Introduction to NFS v4 and pNFS

David Black, SNIA Technical Council, EMC

SNIA

slides by Alan Yoder, NetApp
with thanks to Michael Eisler
and Brent Welch

NFS v4.0

- Under development from 1998-2005
  - primarily driven by Sun, Netapp, Hummingbird
  - some University involvement (CITI UMich, CMU)
  - systems beginning to ship
    - available in Linux

NFS v4.0

- Mandates strong security be available
  - Every NFSv4 implementation has Kerberos V5
  - You can use weak authentication if you want
- Easier to deploy across firewalls (only one port is used)
- Finer grained access control
  - Goes beyond UNIX owner, group, mode
  - Uses a Windows-like ACL
- Read-only, read-mostly, or single writer workloads can benefit from formal caching extensions (delegations)
- Multi-protocol (NFS, CIFS) access experience is cleaner
- Byte range locking protocol is much more robust
  - Recovery algorithms are simpler, hence more reliable
  - Not a separate protocol as in V3

NFS v3 and v4 compared

NFSv3

- A collection of protocols (file access, mount, lock, status)
- Stateless
- UNIX-centric, but seen in Windows too
- Deployed with weak authentication
- 32 bit numeric uids/gids
- Ad-hoc caching
- UNIX permissions
- Works over UDP,TCP
- Needs a-priori agreement on character sets

NFSv4

- One protocol to a single port (2049)
- Lease-based state
- Supports UNIX and Windows file semantics
- Mandates strong authentication
- String-based identities
- Real caching handshake
- Windows-like access
- Bans UDP
- Uses a universal character set for file names

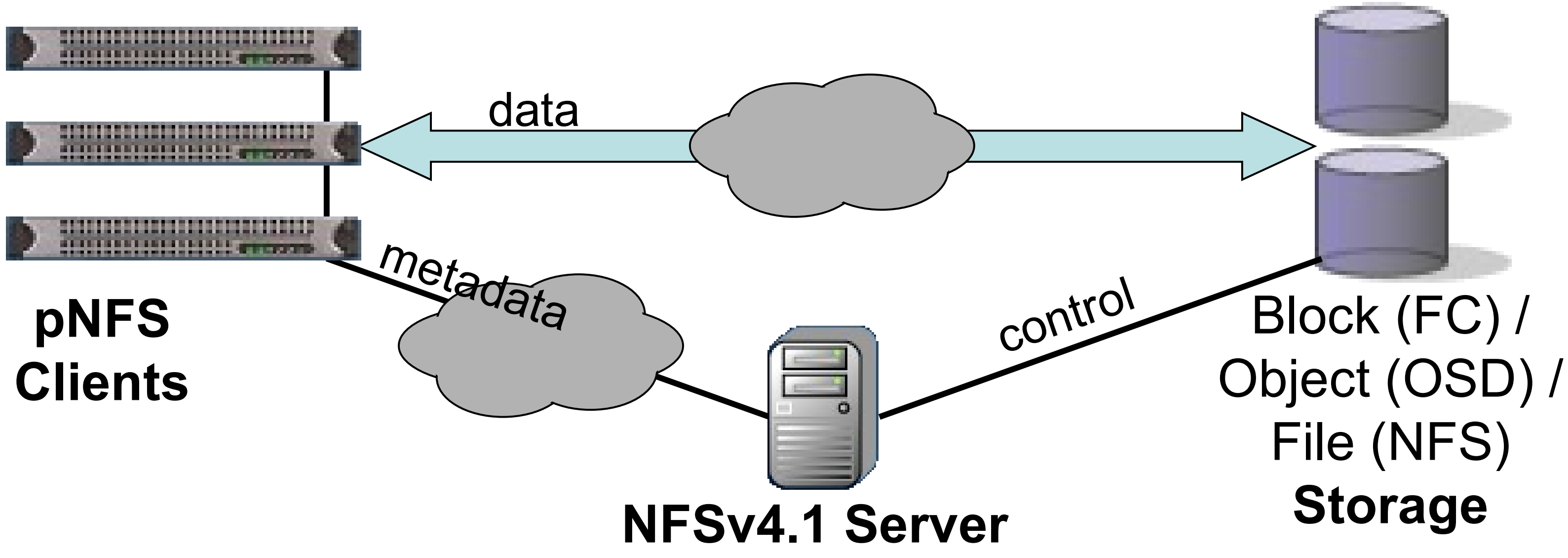
pNFS history

- Idea to use SAN FS architecture for NFS originally from Gary Grider (LANL) and Lee Ward (Sandia)
- Development driven by Panasas, Netapp, Sun, EMC, IBM, UMich/CITI
- Folded into NFSv4 minor version NFSv4.1 in 2006

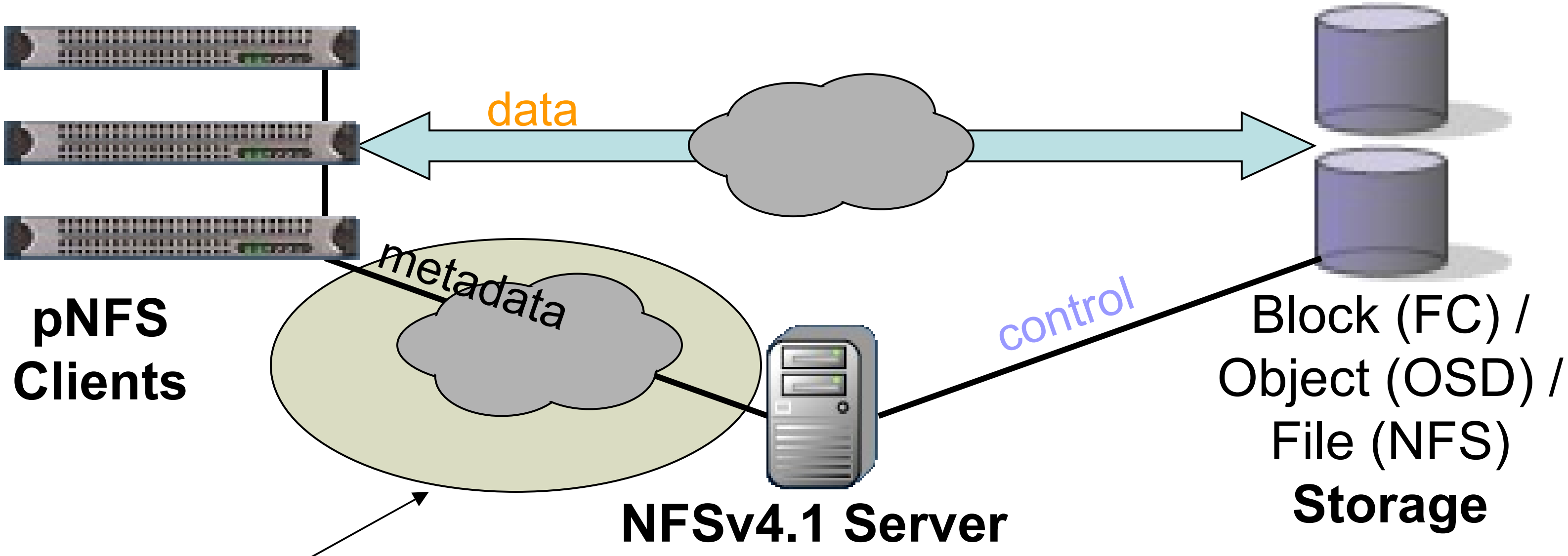
pNFS

- Essentially makes clients aware of how a clustered filesystem stripes files
- Files accessible via pNFS can be accessed via non-parallel NFS (and in the case of filers, CIFS, and other file access protocols)
- Benefits workloads with
  - many small files
  - very large files
- Three supported methods of access to data:
  - Blocks (FC, iSCSI)
  - Objects (OSD)
  - Files (NFSv4.1)

pNFS architecture



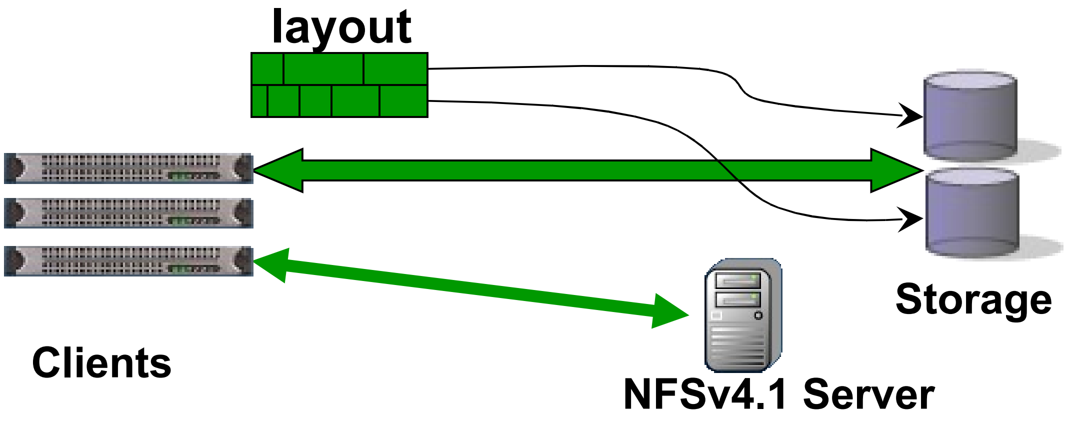
pNFS architecture



- Only this is covered by the pNFS protocol
- Client-to-storage data path and server-to-storage control path are specified elsewhere, e.g.
  - SCSI **Block** Commands (**SBC**) over Fibre Channel (**FC**)
  - SCSI **Object**-based Storage Device (**OSD**) over iSCSI
  - Network **File** System (**NFS**)

pNFS basic operation

- Client gets a layout from the NFS Server
- The layout maps the file onto storage devices and addresses
- The client uses the layout to perform direct I/O to storage
- At any time the server can recall the layout
- Client commits changes and returns the layout when it's done
- pNFS is optional, the client can always use regular NFSv4 I/O



pNFS protocol operations

- LAYOUTGET
  - (filehandle, type, byte range) -> type-specific layout
- LAYOUTRETURN
  - (filehandle, range) -> server can release state about the client
- LAYOUTCOMMIT
  - (filehandle, byte range, updated attributes, layout-specific info) -> server ensures that data is visible to other clients
  - Timestamps and end-of-file attributes are updated
- GETDEVICEINFO, GETDEVICELIST
  - Map deviceID in layout to type-specific addressing information

pNFS protocol callbacks

- CB_LAYOUTRECALL
  - Server tells the client to stop using a layout
- CB_RECALLABLE_OBJ_AVAIL
  - Delegation available for a file that was not previously available

pNFS read

◆ Client: LOOKUP+OPEN NFS Server: returns file handle and state ids ◆ Client: LAYOUTGET NFS Server: returns layout	control path
◆ Client: many parallel READs to storage devices Storage devices: return data	data path
◆ Client: LAYOUTRETURN NFS server: ack	control path

- Layouts are cacheable for multiple LOOKUP+OPEN instances
- Server uses CB_LAYOUTRECALL when the layout is no longer valid

pNFS write

- Client: LOOKUP+OPEN NFS Server: returns file handle and state ids - Client: LAYOUTGET NFS Server: returns layout	control path
- Client: many parallel WRITEs to storage devices Storage devices: ack	data path
- Client: LAYOUTCOMMIT NFS server: "publishes" write - Client: LAYOUTRETURN NFS server: ack	control path

- Server may restrict byte range of write layout to reduce allocation overheads, avoid quota limits, etc.

What pNFS doesn't give you

- Improved cache consistency
  - NFS has open-to-close consistency
- Perfect POSIX semantics in a distributed file system
- Clustered metadata
  - Though a mechanism for this is not precluded

All good projects for OGF! ☺