Overview of the Sun Network File System

Conference Paper · January 1985

CITATIONS
5

READS
18

9 authors, including:

David Goldberg
eBay Inc
60 PUBLICATIONS **11,314** CITATIONS
SEE PROFILE

Tom Lyon
Princeton University
18 PUBLICATIONS **721** CITATIONS
SEE PROFILE

Overview of the Sun Network File System

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

Sun Microsystems, Inc.

Sun's Network File System (NFS) is a vehicle for sharing file systems in a heterogeneous 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 heterogeneous network of machines, operating systems and networks. The view of the file system seen by a client depends upon mutual agreement of the client and servers; 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 extending the UNIX operating system onto the net. 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 has opened the NFS interface to customers and other vendors in order to encourage development of a rich set of applications, machines and operating systems working together on the network.

† UNIX is a trademark of Bell Laboratories.

2. Architecture and Implementation

To add precision to the discussions which follow, it is necessary to define several terms and concepts. First, there is a distinction between the code implementing the operations of a file system and the data making 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*. A *server* is a machine that serves resources to the network, and a *client* is a machine that accesses server resources over the network. A machine may be both a server and a client. Finally, a *user* is a person "logged in" at a client, and an *application* is a program or set of programs that run on a client.

In the Sun implementation of the NFS architecture (Figure 1), there are three interfaces to be considered: the operating system interface, the virtual file system node (VNODE) interface, and the network file system (NFS) interface. The operating system interface is important because it is

used directly by applications. The UNIX operating system interface has been largely preserved in the Sun implementation of the NFS, thereby insuring compatibility for existing applications.

VNODEs are a re-implementation of UNIX inodes that cleanly separate file system operations from the semantics of their implementation. Above the VNODE interface, the operating system deals in vnodes; below the interface, the file system may or may not implement inodes. The VNODE interface can connect the operating system to a variety of file systems (e.g. 4.2 BSD or DOS); these are designated in Figure 1 as VFS's (virtual file systems). A local VFS connects to file system data on a local device.

The remote VFS defines and implements the NFS interface. The remote VFS uses a remote procedure call (RPC) [2] mechanism; RPC 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.

Figure 1 shows the flow of a request from a client (on the left) to a collection of file systems. In the case of access through a local VFS, requests are directed to file system data on devices connected to the client machine. In the case of access through a remote VFS, the request is passed through the RPC and XDR layers onto the net. In the current implementation, we use the UDP/IP protocols and the Ethernet. On the server side, requests are passed through the RPC and XDR layers to an NFS server; the server uses the VNODE interface to access one of its local VFSs and service the request. This path is retraced to return results.

It is useful to revisit the above discussion to see how Sun's implementation of the NFS provides five types of transparency:

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.

Simpler NFS implementations are possible at the expense of some advantages of the Sun version. In particular, a client (or server) may be added to the network by implementing one side of the NFS interface. An advantage of the Sun implementation is that the client and server sides are identical; thus, it is possible for any machine to be client, server or both. Users at client machines with disks can arrange to share via the NFS 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 to interface with Sun network services. RPC and XDR have been placed in the public domain and 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 uninterpreted 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 *mount* command (described below).

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 for 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 that is 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 an extended *mount* command. This *mount* command does not imply state in the server, since it only acquires information for the client to establish contact with a server. The *mount* command may be issued at any time, but is typically executed as a part of client initialization. The corresponding *unmount* command (which replaces the UNIX *umount*) is 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 heterogeneous 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 that 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. Sun is considering implementation of a set of tools for use by applications that need file or record locking, replicated data, or other features implying state and/or distributed synchronization; however, these will not be made part of the base NFS definition.

4. Example

Figure 2 shows a possible file system view as seen by three machines. The server machine (bottom) has exported the */usr2* and */usr* trees with the commands:

```
exportfs -a /usr2
exportfs -a /usr
```

The "-a" option indicates that the directories can be mounted by any client.

The clients (called "blue" and "red") have each mounted subtrees with the commands:

```
mount -t nfs server:/usr2 /usr2
mount -t nfs server:/usr/src /usr/src
```

The "-t nfs" option is used to indicate that the type of file system being mounted is remote over the network; the location of the file system is indicated by prepending the server's machine name and a colon to the name of a directory in an exported tree. Note that the clients selectively mount */usr/src* rather than all of */usr*. With this arrangement, all three machines share the trees named */usr2* and */usr/src*.

Figure 2 also shows a sharing arrangement between the blue and red machines. The blue machine has a directory on a local disk with the name */usr/proj*, and has exported it with the command:

```
exportfs -a /usr/proj
```

The red machine has mounted the subtree with the command:

```
mount -t nfs blue:/usr/proj /usr/proj
```

Thus, the blue and red machines share the */usr/proj* directory, while the server is unaware of the arrangement. If users of the blue machine feel that it is important to keep the sharing more private, the *exportfs* command can restrict the set of machines permitted to mount the subtree.

5. Performance

Our performance goal is to achieve the same throughput as measured on a previous release of the system that used the network only as a disk (and thus did not permit sharing). Measurements are taken on a well-defined configuration and set of benchmarks. Current indications are that we will attain that goal.

The Sun implementation of the NFS has a number of performance enhancements, such as "fast paths" to eliminate the work done for high-runner operations, asynchronous service of multiple requests, 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. Caching and read-ahead do not add state to the server; 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; clients do not free write-behind blocks until the server verifies that the data is written.

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 get the added benefit of multiprocessing without losing transparency. In the case of read-only files, copies can be kept 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.

6. The Yellow Pages

The Yellow Pages (YP) [4] is an independent service from the NFS; we include this discussion because the YP plays an important role in initialization and administration of the NFS as installed at Sun.

From the point of view of the servers and clients, the YP is 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 a collection of cooperating server processes that 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. Thus, the administration of an entire network of servers and clients is done from a single point of contact. Should the control server fail, an alternate server can be designated as the control. The policy for distributing changes through the network yields a weak form of consistency: the databases across the network 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 library routines that access */etc/passwd* to 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. YP servers do not interpret data in the databases, so it is possible for new databases to take advantage of the YP service without modifying the servers.

7. Conclusion

The NFS is designed to provide file system service in a heterogeneous network of machines and networks. Sun has implemented the NFS interface for the Sun Workstation and 4.2 BSD UNIX operating system. 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 has opened the interface to customers and other vendors in order to encourage an atmosphere of mutually beneficial product development.

At this time (December 18, 1984), the NFS has been used for serious work for over two months; steady improvements in robustness and performance have been made based on experience gained. During the first weeks, crashes brought to light many bugs; however, no files were lost. We do not yet have MTBF data, as we tend to boot new versions before failures occur (every 3 to 5 days). Presently, we are converting all internal development to use the NFS, and will collect reliability data based on general use within Sun. The NFS will be shipped to customers in Spring, 1985.

Sun is collaborating with several other vendors to make the NFS available as part of their product line.

8. 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.

9. Bibliography

[1] Joy, W. N. *The UNIX System in the Laboratory*, UNIX/WORLD, vol. 1, no. 4, 1984, pp. 34-38.

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

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

[4] Yellow Pages Reference Manual, Sun Microsystems, Inc.

Figure 1: NFS Architecture and Implementation

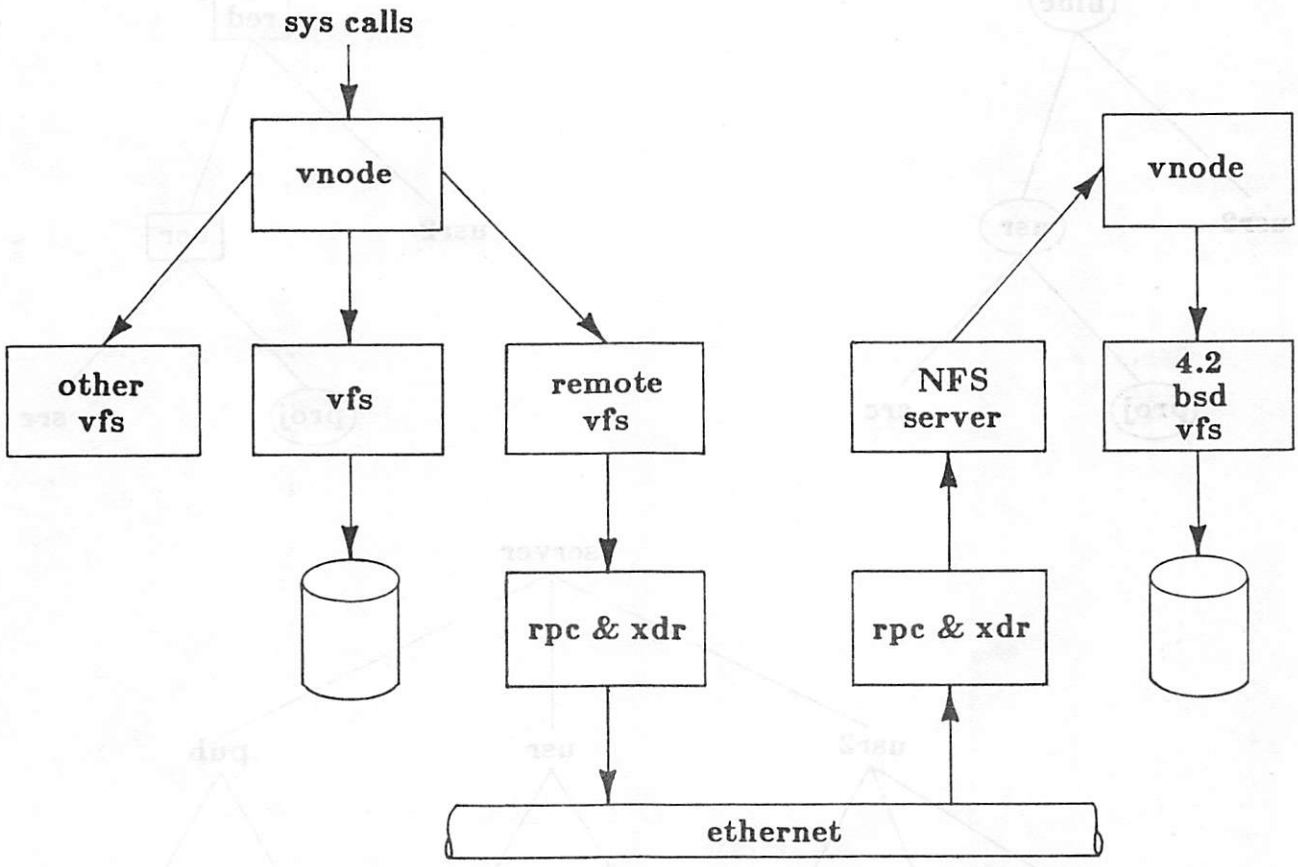

Figure 2: Example NFS Use