NFS Version 4

17/06/05

Thimo Langbehn

Operating System Services and Administration Seminar 2005
Hasso-Plattner-Institute for Software Systems Engineering

thimo.langbehn@student.hpi.uni-potsdam.de

Abstract

The Network File System Version 4 (NFSv4) is an operating-system-independent file sharing protocol, defined in April 2003. The Main Task of it is to integrate remote resources in local file systems, while covering performance, portability and protection issues.
This document describes the NFSv4 from a technical and an administrator's point of view. It compares NFSv4 to other file sharing methods and explains the core concepts. Furthermore, an installation manual for Linux-based operating systems is provided and several configuration options are explained.

Contents:

1. The NFS Version 4 protocol

1.1. History

The Network File System (NFS) was initially developed by Sun Microsystems in 1984 to provide distributed and transparent file access in a heterogeneous network In 1998 Sun ceded change control to the Internet Engineering Task Force (IETF). Later in 1998 Sun published several improvements which advanced the development of NFS Version 4.

1.2 Overview

The task of NFS is to specify a protocol enabling users at one system (called client) to access files on another (called server). The Version 4 was developed to achieve several design goals and to correct some of the known problems of prior versions. In particular, the requirements of NFS Version 4 are specified as:

- Improved access and performance on the Internet
- Extensible security models
- Cross-platform interoperability
- Extensibility of protocol-functionality

NFS Version 4 is designed to be operated system independent. To archive this, it has to support several options and procedures required by one Operating Systems and not supported by another. It has to be able to operate through firewalls and between systems with different users and architecture. Furthermore, it has to address performance issues to operate properly in the Internet environment.

1.3 Basic Structure

A NFS Version 4 client (in the following called client) communicates with a corresponding NFS Version 4 Server (called server) via remote procedure calls (RPC's). The client sends a request and gets a reply from the server. It is not necessary (but desirable) that the server is able to send a request to one of its clients. This, and the fact that NFS Version 4 uses a fix and well known port number (2049), enables NFS Version 4 to work through firewalls.

To ensure that send Messages arrive correctly, IETF-approved congestion control transport protocols have to be used for transport. Furthermore, NFS Version 4 implementations must support the Transmission Control Protocol (TCP) to enhance the possibilities for interoperability.

A NFS Version 4 server can only provide (export) a single, hierarchical file system tree . If a server has to share more then one logical file system tree, the single trees are integrated in a new virtual root directory. This construction, called pseudo-file-system, is the one which is provided (exported) to clients.

To access an exported file or directory, a client has to get a identifying structure called filehandle. When a client establishes a connection to a server the client gets the root filehandle which is a handle to the root of the exported file system tree.

A client which mounts this pseudo file system will only see those parts it is authorized to read. To check this authorization the server and his clients have to use a unique global user name space. This is achieved by connecting the user or group name with the full qualified name of the server, creating a string like user@server.domain.com .

To improve performance, to enable efficient locking and to skip repeated authentication, client and server have to store several information about each other and their current connection. This makes NFS Version 4 a **stateful** protocol.

Different platforms need to store various information corresponding to a file or directory. NFS Version 4 provides those information and the information needed by the protocol itself in key-value pairs called named attributes. There are three groups of attributes, mandatory, required and optional. This describes the priority to support an attribute by any NFS Version 4 implementation. Since a client can ask what attributes are supported by a server implementation, that attributes are extensible and therefore capable to support the needs of future platforms and programs.

1.4 Procedures and operations:

The NFS Version 4 protocol uses only two different RPC procedures, `COMPOUND` and `NULL.` While `NULL` is only used for error-handling purpose, `COMPOUND` is a container-like message listing several operations a client wants to perform on a server (and, if possible, the other way around). Whenever one of those operations fails, the server will cancel the remaining operations and return the results of the already executed operations plus the error.

In NFS Version 4, the stateful operations `OPEN` and `CLOSE` are introduced. While this allows the use of NFS Version 4 in a windows environment, it additionally provides the server the ability to delegate authority to a client (see 1.5 Delegation for details).

Another change to prior versions is the simplicity of a single operation. The `LOOKUP` operations for example only sets the current filehandle to the resolved location. To get this filehandle, a client may use the `GETFH` operation.

1.5 Delegation

Delegation is a new concept allowing servers to temporally transfer full authority over a file or directory tree to a client (using a state id to remember the client and the used connection). In particular, the server guarantees that no further requests to that file or directory will be granted without revoking the delegation first. At this point, the server has to be able to callback to the client. For this reason, delegations can only be granted to those clients reachable by the server.

On the other hand, a client-machine which received a delegation of a file or directory is able to fully cache all operations on those files and to grant locks to different processes on his own without having to consult the server first. This may increase performance of several applications while strongly reducing the network-traffic needed.

1.6 Filehandles

A filehandle is a per server unique data structure to identify a single file system object. Two filehandles which are identical (on the same server) refer to the same object, but the reverse is not granted, two different filehandles on the same server do not necessarily identify two different file system objects.

The NFS Version 4 Protocol specifies two types of filehandles, `VOLATILE` and `PERSISTANT`. While `PERSISTANT` filehandles always point to their target, `VOLATILE` filehandles may expire. In consequence, a client has to store path information for regenerating an expired `VOLATILE` filehandle.

2. A Linux NFS Version 4 implementation

The following tries to explain how to set up a Debian Linux NFS Version 4 server and client using the CITI (citi.umich.edu) reference implementation. I assume that Kerberos is already working.

2.1 Description

The CITI NFS to set up consists of the Kernel Modules nfs, auth_rpcsec_gss, rpcsec_gss(_krb5) and nfsd (on the server). The **nfs** Module provide user-transparent access to the file systems mounted via NFS while **auth_rpcsec** and **rpcsec_gss_krb5** are used to handle secure communication. The **nfsd** module finally provides the exported file systems to other clients.

The programs rpc.idmapd, rpc.mountd, rpc.gssd or rpc.svcgssd and rpc.nfsd are used for communication.

- **rpcsec_gss(_krb5)**: This module provides security for protocols using RPC. Before exchanging any RPC requests, the RPC client must first establish a security context with the RPC server. rpcsec_gss_krb5 in particular uses Kerberos to establish this security context.

- **rpc.idmapd**: This daemon is used by the NFS Version 4 kernel client and server to translate user and group IDs to names and vice versa. It reads the configuration file in /etc/idmapd.conf .

- **rpc.nfsd**: The main functionality is handled by the nfsd kernel module. This program is merely used to start a specified number of kernel threads.

- **rpc.mountd**: The rpc.mountd daemon implements the NFS mount protocol. It checks MOUNT requests received from a client against the list of exported file systems and decides whether the client is permitted to mount or not. If the client is permitted to mount the requested file system, rpc.mountd obtains a filehandle, returns it to the client and creates an entry in the rmtab.

- **rpc.gssd**: This daemon is used by the client's rpcsec_gss module to establish security contexts. It uses the rpc_pipefs file system to communicate with the kernel. Ordinarily, it uses a cached ticked for the user. The -m option is used to map the root user to the principal nfs/host

- **rpc.svcgssd**: This daemon is used by the server's rpcsec_gss module to establish security contexts. It uses the proc file system to communicate with the kernel.

2.2 Installation

This section aims to explain how to set up a NFS Version 4 client and server on Debian-Linux Systems. If Kerberos authentication shall be used, an additional Kerberos infrastructure is required.

What is needed first is a NFS Version 4 capable kernel, which is the case with the Debian kernel in Version 2.6.

To access the Debian packets provided by citi.umich.edu, the following line has to be added to /etc/apt/sources.list of all systems (the second one is optional) :

```
deb http://www.citi.umich.edu/projects/nfsv4/debian unstable main
deb-src http://www.citi.umich.edu/projects/nfsv4/debian unstable main
```

The packets needed on the server are:

```
nfs-common
nfs-kernel-server
libnfsidmap1
librpcsecgss1
```

[`acl, libacl1`]

If Access Control Lists are needed, these packets are required. Note that these are CITI-modified versions. So, they should be upgraded to the CITI version if they are already installed.

After installing them, the communication pipe for the svcgssd subsystem has to be created with

```
mkdir /var/lib/nfs/rpc_pipefs
```

and, together with the nfsd file system, has to be entered in /etc/fstab as:

```
rpc_pipefs /var/lib/nfs/rpc_pipefs rpc_pipefs defaults 0 0
nfsd       /proc/fs/nfsd           nfsd       defaults 0 0
```

After that, the modules are loaded with modprobe for testing purpose. Later, they can be added to /etc/modules.

```
modprobe auth_rpcgss rpcsec_gss_krb5
```

Before the server can be started, the file system points to be provided should be inserted in /etc/exports (see section 3.3.1) and the settings in /etc/idmapd.conf should be adapted.

Finally, the server tools are started with:

```
rpc.mountd
rpc.idmapd
rpc.svcgssd
```

`rpc.nfsd 6` to start 6 nfs server threads in the kernel.

On the client:

`nfs-common`

`libnfsidmap1`

`librpcsecgss1`

`mount` (CITI version with NFS Version 4 support)

[`acl, libacl1`]

After installing this packets, in the same way as on the server, the communication pipe for the gssd subsystem has to be created with:

```
mkdir /var/lib/nfs/rpc_pipefs
```

and has to be entered in `/etc/fstab` as:

```
rpc_pipefs /var/lib/nfs/rpc_pipefs rpc_pipefs defaults 0 0
```

Then, the modules are loaded with modprobe for testing purpose. Later they can be added to `/etc/modules`.

```
modprobe auth_rpcgss rpcsec_gss_krb5
```

After that the configuration files (`/etc/idmapd.conf`) should be edited and then the daemons can be started with:

```
rpc.idmapd
rpc.gssd -m
```

The client may now mount exported file systems using `nfs4` as file system type. To mount an exported file system with Kerberos the option `sec=krb5` has to be used.

```
mount -t nfs4 server.domain.com:/ /target/path/
mount -t nfs4 -o sec=krb5 server.domain.com:/ /target/path/
```

Note that Kerberos has to be set up correctly and configured for NFS Version 4 (for an explanation how to do this, refer to the next section).

Kerberos:

To use CITI's NFS Version 4 implementation together with the Kerberos authentication, an nfs machine principal must be created and its key stored in the local keytab. This can be done by using kadmin:

```
kadmin: addprinc -randkey nfs/hostname.domainname
kadmin: ktadd -e des-cbc-crc:normal -k /etc/keytab \
         nfs/hostname.domainname
```

Furthermore, the `/etc/hosts` file must list the fully-qualified domain name as the first entry on the line with the machine's IP address, and the machine's name must not be included in the localhost line.

2.3 Configuration

exports:

This is the configuration file read by the **exportfs** tool to configure the kernel NFS server.

A Line in exports consists of a file system point to share followed by a list of client-(options) pairs to which this point shall be accessible with the given options.

A host may be an IP-address/net, a NIS group or a DNS-name. Both of it may use the wildcards * ? . A special host-identifier used in this implementation is gss/krb5 (gss/krb5i, gss/krb5p) which matches to any host that can authenticate itself using a nfs/host Kerberos ticket. The security provided by those are as follows:

krb5:

Only the header is signed. In consequence, a client or server is able to check the sender of an request or response packet.

krb5i - Integrity:

The header and the body of each request and response packet is signed. Nobody without the corresponding keys should be able to modify packet-content without invalidating the signature.

krb5p - Privacy:

The header of an request or response is signed, and the body is encrypted. Nobody without the corresponding keys should be able to modify the packet or even read the content.

The following are some important options, to get a complete list refer to the exports-man page:

insecure, secure (default = secure):

The client's source port has to be smaller then IPPORT_RESERVED.

ro, rw (default = ro):

Allow both read and write requests on this NFS volume.

fsid (fsid=value):

This option can be used to assign an unique number to an exported files system. fsid=0 has to be used to specify the 'root' directory of the exported pseudo-file-system.

subtree_check, no_subtree_check (default = subtree_check):

Specifies whether the server has to make sure that a subdirectory is in the exported tree or not.

hide, nohide (default = hide):

Setting the nohide option on a file system makes it visible and available to an appropriately authorized client without the need of explicitly mounting it.

async, sync (default = sync):

This option allows the NFS server to violate the NFS protocol and reply to requests before any changes made by that request have been committed to stable storage (e.g. disc drive).

no_wdelay, wdelay (default = no_wdelay):

Whether to allow lazy writing or not.

mp: Do export a point only if the specified file system has been mounted correctly.

no_root_squash, root_squash (default = root_squash):

Map requests from uid/gid 0 to the anonymous uid/gid.

all_squash, no_all_squash (default = no_all_squash):

Map all uids and gids to the anonymous user.

anongid, anonuid:

These options explicitly set the uid and gid of the anonymous account.

This is an example exports file. /home/shares is exported using any supported authentication:

```
/shares    *(ro,fsid=0,insecure,no_subtree_check)
/shares    gss/krb5(rw,fsid=0,insecure,no_subtree_check)
/shares    gss/krb5i(rw,fsid=0,insecure,no_subtree_check)
/shares    gss/krb5p(rw,fsid=0,insecure,no_subtree_check)
```

idmapd.conf:

This is the configuration file for **idmapd** which maps IDs to names and vice versa.

It consists of two sections, [General] and [Mapping], each of it may contain lines of the form: name = value

Following variables are used:

[General]

Verbosity:

Specifies the verbosity-level of idmapd, this may be a value 0, 1, 2, 3, ...

Pipefs-Directory:

Specifies the location of RPC pipefs. The default value is "/var/lib/nfs/rpc_pipefs".

Domain:

This is used by NFS internally and should be the domain of the server.

[Mapping]

Nobody-User:

Specifies the NFSv4 nobody user. The default value is "nobody".

Nobody-Group:

Specifies the NFSv4 nobody group. The default value is "nobody".

This is an example of idmapd.conf:

```
[General]
Verbosity = 0
Pipefs-Directory = /var/lib/nfs/rpc_pipefs
Domain = asg-platform.org

[Mapping]
Nobody-User = nobody
Nobody-Group = nogroup
```

rmtab:

For every successful mount, the mountd process adds a corresponding entry to the rmtab file (/etc/lib/nfs/rmtab). This entry is removed when mountd process receives an unmount request. However, this file is not used by the CITI NFSv4 implementation and it may contain invalid information.

2.4 Problems

Several of the errors one might get while installing or configuring are very irritating. In the following section, some of them are listed together with their possible causes.

Linux with CITI Implementation:

```
$>mount -t nfs4 -o sec=krb5 server:/ /target/path/
$>cd /target/path
bash: cd: client: Input/output error
```

The root user may not have the necessary access rights to the exported file system. This may be caused by the *_squash options.

Sun Solaris:

```
$>root@sun-solaris:~# mount -F nfs -o ro server.domain.com:/ /path/to/mount
nfs mount: mount: /path/to/mount: Not owner
```

This may occur if the file system is not exported to 'the world' at all. This can be checked using the 'exportfs' command on the server.

3. Resources

- RFC3530, NFSv4 (August 2003)
- RFC1831, RPC (August 1995)
- RFC1832, XDR (August 1995)

- CITI NFSv4 Open Source Reference Implementation
  http://www.citi.umich.edu/projects/nfsv4/ (August 2005)

- SUN System Administration Guide: Network Services
  http://docs.sun.com/app/docs/doc/816-4555/6maoqui84 (July 2005)

- die.net, Linux Manual Pages http://www.die.net/doc/linux/man (June 2005)