The Evolution of NFS

Dave Hitz & Andy Watson | Network Appliance

TECHNICAL REPORT

Network Appliance, a pioneer and industry leader in data storage technology, helps organizations understand and meet complex technical challenges with advanced storage solutions and global data management strategies.

Network Appliance Inc.

Table of Contents

Introduction

NFS version 3 (NFSv3) arrived almost exactly ten years after Sun Microsystems originally introduced NFS. This leaves some people wondering: What took so long? Will it be another ten years before NFS gets another fresh coat of paint?

In part, these questions reflect a conflation between NFS-the- protocol and NFS-the-implementation. While the NFS protocol itself remained unchanged until NFSv3, NFS *implementations* have changed substantially in the past ten years, and they will continue to change in the future even without another protocol revision.

The menu below outlines the evolution of NFS and reflects the sequence of topics discussed in this document.

Many people are surprised that a protocol can change so much without a protocol revision. The NFS specification RFC 1094 defines the exact format of NFS packets transmitted over the network, but it leaves great flexibility in the hardware and software that actually send the packets. In addition, a protocol can have some flexibility designed in from the start.

For instance, NFS implementations have traditionally used UDP for remote procedure calls (RPC), but the RPC specification allows either UDP or TCP. Finally, services such as the automounter can be added to improve NFS without any change at all to the protocol or its implementation.

1. NFS changes since 1985

The changes since 1985, when Sun first released NFS, have been made either at the implementation level, or by adding related features that have improved NFS without changing the protocol itself.

Close-to-Open File Consistency

In the very early days of NFS, updates made to a file on one NFS client might not show up on another NFS client for many seconds. At first this wasn't a problem because users rarely used files from two NFS clients at once. This inconsistency became unacceptable as window systems became popular, making it easy for a single person to use more than one computer at a time. One might edit a source file on one NFS client, for instance, but compile it on another.

Modern NFS implementations make accessing a file from multiple NFS clients safe by supporting "close-to-open" consistency. This means that if you write and then close a file on one client, and then open and read that same file on another client, the data on the second client is guaranteed to be up-to-date.

This is implemented in the NFS client by writing all modified file data to the server in the close(2) system call, and by checking with the NFS server to make sure that any locally cached data is up-to-date in the open(2) system call.

Close-to-open consistency is a perfect example of how an implementation change can dramatically improve a protocol without a formal revision.

Automounter

The automounter was added in order to allow system administrators to create a global network name space for their organization. The initial scheme of requiring all NFS mountpoints to be added to all clients' /etc/fstab files was cumbersome, but with NIS and the automounter, it is possible to manage a corporate-wide name space centrally. The automounter is an example of how a new feature can improve NFS without requiring any change to the protocol itself, or to its implementation.

Performance Improvements

There have been many performance enhancements to NFS since it was introduced in 1985. Indeed, more improvements have been introduced than can be discussed here in detail, but a few are especially interesting:

- Non-Volatile RAM (NVRAM) to improve write performance;
- Dynamic Retry Time Adjustment;
- Improved Retry Cache Heuristics; and
- Client-side Disk Caching.

NVRAM

The biggest performance problem with the early NFS implementations was the requirement that the server write data to disk before responding to client write requests. Servers are still required to execute "safe" write operations (though with NFSv3 there are new options described later in this document).

This has been largely solved, at least in high performance servers, with the use of Non-Volatile RAM. By temporarily storing the data from NFS Write operations in NVRAM, servers are free to respond to client Write requests without waiting for their own Write operations to complete to disk.

Dynamic Retry

Dynamic retry allows clients to adjust their NFS retry values over time based on the performance they see from the server. If a client notices that a server is slow, it increases the retry timeout value to avoid useless retries. If a client notices that a server is fast, it reduces the retry timeout value, so that retries occur faster when a packet is lost.

Improved Retry Cache Heuristics

A few years ago, Chet Juszczak described a set of improved retry cache heuristics in a USENIX paper ["Improving the Performance and Correctness of an NFS Server," *USENIX Conference Proceedings*, January 1989, pp 53-63]. These techniques have now been incorporated into most NFS server implementations. The basic idea is that in some cases the server can tell that a client's retry request is probably redundant, in which case it can safely ignore the request and suppress a retransmission of the reply. This reduces network congestion under heavy load.

Client-Side Disk Caching

The most interesting performance improvement is probably the "CacheFS" feature introduced in SunOS 2.4, which allows NFS clients to cache NFS-accessed files on disk, instead of in memory. Many people have the misconception that client caching on disk is an AFS feature that NFS cannot support; in fact, there is nothing in the NFS protocol that indicates where clients can cache data. Whether to store cached data in memory or on disk is an implementation decision that has nothing to do with the format of bits that are sent over the network.

2. NFSv3

Despite all the changes that have occurred without a protocol revision, there are some changes that do require the protocol itself to be modified. The driving force behind NFSv3 was the desire to handle 64-bit file sizes. This has become important as CPU chips like the SGI (formerly MIPS) R10000, DEC Alpha, and Sun UltraSPARC have started to support 64-bit integers.

Since the initial NFS protocol specification defined file sizes as being 32 bits long, supporting 64-bit file sizes required the NFS protocol revision to be updated.

Protocol revisions are rare, so it isn't sensible to make just one change. As a result, NFSv3 includes several other changes along with the large file size support. The most interesting are a collection of performance improvements described below. (For a more complete description of NFSv3, see "NFS Version 3 Design and Implementation," by Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel, and David Hitz, USENIX, June 1994, Boston, MA. See also RFC 1813.)

As of April 1996, the following vendors have implemented NFSv3 servers and/or clients: Cray (now part of SGI), DEC, NetApp, SGI, and Sun. There are also numerous NFSv3 client implementations for PCs, from companies like FTP, Hummingbird, NetManage, and others.

Large Block Transfers

The NFSv2 protocol specification restricts read and write operations to 8 KB (kilobytes). In NFSv3, the client and server can negotiate any size they like for reads and writes. Current NFSv3 implementations are indicate a consensus for using 32-KB transfer sizes for 10- and 100-Mbps (Megabit per second) networks, and 48-KB in HiPPI environments which run at 100 MBps (MegaByte per second) or higher.

Allowing the client and server to negotiate the optimal transfer size provides flexibility that will allow NFSv3 implementations to evolve in the future, if necessary, in case new networking technology makes even larger block sizes desirable.

Safe Asynchronous Writes

This feature allows the server to reply to writes immediately, instead of waiting for the data to be put safely on disk or in NVRAM. A new operation, called Commit, lets clients check with the server at some point after the WRITE operation, to verify that the server actually has written the data. The client is required to keep its own copy of the written data until the

Commit succeeds, and if the Commit fails, the client is required to resend its copy of the written data.

For systems without NVRAM, this feature improves write performance for large files. On servers that do use NVRAM, it can reduce the CPU time spent copying data into NVRAM, thereby increasing the total throughput capability of the server.

NFSv3 support for asynchronous writing does not enhance by much the speed with which small files can be written. (Writing a small file might only require one or two async Write requests followed by a Commit.) And it doesn't help operations such as Create, Remove, and Rename at all. Therefore, NVRAM will continue to be critical to fast NFS service, even with NFSv3.

Improved Attribute Returns

In NFSv2, some operations return less information than they should. For instance, the Symlink operation creates a new link, but it does not return the file handle or attributes of the link. As a result, an NFSv2 client must send a Lookup request immediately after the Symlink.

In NFSv3, operations return additional information as appropriate, thus reducing the total number of operations that need to be sent.

The Readdirplus Operation

In NFSv2, the Readdir operation returns the names of the files in a directory, but not the attributes. So to handle a command like "ls -l", the Readdir must be followed by a Lookup operation for each file in the directory. An "ls -l" on a directory with 100 entries would require 101 NFS operations.

NFSv3 supports a Readdirplus operation that returns both directory names and file attributes. As a result, "ls -l" could be handled with just one Readdirplus operation. This is especially useful in speeding up recursive tree-walking commands like "find" and "ls -R".

3. Other Recent Changes and Future Possibilities

Just as we saw ongoing evolution of NFS implementations prior to a change in the NFS protocol itself, this process will continue. Even now that NFSv3 is seeing widespread support, there will still be additional changes that do not require protocol modifications. A few examples of the ongoing evolution of NFS are discussed below.

NFS Over TCP

NFS network traffic can be packaged into two different kinds of IP datagrams: UDP and TCP. Traditionally, NFS has used UDP for nearly all commercially-available implementations. Recently, several vendors (including Network Appliance) have introduced support for NFS over TCP. There are differences between UDP and TCP, affecting network utilization and performance, which should be considered when deciding which to use for NFS.

Running NFS over TCP instead of UDP does not require a protocol revision because the RPC (Remote Procedure Call) layer used by NFS is already defined to work over either UDP or TCP.

Kerberized NFS

Kerberos is a method of secure authentication originally developed as part of MIT's Project Athena. "Kerberizing" NFS is usually only considered in high-security environments, or when operating over a WAN (especially an widely-shared WAN like the Internet).

Although some people have experimented with Kerberos to make NFS more secure, it is not yet widely available. It seems likely that Sun will move to productize this more formally, and that other vendors will follow suit. In particular, although Kerberos Version 4 has been used in recent years, various improvements in Kerberos Version 5 make it a more likely candidate for Sun and other vendors to implement as a standard, interoperable adjunct to NFS.

RSA Encryption

In many network environments, and especially across WAN or Internet links, other users can potentially use their systems to snoop traffic on the shared network. This might give them knowledge of passwords, file contents, or NFS file handles. Encryption can be used to prevent unauthorized access to information in transit, by making it useless to anyone lacking the keys.

For use with NFS, the two most commonly discussed methods of encryption are RSA and DES. RSA offers stronger encryption than DES, but at the cost of slower performance. Therefore, for all but the most stringent of secure environments, RSA is used to encrypt only the "envelope" around a payload of DES-encrypted data. Also, by varying the number of bits in the encryption key, the user can adjust the strength of encryption as a trade-off with performance.

With or without DES, RSA is a leading candidate for inclusion in several vendors' future software releases of NFS. Support for RSA does not require a modification to the NFS protocol specification, because the RPC layer beneath NFS is already architected for this capability.

WebNFS

Recently, Sun has proposed a variation of NFS for use with the Web: WebNFS. Just as many hypertext links are resolved by accessing HTML-formatted documents (indicated by the "http:" URL designation), and others result in downloads via the file transfer protocol ("ftp:" URL designation), a link identified by the proposed "nfs:" URL designation would result in a temporary NFS session to provide file access. The design of WebNFS illustrates the extensibility of the NFS protocol.

4. Conclusion

It seems likely that most long-lived, wide-spread standards survived largely because they could adapt to changing requirements over time. FORTRAN, for instance, has remained an important programming language for decades because it has repeatedly incorporated the most successful features of newer languages. The C language has done likewise in its evolution to C++.

Similarly, the research community provides a continuing source of ideas for the evolution of NFS with projects like the Andrew File System (AFS), Sprite, Spritely NFS, NQ-NFS, and others. AFS is a good example, because many features that appeared first in AFS have now migrated to NFS, including close-to-open file consistency, network-wide name spaces, large block transfers, and client-side disk caching.

As a result of its history of adopting new ideas, we can rely on NFS to remain strong for many years to come. A scientific programmer once said: "I don't know what programming language I'll be using in the year 2001, but I'm sure it will be called 'FORTRAN.'" A similar claim might be made of NFS.

NetApp®

Network Appliance, Inc.
495 East Java Drive
Sunnyvale, CA 94089
www.netapp.com