomponents of an object. The subcomponents of an object can be distributed across several physical devices in order to increase availability or decrease access latency. Meta data can be referenced by multiple objects and the same object can be referenced by multiple sets of metadata.

If you think about these file system objects as the rows in a database schema of a relational database (Figure 35), then you should get an idea of a file system managing semi-structured data.



Figure 35: Managing File Objects



Indexes can be built for OIDs, attributes, methods etc. (Figure 36) enabling search and join operations similar to what we can achieve with an SQL-like language in conjunction with a relational database (e.g. object search, content search, join, virtual view, constraints, cursors, etc. ...).

Figure 36: File Object Indexes

Actually this idea is not completely new. There are a lot of applications out there managing unstructured data in an applications specific database (e.g. media players, email applications, ...). These application specific content repositories provide a combination of application, database, data services, data and sometimes pointers into an external file system.

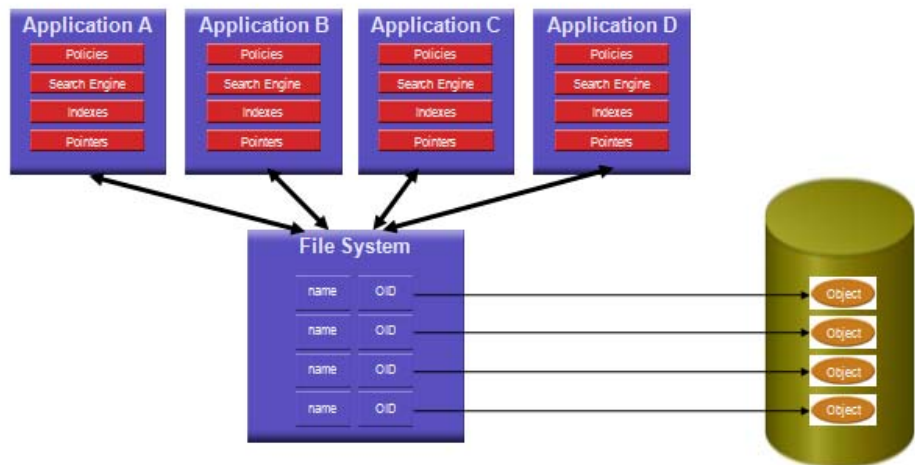
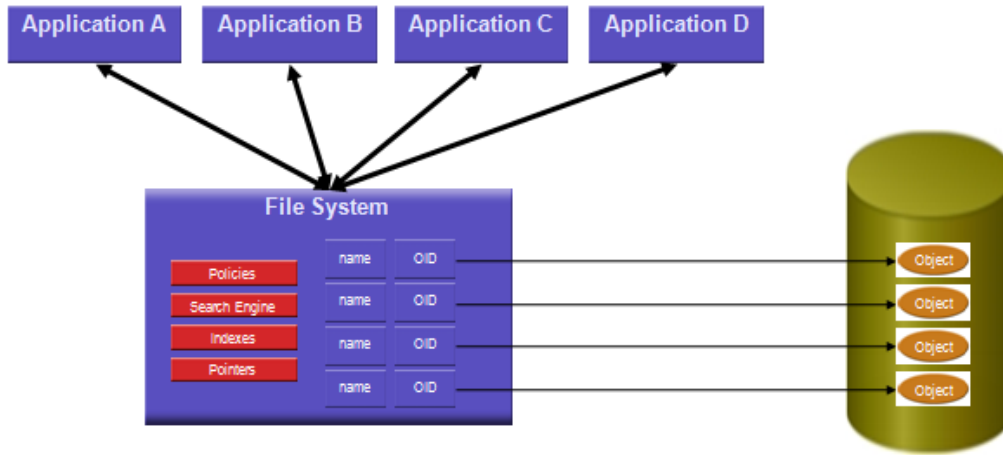


Figure 37: Application Specific Content Repositories



Providing these capabilities in a general purpose file system makes them shareable by many different applications. These files-based content repositories (Figure 38) represent a set of application agnostic data services for unstructured data.

Figure 38: File-Based Content Repositories

This, then, is the storage cloud of the future, based on scale-out storage technology. It may be accessed as files; it might also be accessed as block devices. Data stored in future clouds will become more object-like. Today's directory hierarchy may disappear, replaced by a flat namespace in the cloud managed by some kind of associated database. Along with unstructured data stored in traditional file systems and structured data stored in databases we will see another class of data called semi-structured. Semi-structured data perfectly fits into file systems which combine traditional file operations, semantic-rich objects and database technologies.

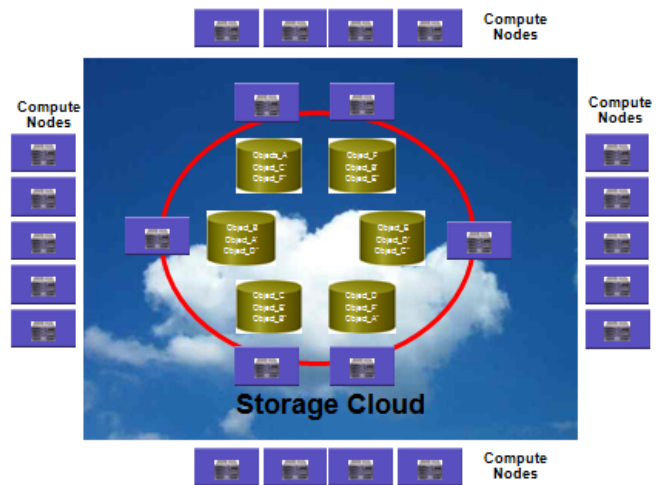


Figure 39: Storage Clouds of the Future



Conclusion

We come to the end of this SNIA Tutorial. I've touched a little bit about objects, but I don't want to give you the impression that objects will rule the world and will substitute files and blocks. At the end, everything is saved and stored as blocks on tape, magnetic disk, optical disk or flash memory. Files are made up of blocks which, by themselves are based on a very simple model. Blocks just represent bits and bytes. The file systems metadata structures are responsible to provide semantics between blocks. The file abstraction introduces metadata and more elaborate semantic behavior.

Extending and enriching files with increasing amounts of metadata and information, so that they can be more autonomous, eventually morphs them into objects. But not everything fits into objects. Thus in the future traditional file systems and file systems managing objects will co-exist and complement each other.

Application may interface with the storage subsystem in anyone of three layers:

- **Block** with highest performance and very little meta data
- **File** with medium performance and some meta data
- **Object** with medium performance and *rich* meta data

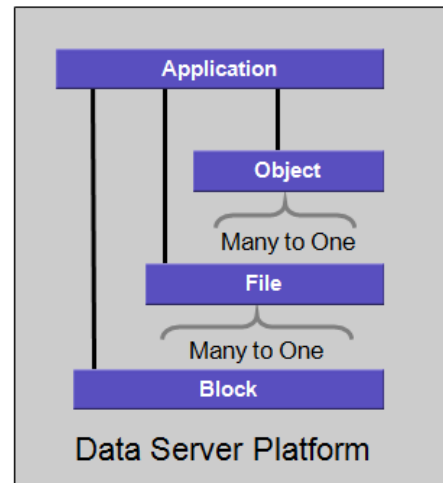


Figure 40: File Systems and Objects Complement Each Other



About the Author

About the SNIA

The Storage Networking Industry Association (SNIA) is a not-for-profit global organization, made up of some 400 member companies and 7,000 individuals spanning virtually the entire storage industry. SNIA's mission is to lead the storage industry worldwide in developing and promoting standards, technologies, and educational services to empower organizations in the management of information. To this end, the SNIA is uniquely committed to delivering standards, education, and services that will propel open storage networking solutions into the broader market. For additional information, visit the SNIA web site at www.snia.org.