NFS and DFS
A Functional Comparison

Saqib Jang

Technical Report 15
April 1997

Abstract

NFS is the de facto standard for file sharing in heterogeneous enterprise networks. Independent market research forecasts show strong growth in the NFS multivendor-installed base from 8.5 million nodes in 1994 to 12 million nodes in 1997. Although NFS has been shipping since the mid-1980s, its continued evolution to support evolving customer requirements has been key to it's continued widespread use.

Specifically, the recent growth in NFS usage is due to the evolution in its capabilities in the areas of performance, security, administration, global support, availability, and heterogeneity. These make NFS optimal as a foundation for sharing mission-critical information within the enterprise.

DFS is the file sharing component of the Open Software Foundation's Distributed Computing Environment (OSF DCE). Implementations of DFS started shipping in 1992-1993 for HP-UX, AIX, and Solaris environments. DFS was designed in the early 1990's to support commercial-level features such as global naming support, security, availability, and administration that were not available in NFS at that time. However, NFS has evolved since that time to address these requirements. This, coupled with NFS's strength as a multivendor standard, makes it the file sharing solution of choice for enterprise environments.

This document contains a comparison of the features and benefits provided by the latest client and server-side advances in NFS and DFS technologies. Where pertinent, a description of the implementation of these features is provided. Unless specifically noted, Auspex is committed to supporting all new *server-side* NFS features discussed in this document (new NFS *client-side* capabilities will typically transparently interoperate with the installed base of Auspex and other NFS servers).

Document # 300-TC043

Auspex Systems Inc.
5200 Great America Parkway,
Santa Clara, California 95054 USA
Phone: 408/980-2000 · Fax: 408/986-2020
http://www.auspex.com

Table of Contents

1 Introduction

1.1 NFS Overview

NFS is a file access protocol that allows transparent access to information on heterogeneous enterprise networks. Specifically, NFS is one among a set of services comprising the ONC+/NFS platform for information-sharing and administration of a client-server environment. ONC+/NFS Include NIS/NIS+ naming/information services, RPC/XDR distributed application development platform, security services, and client-side caching. ONC+/NFS is supported on all key operating system environments, ranging from PCs to workstations to mainframes. NFS is a client/server application built atop ONC+ RPC/XDR layer and uses other ONC+/NFS services. For simplicity reasons, "NFS" is used in the rest of this paper to refer to the ONC+/NFS collection of services that together allow NFS-based high-performance file sharing on today's enterprise distributed networks.

Specifically, NFS has the following features and benefits:

- NFS clients transparently access remote files as if they were local. Transparent access to remote files is provided through an automatic mounting facility.
- NFS includes features, such as memory-based caching, to provide equivalent performance for remote and local file access.
- NFS file operations are generic enough to be easily implemented atop all major operating systems and file systems, allowing file sharing in a heterogeneous environment.
- NFS clients and servers recover quickly and easily from machine and network failures.
- NFS security (based on ONC+/NFS RPC) ensures files are protected from unauthorized intruders.
- Centralized NFS administration (based on ONC+/NFS NIS or NIS+) improves productivity for administration tasks such as relocating or renaming files.

NFS has continued to evolve through a multivendor effort to meet evolving file sharing requirements for commercial enterprise networks. Some of the most important new features provided by NFS include:

- A new revision to the NFS protocol, NFS Version 3, that increases scalability and performance. Release 1.9 or later of Auspex's System Software delivers a high-performance, optimized implementation of NFS Version 3.
- Support of small workgroups to large enterprise networks.
- Highly competitive performance enabling fast access to remote information and optional local disk caching.
- X/Open Federated Naming for intuitive naming of files across the enterprise.
- A flexible security architecture allowing support of multiple network security mechanisms.
- Auspex's DataGuard and ServerGuard value-added products allowing NFS to be fully tolerant of system and network failures.

1.2 DCE DFS Overview

DFS file sharing service is layered on top of the OSF Distributed Computing Environment (DCE) and is an integral part of DCE. DCE fundamental services—RPC, naming, security services, time—provide value-added capabilities to DFS in the areas of administration, security and availability. Deployment of the DCE set of services is required for DFS. For simplicity reasons, "DFS" is used in the rest of the paper to refer to the full collection of DCE services that support DFS-based file sharing.

DFS is made up of two components:

- Every machine—client or server—sharing files within DFS must run the DFS Base Service.
- DFS file server machines run the Enhanced File Service (EFS). EFS implements important DFS services including backup, replication, server monitoring and server re-start.

DFS provides the following capabilities over and above basic network filing functions:

- Files are cached locally either on a local disk or in memory on client machines.
- DFS optionally replicates the files used most often on multiple file servers.
- All information supported by DFS servers becomes part of one, unique file system. Global file naming allows a file to be accessed with a single file name regardless of the file's physical location in the system.
- Authentication and access control mechanisms for security.

This document contains a comparison of the features and benefits provided by the latest client and server-side advances in NFS and DFS technologies. Where pertinent, a description of the implementation of these features is provided. The specific goal behind this discussion is to help system administrators and network architects in making an informed decision about the appropriate networking file services to deploy for their environments.

Unless specifically noted, Auspex is committed to supporting all new *server-side* NFS features discussed in this document (new NFS *client-side* capabilities will typically transparently interoperate with the installed base of Auspex and other NFS servers). Please contact your Auspex representative for specific information regarding specific feature availability.

2 Caching

Client caching mechanisms are commonly employed in distributed file systems to speed up remote file access, increase server scalability and reduce network traffic. Once the client cache is populated with file data (such as pages, attributes, and directory information) the majority of file read/write operations occur locally on the client, thereby avoiding network traffic and server resource utilization. File data is typically cached in units called *chunksizes*. When an application issues a *read* request for file data, the minimum amount of data that will be fetched from the file server is a *chunk*. For large sequential file reads, a larger chunksize generally yields better performance by reducing the overhead associated with the underlying network communications protocols.

Network file systems may use a *read-ahead* mechanism that enhance sequential read performance. When the mechanism detects sequentiality, the client may issue multiple simultaneous requests for chunks. The expectation is that by the time the application issues a read operation on a piece of the file, the data may already be in the client cache or in transit from the server. Once data in the cache has been modified, these changes must eventually be propagated to the server. This is accomplished through the use of *delayed-write* policies.

The client cache can be configured in memory or on disk; either choice is useful in lessening the amount of server traffic. However, using a memory cache typically yields better performance in applications where very large files are used or where cached data are rarely reused. On the other hand, a large disk cache may be more appropriate in environments with heavily loaded networks, with clients that have small system memories, and/or where file data are heavily reused. A disk cache allows the additional benefit of being *persistent* (that is, information in a disk cache will survive a reboot).

2.1 Caching in NFS

NFS supports both memory and disk-based client-side caching for improved performance and server scalability. All NFS clients use the client's physical memory to cache file pages, file attributes, and directory entries. Through page clustering, multiple file pages are combined into chunks of 8 KB. Since the size of the cache is limited by physical memory, the degree of performance improvement gained by memory-based caching depends on the amount of free memory.

NFS uses a read-ahead strategy to improve its sequential read performance. The NFS client processes, called *biods*, pre-fetch file data by issuing RPC read requests to the server on behalf of the application process that issued the *read* system call. Each *biod* can request only a chunksize amount of data at one time. The default (and maximum) chunksize is 8 KB for both read and write buffers for memory-based caching. However, the buffer sizes may be changed by using the *rsize* and *wsize* parameters of the *mount* command. NFS implements delayed-write policies for forwarding modifications to the client cache to the server. NFS file data changes are forwarded to the server by the *biods*:

- If the memory-based caching system performs page replacement and the write is scheduled.
- If the *write-behind*[1] mechanism schedules the write.
- Periodically (every 60 seconds is the default) by the *syncd* daemon.
- Upon a file close or *fsync* operation.

2.1.1 Disk-based Caching—CacheFS

The Cache File System or CacheFS is a new client-side capability that allows local disk caching for NFS[2]. Both NFS V2 and V3 clients still cache as much data in physical memory as is available. CacheFS

[1] Discussed in the Cache Consistency Section (3.0)

significantly increases the cache space available to NFS by using the client's disk. Because of the greater storage available on disk, data can be cached in 64 KB chunks, and whole directories can be cached instead of just directory entries. Expanded cache space means clients can access more data quickly and they make fewer requests of the server.

CacheFS provides dramatic benefits for performance for remote file access. The effect of CacheFS is very noticeable as clients attempt to access data from across the local or wide-area network. A cache hit to local storage can return data 10 to 100 times faster than across the WAN network.

2.2 Caching in DFS

As with NFS, the DFS client cache can be configured in memory or on disk. The DCE/DFS equivalent to CacheFS is the Cache Manager. The DFS Cache Manager retrieves and caches files in 64 KB chunks on a local disk and is managed similarly. DFS also uses a read-ahead mechanism that enhances its sequential read performance. In addition, DFS implements delayed-write policies for transferring modifications in the client cache to the server. DFS file data modifications can be propagated to the server in the following ways:

- On a periodic basis (every 30 seconds) by one of the DFS client processes.
- If the server revokes the client's write token[1].
- If the DFS client cache becomes full.
- Upon a file close or *fsync* operation.

2.3 Comparison Summary

NFS and DFS both provide equivalent functionality for client-side caching. Both file services support memory and disk-based caching and support a maximum of 64 KB cache chunksize. Both allow read-ahead and delayed-write mechanisms with flexible setting of policies.

---

[2] CacheFS is a client-side NFS feature support by on platforms including systems from SGI, Sun, HP, and PCS

[1] Tokens are discussed in the Cache Consistency section.

3 Cache Consistency

As discussed in the previous section, client-side caching is key to allowing improved user response time and server capacity for distributed file systems. Cache consistency refers to mechanisms used to determine the validity of cached data and the propagation of cache modifications to the server. Cache consistency is important in ensuring that data is safely written to the server while at the same time minimizing the amount of time an application waits for its data to be written to stable storage.

Cache consistency mechanisms cannot be thought of as substitutes for strong transactional consistency to protect file data in the face of concurrent writes by multiple clients. Applications that concurrently write to files typically use file locking to protect transactional consistency for remote file access. This option is provided for network file system clients and servers through standard Unix advisory locking and similar locking mechanisms on other operating environments. In addition, applications, such as network-based ECAD, MCAD, and software development packages that require strong file or directory consistency typically incorporate their own locking and synchronization techniques for remote file access.

3.1 Cache Consistency in NFS

The basis of the NFS cache consistency model is the time-limiting of the validity of cached data by clients. This time varies between 3 and 60 seconds, depending on the frequency of update. A file that is updated frequently will have a short cache time-out (nearer 3 seconds), while a file that is updated infrequently will have a longer cache time-out (nearer 60 seconds). These limits are configurable on a per-mount basis, so that cache consistency can be improved by adjusting the lower limit to values smaller than 3 seconds—or even zero. This exacts a trade-off in cache effectiveness, short time-outs increase the rate of consistency checks with the server. Similar cache consistency is maintained with directory entries, though with limits between 30 and 60 seconds.

NFS implementations guarantee *close-to-open* consistency. All file modifications are written to the server when a file is closed. When an application opens a file, it invalidates cached data if it detects that the file has changed on the server. This close-to-open consistency guarantees that after a file is closed its contents will be visible instantly to applications elsewhere on the network that open the file.

The NFS close-to-open cache consistency mechanism is a light weight, efficient means for ensuring validity of cached data. Frequently, mention is made of a drawback in this mechanism: that it does not guarantee data validity in the face of write conflicts (where two or more clients attempt to open a file for write access concurrently). However, it is a misconception that cache consistency mechanisms in network file systems such as NFS are designed to solve the problem of concurrent file access. Specifically,

- Write conflicts are a very infrequent occurrence in typical network file access patterns [Welch91].
- Use of NFS file locking technology, called NFS *Lock Manager*, is required where fine grained control over concurrent file access is desired.

3.1.1 Cache Consistency and Write Throughput

The NFS V2 client stores the data in the local cache as the first step of a *write* request. After the data is written to the cache, the application that issued the request continues with its operation. In parallel, the client takes the written data and submits corresponding *write* requests to the server. The NFS server then writes the data serially to stable storage and responds to the client. Because each individual NFS *write* request from the client requires that data must be written to stable storage on the server before it completes, this mechanism is known as *synchronous writes*.

The NFS V3 protocol [Pawlowski94] dramatically increases write throughput by eliminating the synchronous write requirement while retaining the benefits of close-to-open semantics. Specifically, the NFS V3 client retains a copy of written data which has not been the subject of a COMMIT call (and will retransmit this in the event of a server failure). From time to time, and always at file close time, the client sends a COMMIT request for outstanding data. The server must ensure that all such data is on stable storage before responding to the COMMIT request. However, since the server starts asynchronous writes of data as soon as it is received, there will normally be very little data which is not already on disk at COMMIT time. Thus, the overall throughput approximates that of fully asynchronous write behavior. This feature, referred to as *safe asynchronous writes*, vastly reduces the number of write requests to the server, and thus significantly improves write throughput.

3.2 DFS Cache Consistency

DCE/DFS maintains cache consistency with a server-based token scheme. When accessing a file, a client is required to hold a *token* that reflects the type of file access permitted to the client. To read a file a read token is required from the server. To write to a file the client must request a write token from the server. If a client requests a write token for a file that has several clients reading from it, the server will send a message to the reading clients to invalidate cached data from this file. This is to ensure that they see the most recent data in the file. In the DFS token-based scheme, close-to-open semantics need not be enforced because the outstanding data can be written to the server after the application closes the file. The server will notify the client to yield its token whenever another process wishes to access this file.

DFS is similar to NFS V3 in its support of safe asynchronous writes. In the case of DFS, file data may still be in the server's memory and not yet in stable storage when an application's file close call returns. DFS does, however, commit data to non-volatile storage prior to returning from an *fsync* call.

DFS cache consistency model may have problems when a failure occurs during the time a client process possesses a token. For example, if a client holding a file's token crashes or if a network partition occurs, the server is required to handle complex exception cases (for example, if a new client wishes to access the same file). Problems can also be caused when the first client comes back and has to resolve changes made to the file during its absence. A labor intensive approach is to have the system administrator decide how to handle tokens on a case by case basis.

3.3 Comparison Summary

In summary, it is easy to see that DFS token passing can be difficult to administer and complicated to implement. The asynchronous write feature of NFS V3 provides the same performance and reliability benefits without the administrative overhead.

Another key point to note regarding the DFS token-passing cache consistency model is that it alone cannot solve the problem of concurrent file access by multiple client processes. As with NFS, DFS requires a locking mechanism to enable synchronized write access by multiple clients to a single file.

4 Administration

Distributed computing infrastructure components must enable simplified administration of today's dynamic enterprise network environments. Thus, the ability to manage change must be a key requirement for selecting distributed file systems. Network file services should enable streamlined implementation of tasks such as adding, deleting, relocating, and changing file system configuration information. Finally, it should be possible to administer the file system centrally, thereby minimizing time and effort required to keep the distributed file services up and running.

4.1 NFS Automounter

NFS includes a capability called the *automounter*[1] that provides two key administrative benefits: it gives users, workgroups and applications location-independent access to server file systems and it allows a means for accomplishing administrative operations, such as adding or relocating file systems, in a centralized fashion.

Automounter allows associations between client pathnames and server file systems. The automounter converts these associations or mappings into actual file system mounts and adds them to the client's file tree. In this way, the mappings define the NFS *namespace* or the client file names that are used to access server file systems.

The NFS namespace can be modified to suit the specific needs of different environments. Administrators can establish their own naming policies for simplified access to remote file systems. For example, it is common to set up all users home directories under */home/username* or all shared manufacturing files under */mfg/shared*.

The NFS namespace is also a *global* or shared namespace. This means that the client pathnames are valid throughout the network. Users can move to different locations within the network and are still able to access files and directories using the same pathnames they used when they were on their home system.

The global NFS namespace is configured by editing automounter maps. The fact that maps are stored in the name service (eg. NIS or NIS+) enables the namespace to be administered centrally—no "per client" administration is necessary. For instance, the */home* directory is associated with a map, *auto_home*, that describes the location of every user's home directory, for example,

```
john echo:/export/home/john

jim monterey:/export/home/jim
```

The automounter makes paths like */home/john* work across the organization even if the specific directory is moved to a different server. This allows increased mobility allowing, for example, the user John to log into any workstation and find that the current directory is his home directory. This path is valid even on the server machine *echo* since the actual location of the home directory is private to the automounter map entry.

The automounter also enables efficient *read-only replica* location. Any map entry may describe several locations for a read-only file system. When mounting the file system, the automounter will choose the server that has the nearest network proximity thereby improving response time and reducing network traffic.

---

[1] Automounter is a NFS client-side capability and is supported in NFS implementations for a range of operating environments including HP-UX, AIX, SunOS/Solaris, DOS, Windows 3.11 and Windows NT.

4.2 DFS Namespace

The DFS file namespace of an organization consists of a *single* directory tree spread over any number of DFS file servers. DFS does not require you to know which machine holds your files and directories. To access a DFS file, you need to know the DFS pathname of the file.

A DFS pathname has three components: the cell name, the DFS junction, and the location of the DFS file in the DFS namespace. For example, suppose a user named Judy is from the cell or organization *abc.com*. A directory to which Judy can connect is her home directory: */.../abc.com/fs/usr/judy*. In the pathname, */.../abc.com* indicates the cell name, */fs* indicates the file namespace designation in the *abc.com* cell, and */usr/judy* indicates the user's home directory. If the directory */other* resides in judy's home directory, she can access the directory with the pathname */.../abc.com/fs/usr/judy/other*.

DFS does not require per-client administration as file naming and location information are stored in the DCE Cell Directory Service (CDS). DFS also allows users and applications to access files in a location-independent fashion using the file name-to-location mapping information stored in CDS.

4.3 File Naming Using X/Open Federated Naming (XFN)

X/Open Federated Naming (XFN) [XFN95] is a powerful facility that allows network applications to consistently and easily access the diverse range of network entities such as files, printers, and spreadsheets, across the global network. XFN is based on *X/Open Preliminary Specification for Federated Naming, (July 1994)*. The specification has been endorsed by major vendors such as IBM, Hewlett-Packard, SunSoft, DEC, Siemens-Nixdorf, and Open Software Foundation (OSF) as the standard means for accessing network resources such as files, printers, and spreadsheets, across enterprise networks. XFN support for NFS is shipped by Sun as a component of Solaris 2.5. SunSoft is also licensing XFN portable source for NFS to ONC+/NFS vendors. OSF plans to ship XFN support for DFS to its technology licensees as a component of DCE 1.2 in early '97.

XFN-based file naming uses XFN *naming policies* to provide users with a consistent, intuitive means of accessing file systems across the enterprise. Files may be named relative to users, organizations, hosts, and sites. XFN-based file naming gives users a consistent view of file systems across the enterprise or global network. Existing applications can access file systems supported by XFN, such as NFS or DFS, just as they would any other file system and are not required to use the XFN API. Further, no changes on installed base of file servers is required in order for clients to use XFN-enabled file naming policies to access such servers.

XFN policies enable network entities such as users, hosts, organizations, services, and files to be named in a logical and hierarchical manner. For instance, a user may be named relative to the organization he is in, and a directory may then be named relative to that user. As an example, the project directory of the user Steve in the software division of an enterprise might be accessed by: *% cd /xfn/org/software.eng/user/steve/fs/project*

The notations "*org*", "*user*", "*fs*" in this example are references to the organization or cell, user, and file system namespaces respectively. The organization name "*software.eng*" is a specific cell or domain name.

Files from other organizations within the namespace hierarchy may be accessed in a similar fashion. In addition, users in the same organization as Steve have the option of using a shorthand notation that looks like the following: *% cd /xfn/user/steve/fs/project*

XFN also allows files to be associated with organizations, not just users or hosts. These files can be shared across an organization.

4.4 Comparison Summary

In summary, the naming/location models employed by the two filing services allow NFS to provide key value-added over DFS in the file system administration area. Specifically,

- Both file services allow centralized file system administration and location-independent file server access through employing a global name space.
- NFS global namespace is configurable allowing administrators to setup naming policies to suit the needs of their organization. DFS global namespace is fixed.
- XFN is the next-generation file naming/location capability for both NFS and DFS. However, XFN support in technology and product versions of NFS has been available since 1995 while XFN support for DFS will be available in 1998 at the earliest.

5 Availability

Continuous accessibility of files to users and applications on the network has to be a key requirement for distributed file systems. This is especially true in mission-critical environments where any interruption of service can quickly translate into lost business. There are a number of techniques through which distributed file systems may provide improved availability to users and clients. These include support of *replicated* or redundant servers for continuos access to file server data in the face of server or network failures and *highly available data service* in the face of server hardware or software component failures.

Replication functionality in network file systems is a client- and server-side capability. It can be supported for either read-only or read-write data. Read-only replication refers to support for redundant servers containing program binaries or data that does not change very frequently. Read-write replication refers to replicated servers supporting data that may be read as well as modified by authorized clients. In addition to improving data availability, replicated servers provide a number of key administrative benefits. These include distributing client load over a number of servers and keeping traffic off of network backbones by placing servers near clients. Network file systems may allow clients to failover to a read-only replicated file server at either mount time or whenever a network or server failure occurs.

Highly available data service is a server-side capability that provides a much higher level of fault tolerance compared to replicated read-only servers. The goal behind this functionality is to provide *continuous read-write* access to file server data in the face of failure of any of the software or hardware components of the server. It typically requires duplication of both hardware and software so that when one component fails, another can take over.

5.1 High Availability in NFS

Automounter capability in NFS supports replication of *read-only* file systems. An automounter map entry for a read-only file system may describe several locations for alternative replica servers. At mount time the automounter mounts the file system on the server that has the nearest proximity to the client. This reduces network routing delays, helps keep NFS traffic off the corporate backbone, and improves server scalability.

In addition, there are a number of products from third parties such as Auspex that provide server-side capabilities for highly available NFS data services. Notable among these are DataGuard and ServerGuard high availability software. DataGuard is a software solution that eliminates NFS service downtime that may be caused by UNIX and application instability as well as UNIX reboots. It allows full read/write access to NFS data to continue uninterrupted in face of such events. ServerGuard is a software-based failover solution designed to provide continuous read/write access to NFS data even in the event of a complete system failure. ServerGuard automatically and transparently mirrors file systems on two separate servers on the network. Should either server experience a hardware or software failure, the other server will take over in typically less than 5 seconds allowing for very high availability.

5.2 High Availability in DFS

DFS optionally supports replicated servers. Unlike NFS replication, replication in DFS is supported for both *read-only* and *read-write* data. In addition, DFS allows clients to *dynamically* failover to another replicated server in case of server or network failures whereas the automounter-based replication capability in NFS allows client failover only at *mount time*.

5.3 Comparison Summary

Base-level DFS provides more availability functionality than standard NFS. This includes DFS support for read-write replication and dynamic client failover. Unlike DFS, however, NFS supports value-added solutions from third parties, such as Auspex's ServerGuard and DataGuard software, allowing very high availability for NFS data services in the face of network and server hardware/software failures. Such products combined with NFS's track record of deployment in production environments make it a superior choice as the infrastructure for mission-critical data services.

6 Global Support

Access to information across global networks is a requirement for corporations wishing to compete successfully in the global economy. Global expansion has given rise to workgroups spread across the multiple locations around the world. This has created an environment where file systems must span geographies. In addition, the exponential growth in Internet use is making Internet-based access a key distributed file system requirement. As a result, the definition of a network server has expanded to cover file servers over LANs *and* WANs.

6.1 Global Support in NFS

NFS supports global workgroups by giving users transparent, fast access to server file systems connected to client systems on LANs and WANs. The following capabilities enable efficient global access:

- NFS servers and clients now also support the TCP transport protocol that provides improved performance and reliability for WAN support. Unlike UDP, TCP is a reliable protocol that is designed to provide guaranteed data delivery. Further, all NFS client traffic can be multiplexed over one TCP connection to the server. This keeps overhead to a minimum and allows servers to scale to serve large numbers of clients.
- Automounter keeps files continuously accessible through client-side mount-time failover to read-only file systems on servers worldwide.
- Client side caching gives clients fast access to file data transferred from the remote server and stored locally.
- Auspex's ServerGuard product can allow dynamic failover to read/write file systems on NFS servers for a wide range of hardware and software failures.

6.1.1 WebNFS

WebNFS is an NFS extension designed to make NFS compatible with firewalls and allows Web browsers and other Web clients to directly access NFS file servers anywhere on the Intranet or Internet. WebNFS complements *http* by providing a high performance file access protocol suitable for down-loading static data such as program binaries or images across Intranet/Internet. WebNFS can deliver files over the Internet or corporate intranets up to 10 times faster than *http*. Web sites that use both *http* and WebNFS can support up to three times as many user requests as a site that uses only *http*. WebNFS takes advantage of NFS support for TCP/IP, which provides congestion control and error recovery, for reliability and performance over the Internet. In addition, it implements mechanisms for simplified file access across Internet-based links.

A number of client and server vendors, including Auspex, Netscape, Oracle, Sun, JavaSoft, Spyglass, and Sequent, have announced plans for support of WebNFS. In addition, a specification of WebNFS has been submitted as a draft RFC to IETF.

6.2 Global Support in DFS

DFS provides TCP support and client-side caching for high-performance and reliable WAN-based file sharing. It also provides dynamic client-side failover to replicated WAN-based file servers supporting read-only and read-write file sharing.

6.3 Comparison Summary

Both NFS and DFS contain important functionality for supporting file sharing across global environments. DFS and NFS are equivalent in TCP support and client-side caching for high-performance and reliable file sharing across WAN-based networks. DFS provides additional availability benefits for global environments in allowing support for dynamic client-side failover to replicated

servers for read-write file sharing. However, NFS provides a very high degree of availability for WAN-based environments through deployment of Auspex's ServerGuard product. Finally, NFS is alone in supporting a high-performance file access protocol for Internet/Intranet using WebNFS.

7 Security

Network file systems are inherently vulnerable to unauthorized access due to their distributed nature. There are two specific threats to the security of a distributed file system: compromise of file server security through impersonating a client's identity and sending of forged requests to a file server to access or destroy file data.

7.1 NFS Security

NFS supports two types of security services: *authentication* and *authorization* services. The purpose of the authentication service is to validate user's identity before allowing them to use resources. NFS can be *optionally* configured to use an authenticated protocol, called SecureRPC, to provider authenticated file access. Multiple types of authentication services are supported including one based on the Diffie-Hellman key exchange protocol, another using Kerberos, and the third using a simpler "UNIX" style authentication. NFS is able to utilize multiple authentication "flavors" by virtue of the flexibility of the underlying RPC service. The use of authentication causes a performance overhead and its optional use allows sites to decide if their security requirements justify such a scheme.

Diffie-Hellman and Kerberos authentication schemes use encryption to protect authentication information. Information is protected using an encryption algorithm with a special value or "key." Knowledge of the decryption key is required to convert the data back into readable form.

The purpose of an authorization service is to ensure that users have appropriate permissions to perform specific operations, such as reading or modifying files, once they have been authenticated. NFS supports UNIX style authorization (or "permission checking"). UNIX style permissions involve using the file *permission bits* functionality that is a standard part of UNIX. Permission bits are set to indicate whether read, write and/or execute permission is allowed for the owner of the file, the members of certain groups or for the world.

7.2 DFS Security

DFS supports only Kerberos authentication. All communication between DFS clients and servers takes place using the authenticated RPC mechanism. In addition, every user is required to have a private key stored on a Kerberos ticket granting server.

DCE/DFS provides support for Access Control Lists (ACLs) which expand the amount of file permission information beyond standard UNIX permission checking. Every file has an associated ACL. Clients identify themselves to the ACL manager by providing a list of identification values including their UID and GIDs. The DFS server is responsible for responding to client requests by comparing the client permission values to the file's ACL and either granting or denying access.

7.3 Comparison Summary

The NFS authentication model allows a number of degrees of flexibility not available in the DFS authentication model. These are:

- NFS allows sites to *choose* if they want to deploy authentication services or not in view of their security requirements. DFS, on the other hand, *requires* the use of Kerberos-based authentication. This is important as the use of encryption-based authentication mechanisms like Kerberos have additional network performance overhead.

- NFS allows use of multiple "flavors" of authentication such as Kerberos and Diffie-Hellman whereas the DFS authentication is limited to support of Kerberos. The RPC mechanism underlying NFS has the flexibility to support newer encryption models such as the RSA encryption which is more prevalent on the Internet.

DFS is superior to NFS in the support of ACLs that provide a finer level of file access control than is allowed by UNIX permissions. However, use of ACLs requires additional overhead in setup and on-going administration of ACL information.

8 Performance

High performance is an important criterion for measuring the effectiveness of a distributed file sharing solution. Distributed file sharing performance is intimately tied to the productivity gains made possible through information sharing across enterprise networks.

8.1 NFS Performance

NFS performance has increased an order of magnitude in the last five years, from hundreds to thousands of operations per second. The industry standard LADDIS benchmark developed by SPEC motivates NFS vendors to aggressively tune their latest systems and publish updated benchmark results. In addition, NFS V3 protocol and client-side disk caching (discussed in the Caching section) allow enhanced performance. The features of NFS V3 contributing to improved NFS performance are discussed below.

8.1.1 NFS V3 Performance Improvements

NFS V3 includes a range of performance improvements. The first key improvement is the support of *safe asynchronous writes* (discussed in the section on Caching) which shows about *100%* improvement in write throughput in the Auspex NFS implementation between similarly configured clients and servers upgrading from NFS V2 to NFS V3.

NFS V3 also implements fewer on-the-wire operations between clients and servers for improved performance. NFS V2 clients check to make sure that their cached data does not become invalid by periodically acquiring the file's *attributes*, which includes the time the file was last modified. Clients obtain file attributes by periodically querying the server with a *getattr* request. NFS V3 keeps these attributes requests to a minimum by returning attribute information for **all** operations, thus increasing scalability and performance.

NFS V2 has an 8 KB maximum buffer size limitation which restricts the amount of data that can be sent over the network at any one time. In NFS V3, this restriction has been relaxed, enabling NFS to construct and send large chunks of data. This allows NFS to more efficiently utilize high bandwidth network technologies such as FDDI and 100BaseT and has contributed substantially to NFS performance gains.

8.2 DFS Performance

Unlike NFS, there is no industry standard benchmark for DFS nor is generally available, vendor-independent information available on performance of different DFS implementations. Thus, a comparison of NFS and DFS performance is very difficult.

8.3 Comparison Summary

NFS demonstrates a track record of providing requisite performance for mission-critical applications across large-scale networks. NFS V3 implements features designed to enhance performance further. There is very limited publicly available information on DFS performance.

9 Multiplatform Support

It is typical for corporations to have a mixture of dissimilar system that do not allow information to be easily shared throughout the corporation. The availability of open, multiplatform solutions is very important to those integrating heterogeneous systems into enterprise networks.

9.1 Multiplatform Support for NFS

The multivendor NFS installed base tripled during 1991-1994 [Dataquest95] and is expected to grow from 8.5 million nodes in 1994 to 12 million nodes in 1997. Growth in NFS deployment is forecast to be strong in both UNIX and PC segments and it is expected to remain the dominant distributed file service for heterogeneous environments.

NFS is considered an open standard. It is described in several Internet RFCs [IETF1,IETF2] and is also part of X/Open's CAE (Common Application Environment) specification. NFS code is licensable. NFS implementations either developed from RFC specifications or portable source are available today on every major operating system and hardware platform. NFS is a *core* component of all UNIX variants. In addition, NFS software products exist for MS-DOS/Windows/WFW/Windows NT, VMS, MacOS, and MVS.

9.1.1 Connectathon

The goal of the annual Connectathon event is to provide a forum for different vendors to test interoperability of their NFS client and server products. Almost all major hardware and operating system vendors are represented at Connectathon. Connectathon serves another important purpose: to provide a forum for NFS vendors to plan improvements to NFS. Enhancements such as NFS V3 have been the result of such work.

9.2 Multiplatform Support for DFS

DCE fundamental services and DFS are in relatively early stages of deployment. Little data is publicly available regarding existing or projected installed base for DCE or DFS.

Implementations of DCE fundamental services for primarily UNIX and PC operating environments have been shipping since 1992-1993. These include implementations for HP-UX, AIX, SunOS/Solaris, and DOS/Windows environments. DFS implementations (available only as add-on products) for UNIX environments have been shipping since 1995. PC implementations for DFS (limited to supporting Windows NT) have been shipping only since 1996.

9.3 Comparison Summary

NFS has much stronger multivendor support, a much larger installed base and *de facto* standard status compared to DFS. This makes it a much superior offering as an enterprise-level data services infrastructure.

10 Conclusion

This paper has a key thrust: to compare NFS and DFS distributed file services on how each meets key requirements for mission-critical information sharing in enterprise networks. These include caching, administration, availability, administration, global support, security, and performance. The comparison indicates the following conclusions:

- NFS and DFS both provide equivalent functionality for client-side caching. DFS token passing can be difficult to administer and complicated to implement.
- Both file services allow centralized file system administration and location-independent file server access. NFS global namespace is configurable allowing administrators to setup naming policies to suit the needs of their organization while DFS global namespace is fixed.
- XFN is the next-generation file naming/location capability for both NFS and DFS. However, XFN support for NFS has been available since 1995 while XFN support for DFS will be available in 1997 at the earliest.
- Base-level DFS provides more availability functionality than standard NFS such as support for read-write replication and dynamic client failover. Unlike DFS, however, NFS supports value-added solutions from third parties, such as Auspex's ServerGuard and DataGuard software, allowing very high availability for NFS data services.
- The NFS authentication model allows a number of degrees of flexibility not available in the DFS authentication model. DFS is superior to NFS in the support of ACLs. However, use of ACLs requires additional overhead in setup and on-going administration.
- NFS demonstrates a track record of providing requisite performance for mission-critical applications across large-scale networks. NFS V3 implements features designed to enhance performance further. There is very limited publicly available information on DFS performance.
- NFS has much stronger multivendor support, a much larger installed base and *de facto* standard status compared to DFS.

The overall conclusion of the paper is that NFS is a significantly superior solution for information sharing in multivendor enterprise network environments. Auspex is strongly committed to supporting the latest advances in NFS technology and in allowing NetServer customers to take advantage of such technology in a timely fashion. In addition, Auspex continues to monitor DFS evolution and will support it if market conditions warrant.

11 Acknowledgments

Many people contributed to the reviews or the results of this paper. They include Brian Atwood, Dan Shea, Ray Villeneuve, Philip Trautman, and David Higgen.

12 References

[Dataquest95] *NFS and Heterogeneous File Sharing—The Choice is Easy.*
Dataquest Perspective, Client/Server Program, August 7, 1995.

[Kong95] Michael M. Kong
DCE: An Environment for Secure Client/Server Computing.
HP Journal, Vol. 46, No. 6, December 1995.

[Nemeth1] *UNIX System Administration Handbook.*
Evi Nemeth, Garth Snyder and Scott Seebass, Prentice Hall, 1995. 2nd Edition.

[OSF1] *File Systems in a DCE Environment.*
OSF White Paper, OSF-O-WP8-0990-2.

[Pawlowski94] Pawlowski, B., et. al.
NFS Version 3 Design and Implementation.
Winter USENIX Conference Proceeding, USENIX Association, Berkeley, CA, January 1994.

[Stern92] *Managing NFS and NIS.*
Hal Stern, O'Reilly & Associates, 1992.

[Welch91] Brent B. Welch
Measured Performance of Caching in the Sprite Network File System.
University of California, Berkeley, 1991.

[XFN95] X/Open Federated Naming Specification C403 ISBN 1-85912-052-0, 7/95.

[IETF1] *RFC-1094 (NFS Version 2 specification).*
DDN Network Information Center, SRI International, Menlo Park, CA.

[IETF2] *RFC-1813 (NFS Version 3 specification)*
DDN Network Information Center, SRI International, Menlo Park, CA.