Kleiman

sun
microsystems

Subject: **Sun NFS Technical Description** Date: 17 October 1984

From: Gary R. Sager

To: List

Attached is a memo describing some of the features of the NFS. The memo is intended to cover the technical properties and merits of the NFS. It should serve as introductory material for a technical audience; marketing people may find it to be a useful source of information, and may want to give it to selected customers or prospects.

Although I consider the memo to be presentable, it is not necessarily finished. I will entertain suggestions for improving it, and we may update it as the NFS project progresses and evolves.

LIST:

Carol Bartz
Dave Cardinal
Bruce Clarke
Lorrie Duval
Beau James
Bill Joy
Bernie Lacroute
Tom Lyon
Marleen Martin
Jay Puri
Jackie Rae
Steve Saperstein
Eric Schmidt
Bill Shannon
Tony West
All Software Managers
All Members, NFS Project

Overview of the Sun Network File System

Bob Lyon, Gary Sager,
and members of the Sun NFS project:
J. M. Chang, D. Goldberg, S. Kleiman,
T. Lyon, R. Sandberg, D. Walsh, and P. Weiss

Sun Microsystems, Inc.

Sun's Network File System (NFS) is a vehicle for sharing file systems in a heterogenous network of machines, operating systems and networks. The NFS interface is open, and Sun encourages customers, users and other vendors to take advantage of the open interface to extend the richness of the product.

1. Introduction

Sun's Network File System (NFS) permits transparent sharing of file systems in a heterogenous network of machines, operating systems and networks; it is intended for use in a networked environment of multiple server and client workstations. The view of the file system seen by a client depends upon mutual agreement of the server and client; servers supply parts of the file system to the network, and clients have a great deal of freedom in setting up their access. It will usually be most convenient to provide a large UNIX† file system to all clients of the network, but service can be tailored to a variety of individual requirements. Clients can make informal arrangements to share files via the NFS without special privilege and without appeal to authority. Because this flexibility can make the maintenance and administration of the system difficult, new tools are provided to make the administrator's job easier.

The NFS was *not* designed by simply extending the UNIX operating system. Instead, the NFS is designed to fit into Sun's network services architecture [1]. Thus, the NFS interface is not a step towards a distributed operating system; rather, the NFS interface is designed to allow a variety of machines and/or operating systems to play the role of client or server. Sun intends to open the NFS interface to customers and other vendors in order to encourage development of a rich set of applications, machines and operating systems for the network.

2. NFS Architecture

Figure 1 illustrates the important elements of the NFS architecture. We first note the distinction between the code which implements the operations of a file system and the data which make up the file system structure and contents. We refer to the former as the file system operations and the latter as the file system data. In our discussions, a *server* is a machine which serves resources to the network, and a *client* is a machine which accesses server resources over the network. A machine may be both a server and a client. A *user* is a person "logged in" at a client, and an *application* is a program or set of programs which run on a client.

There are three interfaces to be considered in the NFS architecture: the operating system interface, the virtual file system node (VNODE) interface, and the network file system (NFS) interface.

† UNIX is a trademark of Bell Laboratories.

The operating system interface is important because it is used directly by applications. In order to insure compatibility with existing applications, the operating system interface has been preserved in the Sun implementation of the NFS.

VNODEs are a re-implementation of UNIX inodes that cleanly separate the implementation of inodes from the semantics. The VNODE interface makes the connection to a particular type of file system (e.g. UNIX or DOS). These are designated in Figure 1 as virtual file systems (VFS's). A local VFS connects to a file system on a local device.

Finally, one type of file system VNODE interfaces to is a remote VFS. VNODE does not differentiate between a local and a remote VFS. It is the remote VFS which defines and implements the NFS interface as seen on the network. The NFS interface is defined using a remote procedure call (RPC) [2] mechanism which allows communication with remote services in a manner similar to procedure calling mechanisms available in many programming languages. The NFS and RPC "high-level protocols" are described using the external data representation (XDR) [3] package. XDR permits a machine-independent representation and definition of high-level protocols on the network.

It is useful to revisit the above discussion to see how the NFS provides five types of transparency for clients:

1. **File System Type:** VNODE in conjunction with one or more local VFS's (and possibly remote VFS's) permits an operating system (hence client and application) to interface transparently to a variety of file system types.
2. **File System Location:** Since there is no differentiation between a local and a remote VFS, the location of file system data is transparent.
3. **Operating System Type:** The RPC mechanism allows interconnection of a variety of operating systems on the network and makes the operating system type of a remote server transparent.
4. **Machine Type:** The XDR definition facility allows a variety of machines to communicate on the network and makes the machine type of a remote server transparent.
5. **Network Type:** RPC and XDR can be implemented for a variety of network and internet protocols, thereby making the network type transparent.

The detailed architecture of Figure 1 reflects the Sun implementation. It is possible to develop simpler architectures at the expense of some of the advantages of the Sun version. In particular, a node may be added to the network by simply implementing the NFS interface. One advantage of the Sun architecture is that the client and server sides of the interface have a common implementation; thus, it is possible for any machine to be a client, server or both. This fact is useful because clients with disks can set up their own informal server arrangements without having to appeal to a system administrator or configure a different system on their workstation.

3. The NFS Interface

As mentioned in the preceding section, a major advantage of the NFS is the ability to mix file systems transparently. In keeping with this, Sun encourages other vendors to develop products which will interface with Sun network services. RPC and XDR have been placed in the public domain and should serve as a standard for anyone wishing to develop applications for the network. Furthermore, the NFS interface itself is open and can be used by anyone wishing to implement an NFS client or server for the network.

The NFS interface defines traditional file system operations for reading directories, creating and destroying files, reading and writing files, and reading and setting file attributes. The interface is designed such that file operations address files with an opaque identifier, starting byte address and length in bytes.

Commands are provided for NFS servers to initiate service (*mountd*), to serve a portion of their file system to the network (*exportfs*), and to retract a portion of their file system from the network (*unexportfs*). A client "builds" its view of the file systems available on the network with the *rmount* and *mount* commands (described later).

The NFS interface is defined such that a server can be *stateless*. This means that a server does not have to remember from one transaction to the next anything about its clients, transactions completed or files operated on. For example, there is no *open* operation, as this would imply state in the server; of course, the UNIX interface uses an *open* operation, but the information in the UNIX operation is remembered by the client to use in later NFS operations.

An interesting problem occurs when a UNIX application *unlink*s an open file. This is done to achieve the effect of a temporary file which will be automatically removed when the application terminates. If the file in question is served by the NFS, the *unlink* will remove the file, since the server does not remember that the file is open. Thus, subsequent operations on the file will fail. In order to avoid state on the server, the client operating system detects the situation, renames the file rather than unlinking it, and *unlink*s the file when the application terminates. In certain failure cases, this leaves unwanted "temporary" files on the server; these files are removed as a part of periodic file system maintenance.

Another example of how the NFS provides a friendly interface to UNIX without introducing state is the *mount* command. A UNIX client of the NFS "builds" its view of the file system on its local devices using the *mount* command; thus, it is natural for the UNIX client to initiate its contact with the NFS and build its view of the file system on the network via a similar *rmount* command. As with the *mount* command, the *rmount* command may be issued at any time. The *rmount* command does not imply state in the server, since it only acquires information for the client (via the Yellow Pages, discussed below) to establish contact with a server. The corresponding *umount* command (shared with UNIX) is thus only an informative message to the server, but it does change state in the client by modifying its view of the file system on the network.

The major advantage of a stateless server is robustness in the face of client, server or network failures. Should a client fail, it is not necessary for a server (or human administrator) to take any action to continue normal operation. Should a server or the network fail, it is only necessary that clients continue to attempt to complete NFS operations until the server or network is fixed. This robustness is especially important in a complex network of heterogenous systems, many of which are not under the control of a disciplined operations staff and may be running untested systems and/or may be rebooted without warning.

An NFS server can be a client of another NFS server. However, a server will not act as an intermediary between a client and another server. Instead, a client may ask what remote mounts the server has and then attempt to make similar remote mounts. The decision to disallow intermediary servers is based on several factors. First, the existence of an intermediary will impact the performance characteristics of the system; the potential performance implications are so complex that it seems best to require direct communication between a client and server. Second, the existence of an intermediary complicates access control; it is much simpler to require a client and server to establish direct agreements for service. Finally, disallowing intermediaries prevents cycles in the service arrangements; we prefer this to detection or avoidance schemes.

The NFS currently implements UNIX-style file protection by making use of the authentication mechanisms built into RPC. This retains transparency for clients and applications which make use of UNIX file protection. Although the RPC definition allows other authentication schemes, their use may have adverse effects on transparency.

Although the NFS is UNIX-friendly, it does not support all UNIX file system operations. For example, the UNIX "special file" abstraction of devices is not supported for remote file systems because it is felt that the interface to devices would greatly complicate the NFS interface; instead, devices are implemented in a local */dev* VFS. Other incompatibilities are due to the fact that NFS servers are stateless. For example, file locking and guaranteed APPEND_MODE are not supported in the remote case.

Our decision to omit certain features from the NFS is motivated by a desire to preserve the stateless implementaton of servers and to define a simple, general interface to be implemented and used by a wide variety of customers. The availability of open RPC and NFS interfaces means that customers and users who need stateful or complex features can implement them "beside" or "within" the NFS.

4. Performance

Our performance measurements indicate that a machine running the VNODE version of the UNIX file system has about 3% less throughput than the same machine running vanilla 4.2 BSD UNIX. Other than this, it is difficult to quantify the performance of the NFS, as performance will depend upon variables such as the benchmark used, processor speed, disk speed and network characteristics. We are currently working on obtaining performance data and intend to use that data to improve NFS performance.

The flexibility of the NFS allows configuration for a variety of cost and performance trade-offs. For example, configuring servers with large, high-performance disks and clients with no disks may yield better performance at lower cost than having many many machines with small, inexpensive disks. Furthermore, it is possible to distribute the file system data across many servers and thereby get the added benefit of multiprocessing without losing transparency. In the case of files which are mostly read-only, it is possible to keep multiple copies on several servers to avoid bottlenecks to the information. In more complex situations, trade-offs can be made between keeping portions of the file system data local or remote.

The Sun implementation of the NFS has a number of performance enhancements, such as asynchronous service of multiple requests, "fast paths" for critical operations, caching of disk blocks and asynchronous read-ahead and write-behind. The fact that caching and read-ahead occur on both the client and the server effectively increases the cache size and read-ahead distance of the clients. It should also be noted that caching and read-ahead do not add state to the server because nothing (except performance) is lost if cached information is thrown away. In the case of write-behind, both the client and server attempt to flush critical information to disk whenever necessary to reduce the impact of an unanticipated failure.

5. The Role of the Yellow Pages

The Yellow Pages (YP) [4] is an independent service from the NFS, but we include a discussion of it because it plays an important role in the initialization and administration of the NFS.

From the point of view of the servers and clients, the YP appears to be a centralized read-only database. For a client, this means that an application's access to the data served by the YP is independent of the relative locations of the client and server.

It is important to note that the YP provides a client access to data without recourse to the file system. This fact allows greater generality in the initialization of clients by allowing them to access information needed to mount file systems without requiring them to mount the filesystem containing a file with that information.

The YP is actually a collection of cooperating server processes which use a simple discipline to distribute data among themselves. Thus, the servers share the load of providing access to data and the failure of a server need not disable the network. The YP does not implement a true distributed database: for every relation in the database, one YP server is designated to control the update of data for the entire collection of YP servers. This means that the administration of an entire network of servers and clients can be done from a single point of contact. Should the control server fail, it is possible to designate an alternate server as the control. The policy for distributing changes through the network yields a weak form of consistency: the databases will be consistent after a "reasonable" time has elapsed. A system administrator can choose to have changes distributed periodically according to a schedule, or can cause them to propagate immediately.

The most obvious use of the YP is for administration of /etc/passwd. Since the NFS uses a UNIX protection scheme across the network, it is advantageous to have a common /etc/passwd database for the servers and clients on the network. The YP allows a single point of administration and gives all servers and clients access to a recent version of the data, whether or not it is held locally. To install the YP version of /etc/passwd, existing applications were not changed, they were simply relinked with library routines that know about the YP service. Conventions have been added to the /etc/passwd library routines that allow each client to administer its own local subset of /etc/passwd; the local subset modifies the client's view of the system version. Thus, a client is not forced to completely bypass the system adminstrator in order to accomplish a small amount of personalization.

The YP interface is implemented using RPC and XDR, so the service is available to non-UNIX operating systems and non-Sun machines. The YP servers do not interpret the data in the databases, so it is possible for new new databases to take advantage of the YP service without modifying the servers.

6. Conclusion

The NFS is designed to provide file system service in a heterogenous network of machines and networks. The Sun implementation of the NFS interfaces the Sun Workstation and UNIX operating system to the NFS. New tools are provided to aid in the administration of an NFS network. The NFS interface is designed to encourage further development of the nodes and the network; Sun plans to use it as a basis for further product development and will open the interface to customers and other vendors in order to encourage an atmosphere of mutually beneficial product development.

7. Acknowledgements

The network services architecture and the NFS were championed at Sun by Bill Joy. Bill Shannon and other Sun employees have provided a great deal of valuable advice to the NFS project.

8. Bibliography

[1] Joy, W. N. *The UNIX System in the Laboratory*, UNIX/WORLD, v1n4, 1984, pp 34-38.

[2] Remote Procedure Call Reference Manual, Sun Microsystems, Inc., Oct., 1984.

[3] External Data Representation Reference Manual, Sun Microsystems, Inc., Oct., 1984.

[4] Yellow Pages reference, to be written.

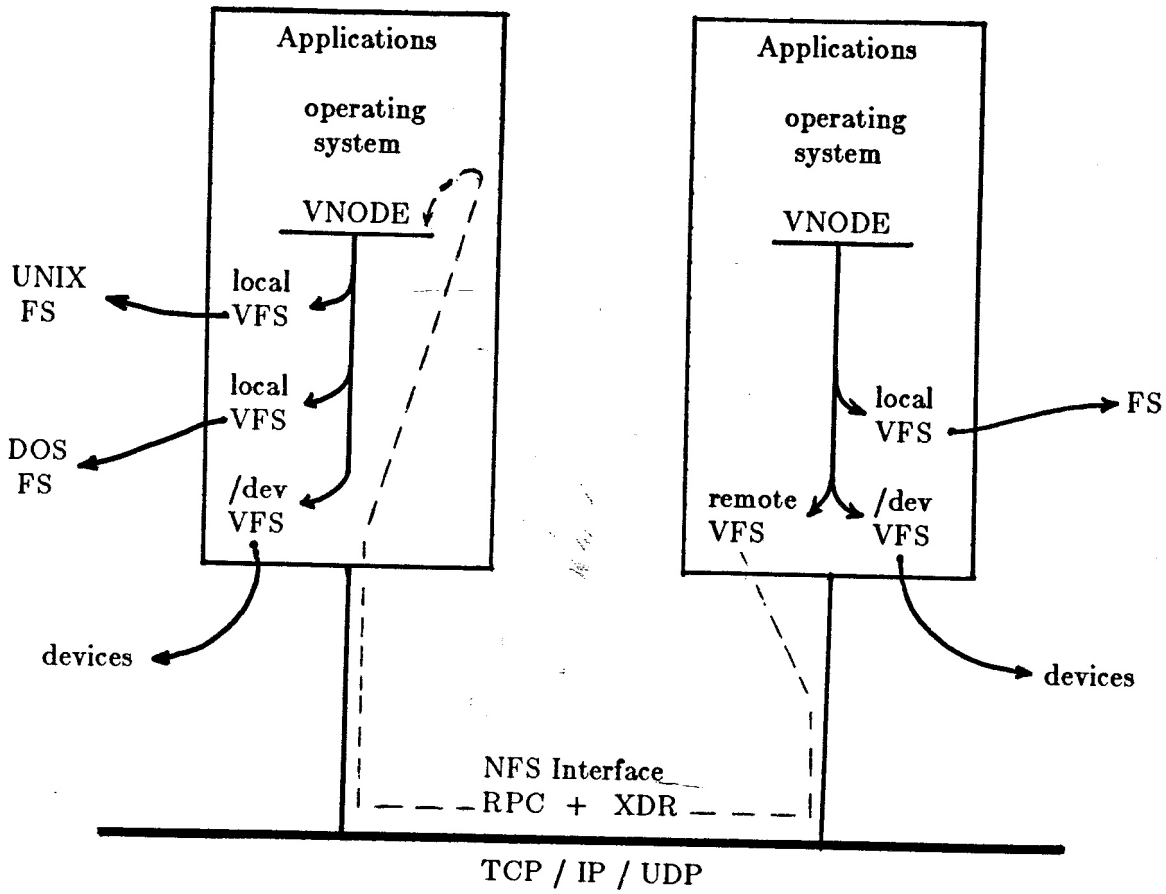

Figure 1: Architecture of the Sun NFS