



pNFS

Frank Batschulat
Solaris Sustaining Engineering
MTS, Sun Microsystems



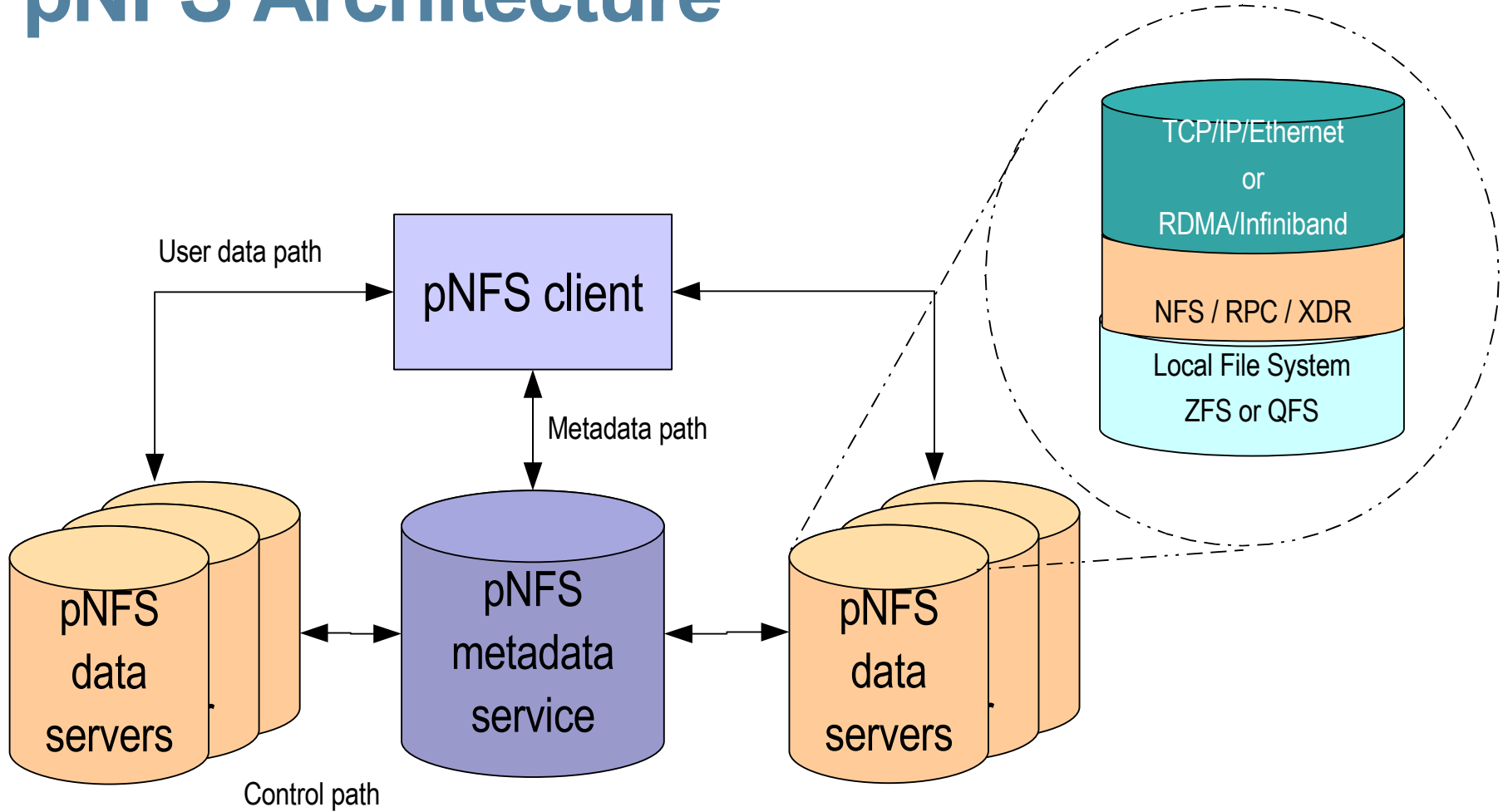
What is pNFS?

- Parallel extensions to NFSv4 to improve bandwidth
 - > Designed by HPTC community
 - > Adopted by IETF NFS WG as part of NFSv4.1
 - Draft standard expected consensus end 08 (presently draft 23)
- Major features
 - > Parallel transfer from multiple servers to single client
 - > Global name space
 - > Horizontal scale of data storage
 - > Exists within mental model of existing NFS community
 - > Three types of implementation: *files*, objects, blocks
 - All offer *identical* POSIX file semantics to client

Architecture

- “Classical” cluster file system architecture
 - > Metadata is handled by centralized server (MDS)
 - > User data is distributed to many data servers (DS)
- Clients open files by asking the MDS (new LAYOUTGET)
 - > MDS verifies permissions, sends back a *layout*
 - > Layout describes how file data is organized and where
 - > Standard supports striped RAID-0 layouts and replicated data
 - Striping done at the file level
 - Files have custom organizations, even in same directory
 - /pnfs/file1 might be 5-way striped on s1, s99, s4, s12, s04
 - /pnfs/file2 might be a singleton file on s4
- Clients are given ability to access the data servers directly
- The data can be striped across many data servers

pNFS Architecture



Architecture

- pNFS is *optional* for NFSv4.1 clients
 - > MDS must be prepared to proxy IO from DS to client if client cannot handle layouts
 - > Not expected to be common
 - > However, this required capability may prove “useful”
- Data on all DS shares – global – name space
 - > Could be hundreds or thousands of data servers
- LAYOUTGET returns a layout that describes the striping pattern for a given file
- layouts are recallable which allows pNFS servers to re-stripe a file if desired or necessary

pNFS Storage Device Types

- pNFS supports multiple Storage Device types (aka layout types)
- A layout can stripe a file over just one type of device
- The NFSv4 working group currently specifies three types:
 - Files [storage “device” is an NFSv4.1 server]
 - Blocks [storage device is an iSCSI or FC target]
 - Objects [storage device is an Object Storage Device (OSD)]
- Additional types require a standards-track specification
- If a client does not support a given device type it can issue I/O directly to MDS

Architecture and Implementation

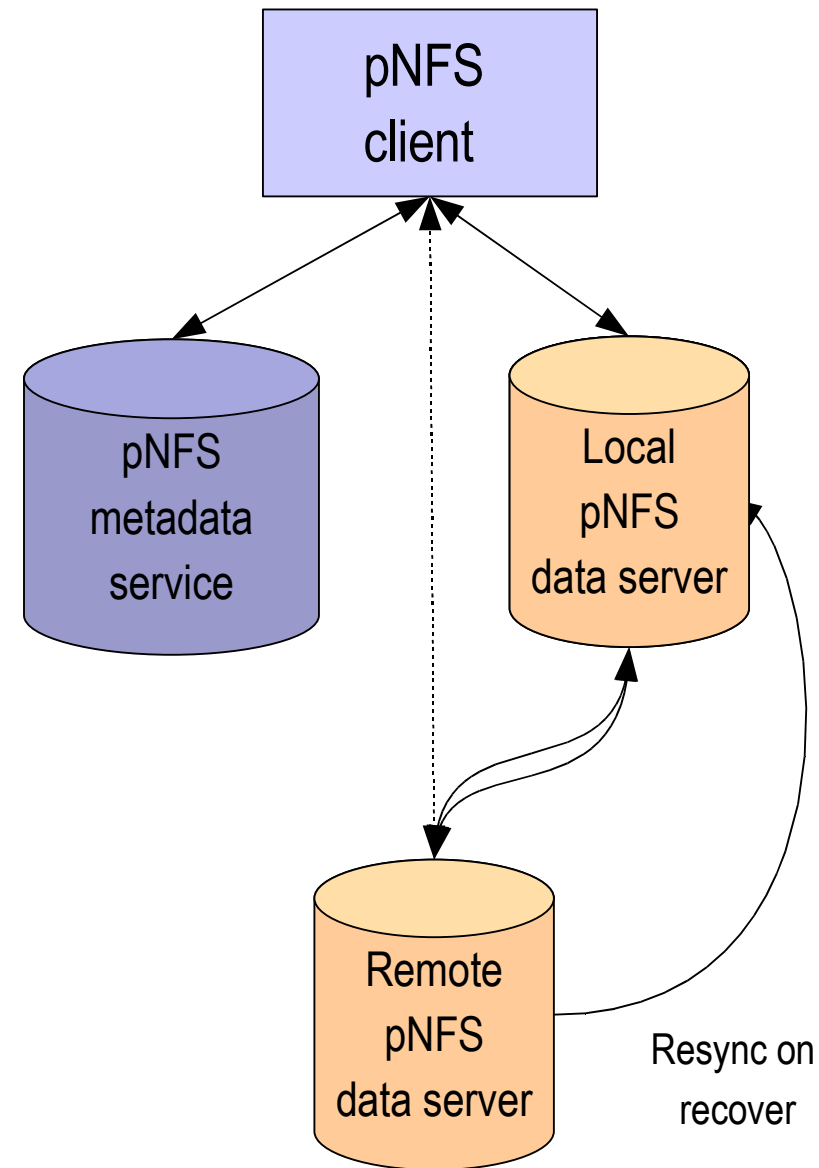
- MDS must approve all data access
 - > Clients must trust the server, but need *not* trust clients
- Clients access data directly from DS
- Solaris DS stores (sub-)file data in ZFS DMU
 - > Data objects are stored as DMU objects
 - > ZPL is bypassed
 - Not normally visible to logged-in users
 - Avoids POSIX overhead
 - Future option to use QFS object pool DS
- Solaris MDS uses ZFS (ZPL) for namespace and attribute storage (eg “system” attributes used for pNFS layouts)

How it works...

- A “Layout” manages the location of file data
- A client asks for a “Layout” when it wants to access the data
- The “Layout” contains a list of deviceids (shorthand references for the storage devices) and striping information
- The granting of a “Layout” gives the client the permission to access the data servers directly

Replicated Layouts

- File-level replication
- Server-side only – like remote copy (SNDR)
- Transparent failover
 - > MDS detects DS loss, *recalls* layout
 - > Client returns layout, requests new layout
 - > MDS returns layout pointing to replica
 - > Can happen during access assuming replica is consistent



Design Points

- A single Solaris pNFS community (eg a single mount point) will handle:
 - > ~ 50 billion files
 - > Hundreds or thousands of DS
 - > Files of essentially unlimited size
 - > No practicable upper limit on community-wide bandwidth
- Communities will be able to cooperate for HA purposes
 - > Active/active design (not 1.0)
 - > Further considering how to handle geographically disbursed data using same core mechanisms

PNFS sweet spots

- High Performance
 - > Provide scalable parallel access to files distributed among multiple data servers
- Horizontal Scalability
 - > Reduce the resource limitation of the single NFS server serving all of the files in an exported file system
- Separation of Data and Metadata
 - > easy to move data around
 - > dynamically notify client of data location
- Kerberos support on metadata, data and control paths

pNFS Implementation Status

- Code is in advanced prototype stage
- Source and BFU archives released to OpenSolaris
- Already demonstrated interoperability with several other implementations – including Linux, NetApp
- Estimated put back into OpenSolaris mid-'08

Practical Implications

- pNFS + NFS-over-RDMA = FAST bulk data transfer
 - > Parallel transfer across multiple servers
 - > *Each* x2100m1 server can deliver ~1 GB/sec
 - > A large client can transfer near bridge bandwidth
 - Expect x4600 client to move ~2.5 GB/sec to/from servers
- DB-on-NFS
 - > OLTP on NFS is practical today (S9U5 and later)
 - > pNFS + RDMA makes DSS or DSS + OLTP practical
- Horizontal scale of DS eliminates hot spots on DS
 - > Benefits apps without high bandwidth requirements

Links

- Demo:
 - > <http://opensolaris.org/os/project/nfsv41/pnfsdemos/basics>
- OpenSolaris community page:
 - > <http://opensolaris.org/os/project/nfsv41/>
 - > <http://wikis.sun.com/display/NFS/Home>
- Current IETF draft standard (draft 23):
 - > <http://www.nfsv4-editor.org/draft-23/draft-ietf-nfsv4-minor>
- Linux:
 - > <http://www.citi.umich.edu/projects/nfsv4/pnfs/block/>
 - > https://wiki.linux-nfs.org/wiki/index.php/PNFS_prototype_



Thank You!

frankB

frank.batschulat@sun.com