

NFS v4 and HPSS

Philippe DENIEL
CEA/DAM
philippe.deniel@cea.fr



The scope of this presentation

- Introducing the NFS v4 protocol, focusing on the differences with the former version of the protocol (v2 and v3), and its interaction with HPSS
- Two aspects will be seen
 - 1st : Possible improvements with NFSv4 in terms of
 - Security
 - Reliability
 - Scalability
 - Interoperability
 - 2nd: NFSv4 interface to HPSS

PART 1: Let's talk about the protocol

NFS v4 is an IETF protocol

- NFS v2 was known for being a "home made" protocol, designed by Sun Microsystems, in 1984
- NFS v3 was a little more discussed and several companies took part in the design of the protocol
- NFSv4 is the result of an IETF working group, like TCP, UDP or IPv6. Design process started in 1997, and ends with edition of RFC3530 in late 2003.




NFSv4 key features



- NFSv4 is designed to emphasize the following core features:
 - Improved access and good performances on the Internet
 - Strong security, negotiation built within the protocol
 - Cross platform interoperability
 - Ready for protocol extensions

A more integrated protocol



- NFS v4 is defined by RFC3530
 - NFSv4 is a standalone protocol; it requires no ancillary protocol
 - Mount protocol, NLM, NFS Stat are no more needed
 - Port 2049 is the only resource required by NFSv4
 - This value is written explicitly in the RFC
-  NFS v4 is firewall friendly
- NFSv4 is not bound to a Unix semantic
 - File system objects's attributes are shown as self-described bitmaps, not as a unix-like structure



NFS v4 is not bound to Unix

Designed for the Internet

- NFSv4 is design to work on high latency / low bandwidth network



NFSv2 and v3

- latency ~1ms
- rate ~MB/sec
- Distance ~ 100 meters
- Designed for LAN

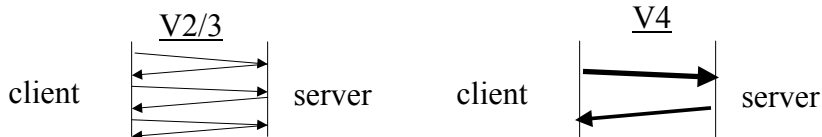


NFSv4

- latency ~100ms
- rate ~KB/sec
- Distance ~ 1000 km
- Designed for WAN

Minimizing client/server dialog

- Use of multi component LOOKUP: LOOKUP(a/b/c) instead of LOOKUP(a) + LOOKUP(b) + LOOKUP(c)
- Classical POSIX functions on files requires several nfs v2 or v3 requests:
 - example :
 - open(file) = LOOKUP(file) + ACCESS(file) + GETATTR(parent)
- NFSv4 does many operations in one request by making COMPOUND requests
 - A compound in a "todolist" made of operations to be done on the server handside:
 - Example:
 - Mkdir = Compound[
 - » ACCESS,
 - » GETATTR
 - » CREATE(dirobject)]



Aggressive caching



- Cache coherency management oriented operations
 - OP_VERIFY
 - OP_NVERIFY
 - partial OP_GETATTR (getting FH, or ctime/mtime/atime)
- Use of leases based file delegations to allow client to aggressively cache data
 - Client deals with the file on his own, locally
 - No other client access the file for the duration of the lease
 - Server perform callbacks to get the file state
- NFSv4 could be very efficient in a large scale through a proxy caching
 - Proxies with policies similar to HTTP proxies can cache files accessed by a pack of clients

NFSv4 is security oriented



- NFSv4 is firewall friendly
 - Only ports 2049/tcp and 2049/udp will be use; no other port is required
- NFSv4 is ONC/RPC based
 - RPCSEC_GSS is explicitly supported (negociation can be made with the help of OP_SECINFO)
 - Every security paradigm with a GSSAPI integration can be used with RPCSEC_GSS : krb5, LIPKEY, SPKM3
- NFSv4 is connection oriented
 - Connection based security (like SSL) is possible
 - RFC3530 recommends not to use SSL, but to use LIPKEY via GSSAPI and RPCSECGSS instead



Cross-Platform interoperability (1)



- Files attributes are shown as bitmaps



- Semantic is not necessary a Unix-like system
- NFSv4 can export filesystems with reduced attributes (like PC FAT)
- NFSv4 can export filesystems with extended attributes and make them visible on the client handside
- User and Group are shown as strings, not as UID/GID (unix semantic)



- Access Control Lists are natively supported
 - ACL model suits the need of diverse ACLs models
 - Unix/POSIX style
 - NT style



Cross-Platform interoperability (2)



- NFSv4 supports Windows2000's share reservation and OPEN/CLOSE semantic.
 - Compliancy with Windows based architecture
- File Locking is managed
 - Lock management on top of a lease based mechanism
- Internationalization is supported (accentuated characters, cyrillic, hebrew, kanjis...) with use of UTF8 strings
- Protocol is extensible: protocol implementation includes minor versioning



Part 2: And what about HPSS ?

What is to be supported ? (1)



- NFSv4 is a relatively young protocol
 - The (experimental) client supports a subset of the RFC3530 features
 - More features will be available as the vendors will begin to provide NFSv4 support
- NFSv4 is a very complete protocol, but some functionality could be not implemented (and NFS4ERR_NOTSUPPORTED is returned)
 - HPSS has no lock similar to fcntl(2).
 - Lock management will not be supported first
- There are very few NFSv4 client today. They do not support delegation for the moment
 - Delegation will be implemented in a second step, when client with this capability will be available

What is to be supported ? (2)



- NFSv4 can make many things manageable by clients
 - ACLs (both Unix and NT)
 - File's Creation Time
 - Files extended attributes, as NFSv4 named Attributes
 - Class Of Service
 - Fileset Name
 - site local attribute

Possible extension to the hpss_nfs daemon



- NFSv4 could be use for small file management
 - Small files could be identify by a specific extended attributes
 - Server datacache layer pack the small file together to produce a larger one, stored in HPSS
 - Small files are accessed from within the big file
 - Small files within a "Big File" are accessed through volatile filehandle one the big file is staged and accessible through nfs v4
- Use of NFSv4 proxies for cross-site exports

Extension to be made to hpss_nfs daemon



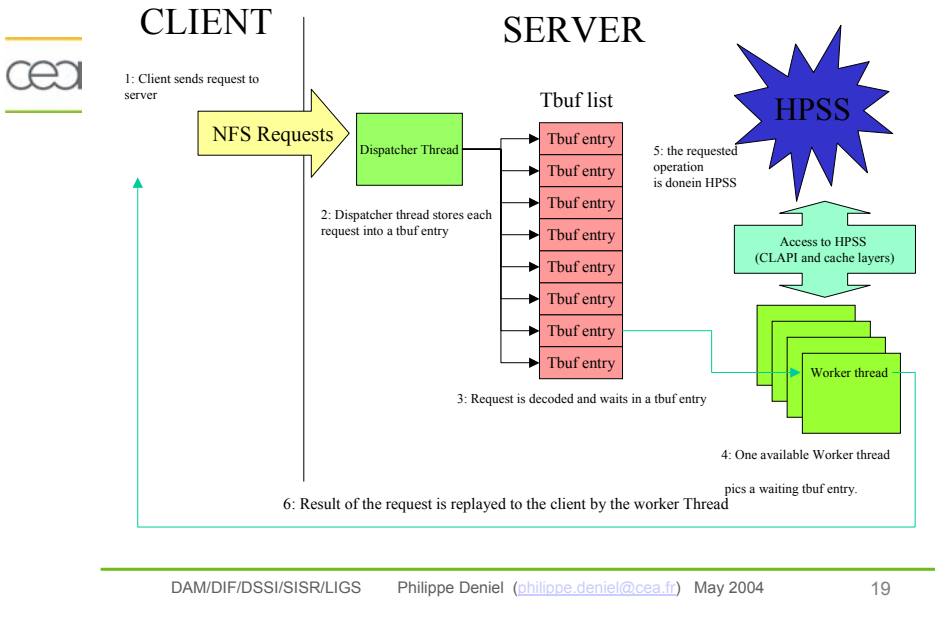
- Daemon MUST support the ONC/RPC on top of TCP/IP
- Daemon MUST implement RPCSEC_GSS (and use it via NFSv4 or NFS v2 and v3 with implementation described in RFC2623)
- Daemon will manage NFSv2 and v3 with specific threads, another pool of threads will manage NFSv4
 - Requests behavior are similar in v2 and v3, but quite different in v4 (COMPOUND request, delegation)
 - Avoid V4 to starve V2 and V3 management
 - Avoid V2 and V3 to slow down V4
- Some changes to be made to hpss_nfs internal cache layer to take care of NFSv4 caching capabilities
 - The capability of explicitly remove an entry from the internal cache is required (delegation management)
 - Making the cache layer more "lease management aware"

What is ready now ?

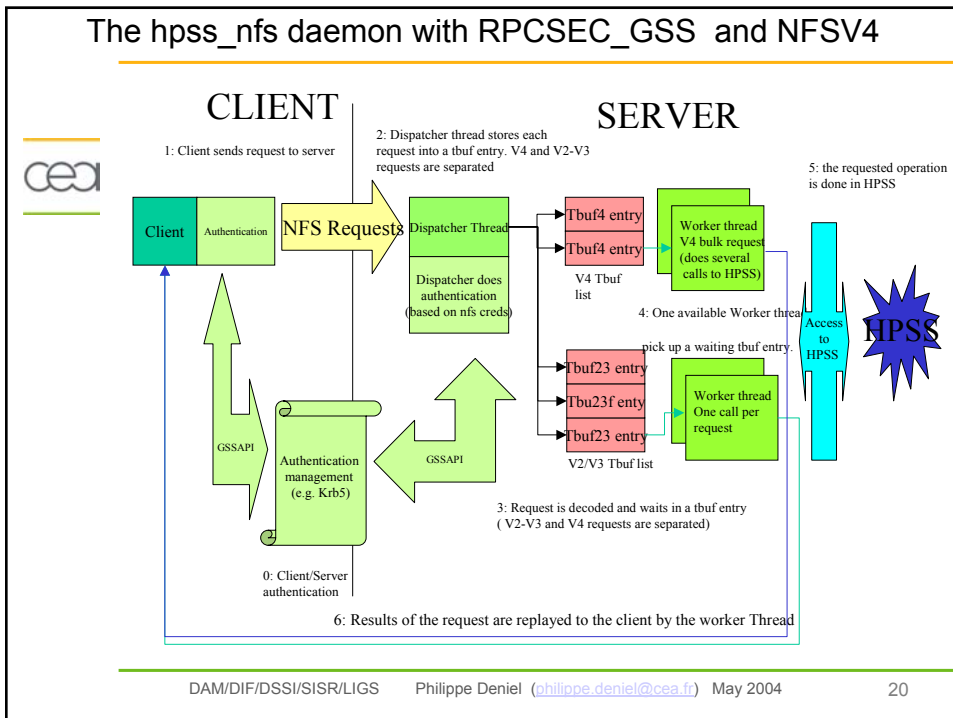


- NFSv4 client/server from CITI was installed on several linux boxes
 - Client/server interaction in NFSv4 with different security flavour was validated
 - Interaction with NetApp filer was validated
- NFSv4 RPC layer design was added to HPSS/NFS daemon design paper
- NFSv4 RPC layer was implemented to validate correct implementation of the RFC
- Basic Connectathon for nfsv3 goes ok on a nfsv4 client on a Linux box in front of a hpss_nfs4 beta (including the RPC layer for v4)
- NFSv4 specific stuff is to be added (like delegation)
 - Linux client is very basic (emulates v3 with V4 compound requests)
 - Miscellaneous RPCSEC_GSS stuff (krb5 cross-platform test, SPKM3, LIPKEY)
 - Waiting for client to support this feature to continue

The hpss_nfs daemon now



The hpss_nfs daemon with RPCSEC_GSS and NFSV4



Questions ?

cea

