The Network File System

Jaime Spacco David Hovemeyer

Outline

- Introduction

- Protocol

- Conclusions

Introduction

- NFS - Network File System
- Version 2 is described in RFC 1094
- Version 3 is described in RFC 1813
- Layered on top of RPC and XDR
  - ▷ RPC - Remote Procedure Call
  - ▷ XDR - eXternal Data Representation
- May be implemented as a stand-alone protocol

RPC – protocol for making prodedure calls to remote hosts

- Described in RFC 1057
- Uses XDR for arguments and return values
- Uses either UDP or TCP for transport
- A request-reply protocol, which is cool because call and response headers differ
- Approximates “at most once” semantics to handle duplicated requests
- Uses port mapper (described in appendix A) to connect client to server

- Supports authentication, but in a much later chapter than I read

XDR - A ‘canonical format for data exchange’

- Described in RFC 1014
- Specifies endianness and alignment
- Can describe complex recursive data structures
- Looks a lot like C actually...

Design goals

- Simple
  - Can be implemented on all kinds of clients
  - Doesn't provide access to block devices or printers
- Stateless
  - No hard state is kept on the server; i.e. server doesn't track who has opened a file
  - Server can crash and reboot; clients won't lose data
  - Implies that I/O on server must be synchronous!
  - In practice, servers often store some state information in a cache to help with "at most once" semantics
- Unix semantics

Some notes about NFS versions

- NFS is up to version 3, version 4 under construction
- But version 2 is still widely deployed
- Version 2 is geared toward Unix, and doesn't provide access to all available features on other OSes
- v3 removes some unused fields from v2
- Also removes two unimplemented/undefined procedures
- and of course, adds more functionality
- Including:
  - ▷ 64 bit offsets to allow files bigger than 4 Gb

  - explicit access checks, freeing protocol from reliance on UNIX-style bit modes
  - changes filehandles from fixed 32 bytes to variable length 64 byte max
  - removes maximum data sizes, allows clients and servers to negotiate preferred sizes
  - Allows errors to return data
  - Every procedure now returns file attributes to cut down on number of GETATTRibutes requests by clients
  - What do we do about a 32-bit client and a 64-bit server? Left up to the implementors...

- Which one are we focused on? I'll go over v2 procedures

- And mention v3 tidbits when I can

- and if there's time talk about some of the new v3 procedures

Outline

- Introduction
- Protocol
- Conclusions

Protocol

- The Filehandle
  - ▷ Opaque identifier– 32 byte fixed-length in v2, 64 byte variable length in v3
  - ▷ Refers to a file or directory
  - ▷ Virtually all NFS protocol functions take it as a parameter
  - ▷ Clients expect filehandles to be persistent, i.e. to survive a server crash.
  - ▷ It may become ‘stale’ at any time (for example, if someone moves the file)
  - ▷ A separate protocol (the MOUNT protocol, Appendix I) is used to get the root filehandle

Consistency issues

- In NFS v3, most operations may optionally return pre- and post-operation file attributes

- The pre-operation attributes can be used to detect whether the file was modified by another client prior to the operation
  - ▷ If so, the client can invalidate any cached data from previous operations
  - ▷ This provides an extremely weak form of consistency, while allowing caching on the client to improve performance
  - ▷ Clients that need real consistency guarantees must use an external locking protocol
  - ▷ There's something called Network Lock Manager that seems to have something to do with this

- The optional file attributes are the `wcc_data` struct

Naming issues

- All name lookups take place one path component at a time

- For example, to look up ‘foo/bar’, two transactions are required:
  - ▷ look up ‘foo’, ensure it’s a directory
  - ▷ look up ‘bar’ within ‘foo’

- This avoids all path-encoding issues in the NFS protocol

- Servers may reject particular characters in names at their whim
  - ▷ For example, UNIX server won’t allow ‘/’

- Unix pathnames only used in MOUNT protocol

NFS v2 Server Procedures

- NULL
- GETATTR
- SETATTR
- LOOKUP
- READLINK
- READ
- WRITE
- CREATE

- REMOVE
- RENAME
- LINK
- SYMLINK
- MKDIR
- RMDIR
- READDIR
- STATFS
- ROOT – never described/implemented

- WRITECACHE – never described/implemented

NULL

- Does nothing; available in all RPC services for “server response testing and timing”.

GETATTR — Get file attributes

- Parameters: filehandle

- Returns
  - ▷ On error, the error code
  - ▷ On success, the file attributes struct (`fattr3`); basically, the same info as would be returned by a stat() call

SETATTR – Set file attributes

- Parameters: filehandle, new file attributes, optional guard object (specifying time of last attribute modification)

- Returns
  - ▷ On error, the error code and optional file attributes (`wcc_data`)
  - ▷ On success, optional file attributes (`wcc_data`)

- Not guaranteed atomic

LOOKUP — Look up file in directory

- Parameters: directory filehandle, name of file or directory

- Returns:
  - ▷ On error, error code and optional directory attributes
  - ▷ On success, the filehandle corresponding to the requested object, and optionally the pre- and post- operation attributes of the directory and/or requested object

- Note: server will not allow a lookup to cross a mount point into a new filesystem

READLINK — Read symbolic link data

- Parameters: filehandle

- Returns:
  - ▷ On error, the error code and optional attributes
  - ▷ On success, optional file attributes and symlink data

- Symlink data is not interpreted by the server, only by the client

READ

- Parameters: filehandle, offset, count, totalcount (unused, removed in v3)

- Returns:
  - ▷ On error, the error code and optional attributes
  - ▷ attributes of file on complete of read
  - ▷ data: data read from the file

- Note: if the server returns fewer than `count` bytes of data, the client assumes the last byte of data is the end of the file.

- v3 adds a more precise indication of the end of file

WRITE

- Parameters
  - ▷ filehandle
  - ▷ beginoffset (unused; eliminated in v3)
  - ▷ offset into the file, 0 is the beginning
  - ▷ totalcount (unused; eliminated in v3)
  - ▷ opaque data to be written to file
- Returns:
  - ▷ status: NFS_OK if success, else error code
  - ▷ attributes of the file after the write
- Note: The server must write all the data, or return an error. The server must also write the data to "stable storage", though this definition isn't exactly clear.

CREATE

- Parameters: filehandle, filename, attributes

- Returns:
  - ▷ NFS_OK or else an error
  - ▷ filehandle of the newly created file
  - ▷ attributes of the newly created file

- Does not support “exclusive create” semantics

- You create the file if it doesn’t exist; else you get an error

REMOVE

- Removes a file, or a directory depending upon the implementation
- One process cannot remove a file from under another process
- This leads to the Last Close Problem for our stateless NFS server:
  - ▷ A client can open a file, then remove the file by its name
  - ▷ Now the file has no name, but is still open
  - ▷ Thus the client can continue to read/write the file
  - ▷ A stateless server cannot know when to perform the “last close” of this file and remove it
  - ▷ To approximate this, a client can keep refcounts of its open files
  - ▷ If refcount > 1, don’t REMOVE, but rather RENAME the file
  - ▷ This is where all the .nfsXXXX files that won’t go away come from

- What if another client has the file open? No general solution given.

RENAME

- Renames a file
- Atomic to the client
- Thus, it cannot fail in a way that leaves the directory partially renamed or otherwise inconsistent

MKDIR, RMDIR

- MKDIR, RMDIR make and remove directories

READDIR

- Parameters:
  - ▷ filehandle
  - ▷ cookie: set to 0 on first read, cookie returned by sever on subsequent reads
  - ▷ count: maximum size of results, including XDR overhead
- Returns:
  - ▷ error code if something went wrong
  - ▷ entries: List of directory entries, each consisting of:
    - ◦ attribute of the direcotry
    - ◦ name of the directory
    - ◦ cookie: opaque identifier to the next entry for subsequent READDIR calls

- rename and unlinks can invalidate cookies, causing subsequent READDIRs to miss or repeat entries

- v3 adds a cookie verifier to detect stale cookies

STATFS

- Parameter: filehandle
- Returns:
  - ▷ NFS_OK or else an error
  - ▷ tsize: optimum transfer size the server would like in data portion of READs and WRITEs
  - ▷ bsize: blocksize of filesystem
  - ▷ blocks: total number of bsize blocks on filesystem
  - ▷ bfree: number of free bsize blocks
  - ▷ bavail: bsize num of blocks available to non-priviledged users
- Not all servers support all of these attributes.

ACCESS — Check access permission

- Parameters: filehandle, set of permissions to check

- Returns:
  - ▷ On error, the error code and optional file attributes
  - ▷ On success, optional file attributes, and the set of permissions granted to client

- Added in NFS v3 to check access permissions prior to performing an operation
  - ▷ To more closely emulate Unix semantics on client, where access permissions are checked when the file is opened
  - ▷ NFS has no file open, since the server's state is soft!

- Permissions may be revoked or granted at any time

Other cool changes in v3

- READ returns a boolean to detect end of file correctly
- WRITE can now write less data than requested, and can return an indictor of the level of cache synchronization required by the client (whatever that means)
- READDIR can now validate cookies
- READDIRPLUS extends functionality of READDIR by returning filenames and fileids, along with filehandles and attributes.
- more stuff I don't have time for

Outline

- Introduction
- Protocol
- Conclusions

Conclusions

- NFS performance is closely tied to RPC performance

- Both perform best on fast LANs, no surprise

Sources

- RFC 1813 (NFS version 3), RFC 1057 (RPC), RFC 1014 (XDR)
- Sandberg, et. al., Design and Implementation of the Sun Network Filesystem, in *USENIX Conference Proceedings*, Summer 1985.
- Callaghan, Brent, NFS Illustrated. *Addison Wesley Professional Computing Series* , 2000.