SNIA Education

Practical Steps to Implementing pNFS and NFSv4.1

Alex McDonald, NetApp

SNIA Legal Notice

- The material contained in this tutorial is copyrighted by the SNIA unless otherwise noted.
- Member companies and individual members may use this material in presentations and literature under the following conditions:
  - Any slide or slides used must be reproduced in their entirety without modification
  - The SNIA must be acknowledged as the source of any material used in the body of any document containing material from these presentations.
- This presentation is a project of the SNIA Education Committee.
- Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be, or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.
- The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

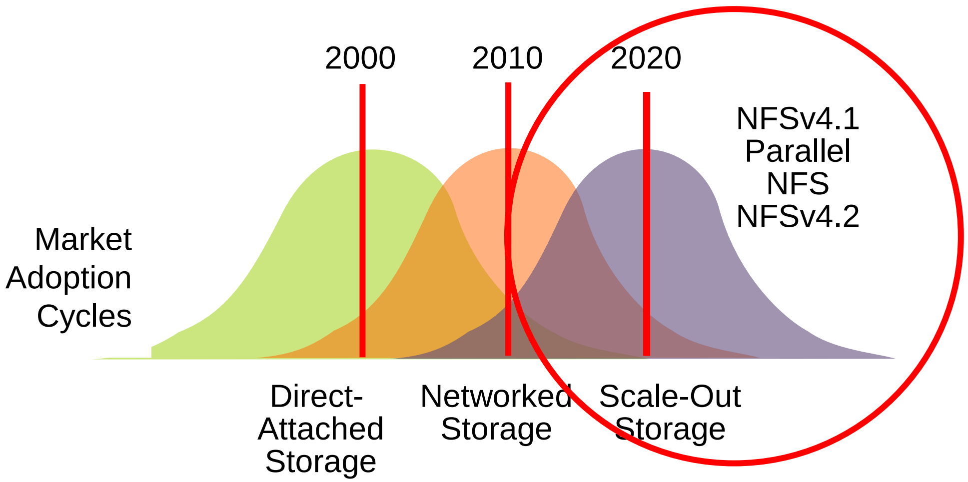
  NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.

Abstract

- **Practical Steps to Implementing pNFS and NFSv4.1**
  - Much has been written about pNFS (parallelized NFS) and NFSv41, the latest NFS protocol. But practical examples of how to implement NFSv4.1 and pNFS are fragmentary and incomplete. This presentation will take a step-by-step guide to implementation, with a focus on file systems. From client and server selection and preparation, the tutorial will cover key auxiliary protocols like DNS, LDAP and Kerberos.
  - Learning Objectives
    - An overview of the practical steps required to implement pNFS and NFSv4.1
    - Detailed information on the selection of software components to ensure a suitable environment
    - Show how these parts are engineered and delivered as a solution

NFS; Ubiquitous & Everywhere

- NFS is ubiquitous and everywhere
- NFSv3 very successful
  - Protocol adoption is over time, and there have been no big incentives to change
- Industry – and hence NFS – doesn't stand still
  - NFSv2 in 1983
  - NFSv3 in 1995
  - NFSv4 in 2003
  - NFSv4.1 in 2010
  - NFSv4.2 to be agreed at IETF shortly
  - Faster pace for minor revisions
- NFSv3 very successful
  - Protocol adoption is over time, and there have been no big incentives to change



Evolving Requirements

- Adoption has been slow; why?
  - Lack of clients was a problem with NFSv4
  - NFSv3 was just “good enough”
- Industry is changing, as are requirements
  - Economic Trends
    - Cheap and fast computing clusters
    - Cheap and fast network (1GbE to 10GbE, 40GbE and 100GbE in the datacenter)
    - Cost effective & performant storage based on Flash & SATA
  - Performance
    - Exposes NFSv3 single threaded bottlenecks in applications
    - Increased demands of compute parallelism and consequent data parallelism
    - Analysis begets more data, at exponential rates
    - Competitive edge (ops/sec)
  - Business requirement to reduce solution times
    - Beyond performance; NFSv4.1 brings increased scale & flexibility
    - Outside of the datacenter; requires good security, scalability

NFSv4 and beyond

- Areas address by NFSv4, NFSv4.1 and pNFS
  - Security
  - Uniform namespaces
  - Statefulness & Sessions
  - Compound operations
  - Caching; Directory & File Delegations
  - Parallelisation; Layouts & pNFS
- Future NFSv4.2 and FedFS (in addendum slides)
  - New features in NFSv4.2
  - FedFS: Global namespace; IESG has approved Dec 2012

Agenda

- We'll cover
  - Selecting the application for NFSv4.1
  - Planning;
    - Filenames and namespace considerations
    - Firewalls
    - Understanding statefulness
    - Security
  - Server & Client Availability
  - Where Next
    - Considering pNFS
- This is a high level overview
  - Use SNIA white papers and vendors (both client & server) to help you implement

Selecting the Parts

- 1 – An NFSv4.1 compliant server
  - Question; files, blocks or objects?
- 2 – An NSFv4.1 compliant client
  - Will almost certainly be *nix based; no native NFS4 Windows client
  - Some applications are their own clients; Oracle, VMware etc
- 3 – Auxiliary tools;
  - Kerberos, DNS, NTP, LDAP
- 4 – If you can; use NFSv4.1 over NFSv4

Selecting an Application

- First task; select an application or storage infrastructure for NFSv4.1 use
  - Home directories
  - HPC applications
- Don’t select…
  - Oracle; use dNFS built in to the Oracle kernel
  - VMware & other virtualization tools; no support for anything other than NFSv3 as of this date
  - “Oddball” applications that expect to be able to internally manage NFSv3 “maps” with multiple mount points, or auxiliary protocols like mountd, statd etc;
  - Any application that requires UDP; NFSv4 doesn’t support anything except TCP

NFSv4 Stateful Clients

- NFSv4 gives client independence
  - Previous model had “dumb” stateless client; server had the smarts
- Allows delegations & caching
- No automounter required, simplified locking
  - Mounting & locking incorporated into the protocol
  - Simplifies administration
- Why?
  - Compute nodes work best with local data
  - NFSv4 eliminates the need for local storage
  - Exposes more of the backend storage functionality
    - Client can help make server smarter by providing hints
  - Removes major source of NFSv3 irritation; stale locks

NFSv4.1 Delegations

- Server delegates certain responsibilities to the client
  - Directory & file
- At OPEN, the server can provide
  - READ delegation; server guarantees no writers
  - WRITE delegation; server guarantees exclusive access
- Allows client to locally service operations
  - E.g OPEN, CLOSE, LOCK, LOCKU, READ, WRITE

NFSv4.1 Sessions

- NFSv3 server never knows if client got reply message
- NFSv4.1 introduces Sessions
  - Major protocol infrastructure change
  - Exactly Once Semantics (EOS)
  - Bounded size of reply cache
  - Unlimited parallelism
- A session maintains the server's state relative to the connections belonging to a client
- Action
  - None; use delegation & caching transparently; client & server provide transparency
  - NFSv4 advantages include session lock clean up automatically

NFSv4 Compound Operations

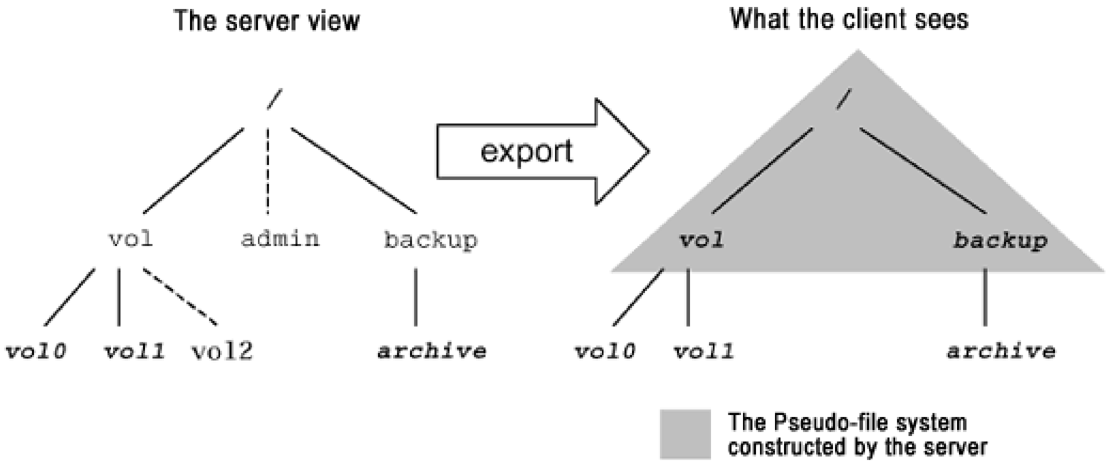
- NFSv3 protocol can be “chatty”; unsuitable for WANs with poor latency
- Typical NFSv3; open, read & close a file
  - LOOKUP, GETATTR, OPEN, READ, SETATTR, CLOSE
- NFSv4 compounds into a single operation
  - Reduce wire time
  - Simple error recovery

NFSv3 Operation	SPECsfs2008
GETATTR	26%
LOOKUP	24%
READ	18%
ACCESS	11%
WRITE	10%
SETATTR	4%
READDIRPLUS	2%
READLINK	1%
READDIR	1%
CREATE	1%
REMOVE	1%
FSSTAT	1%

Table 1; SPECsfs2008 %ages for NFSv3 operations

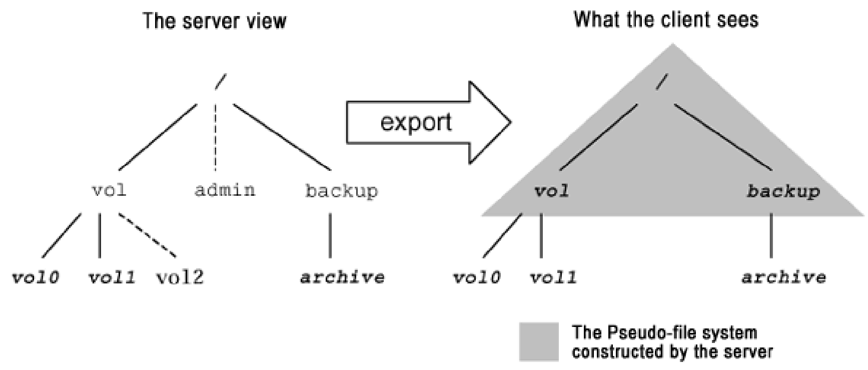
NFSv4 Namespace

- Uniform and "infinite" namespace
  - Moving from user/home directories to datacenter & corporate use
  - Meets demands for "large scale" protocol
  - Unicode support for UTF-8 codepoints
- No automounter required
  - Simplifies administration



NFSv4 Namespace

- Namespace Example
  - Server exports
    - /vol/vol0
    - /vol/vol1
    - /backup/archive



- Mount root / over NFSv3:
  - Allows the client to list the contents of vol/vol2
- Mount root / over NFSv4:
  - If /vol/vol2 has not been exported and the pseudo filesystem does not contain it; the directory is not visible
  - An explicit mount of vol/vol2 will be required

NFSv4 Namespace

- Namespaces
- Action
  - Consider using the flexibility of pseudo-filesystems to permit easier migration from NFSv3 directory structures to NFSv4, without being overly concerned as to the server directory hierarchy and layout
- However;
  - If there are applications that traverse the filesystem structure or assume the entire filesystem is visible, caution should be exercised before moving to NFSv4 to understand the impact presenting a pseudo filesystem
  - Especially when converting NFSv3 mounts of / (root) to NFSv4

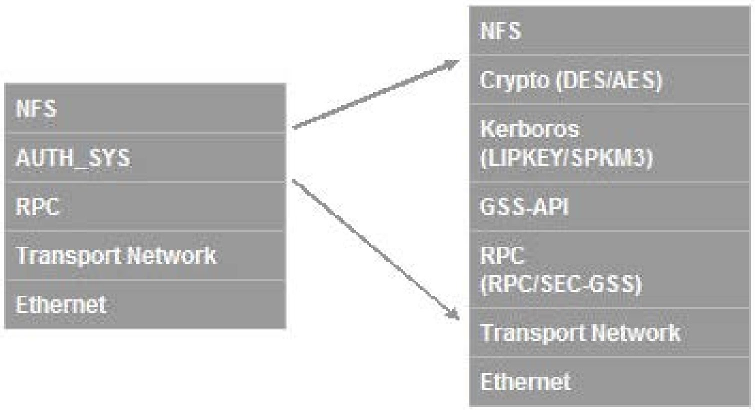
NFSv4 I18N Directory & File Names

- Directory and File Names
  - NFSv4 uses UTF-8
    - Backward compatible with 7 bit ASCII
  - Check filenames for compatibility
    - NFSv3 file created with the name René contains an 8 bit ASCII
    - UTF-8 é indicates a multibyte UTF-8 encoding, which will lead to unexpected results
- Action
  - Review existing NFSv3 names to ensure that they are 7 bit ASCII clean
  - These aren't;

¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯
° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿
À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
à á â ã ä å æ ç è é ê ë ì í î ï
ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

NFSv4 Security

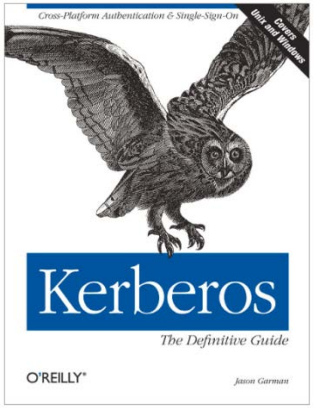
- Strong security framework
- Access control lists (ACLs) for security and Windows® compatibility
- Security with Kerberos
  - Negotiated RPC security that depends on cryptography, RPCSEC_GSS
- NFSv4 can be implemented without implementing Kerberos security
  - Not advised; but it is possible



NFSv4 Security (cont.)

- Implementing without Kerberos
  - No security is a last resort!
- NFSv3 represents users and groups via 32 bit integers
  - UIDs and GIDs with GETATTR and SETATTR
- NFSv4 represents users and groups as strings
  - user@domain or group@domain
- Requires NFSv3 UID and GID 32 bit integers be converted to all numeric strings
  - Client side;
    - Run idmapd6
    - /etc/idmapd.conf points to a default domain and specifies translation service nsswitch
  - Incorrect or incomplete configuration, UID and GID will display nobody
  - Using integers to represent users and groups requires that every client and server that might connect to each other agree on user and group assignments

NFSv4 Security (cont.)



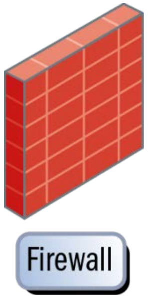
- Implementing with Kerberos
- Find a security expert
  - Requires to be correctly implemented
  - Do not use NFSv4 as a testbed to shake out Kerberos iss
- User communities divided into realms
  - Realm has an administrator responsible for maintaining a database of users
  - Correct user@domain or group@domain string is required
  - NFSv3 32 bit integer UIDs and GIDs are explicitly denied access
- NFSv3 and NFSv4 security models are not compatible
  - Although storage systems may support both NFSv3 and NFSv4 clients, be aware that there may be compatibility issues with ACLs. For example, they may be enforced but not visible to the NFSv3 client.
- Resources:
  - http://web.mit.edu/kerberos/

NFSv4 Security (cont.)

- Action
  - Review security requirements on NFSv4 filesystems
  - Use Kerberos for robust security, especially across WANs
  - If using Kerberos, ensure it is installed and operating correctly
    - Don't use NFSv4 as a testbed to shake out Kerberos issues
- Consider using Windows AD Server
  - Easy to manage environment, compatible
- Last resort
  - If using NFSv3 security, ensure UID and GUID mapping and translation is uniformly implemented across the enterprise

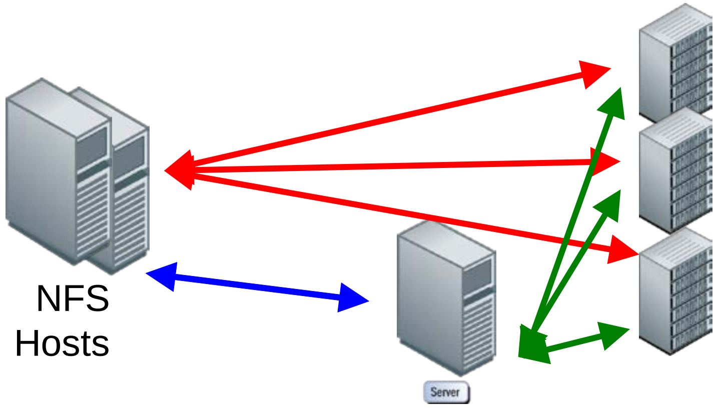
NFSv4 Security (cont.)

- Firewalls
  - NFSv3 promiscuously uses ports; including 111, 1039, 1047, 1048, and 2049 (and possibly more…)
  - NFSv4 has no “auxiliary” protocols like portmapper, statd, lockd or mountd
    - Functionality built in to the protocol
    - Uses port 2049 with TCP only
  - No floating ports required & easily supported by NAT
- Action
  - Open port 2049 for TCP on firewalls



NFSv4.1 Layouts

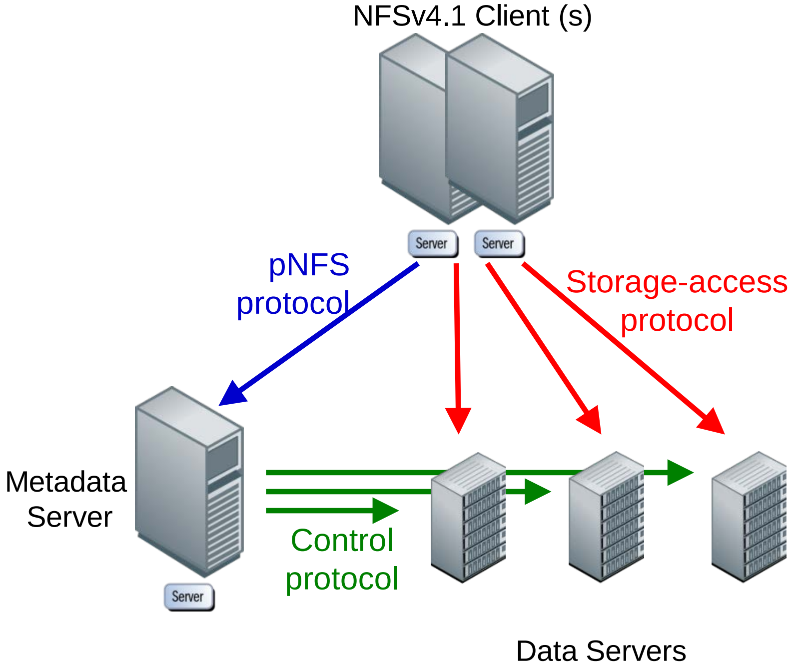
- Layouts
  - Files, objects and block layouts
  - Provides flexibility for storage that underpins it
  - Location transparent
    - Striping and clustering
- Examples
  - Blocks, Object and Files layouts all available from various vendors



pNFS

- NFSv4.1 (pNFS) can aggregate bandwidth
  - Modern approach; relieves issues associated with point-to-point connections

- pNFS Client
  - Client read/write a file
  - Server grants permission
  - File layout (stripe map) is given to the client
  - Client parallel R/W directly to data servers
- Removes IO Bottlenecks
  - No single storage node is a bottleneck
  - Improves large file performance
- Improves Management
  - Data and clients are load balanced
  - Single Namespace

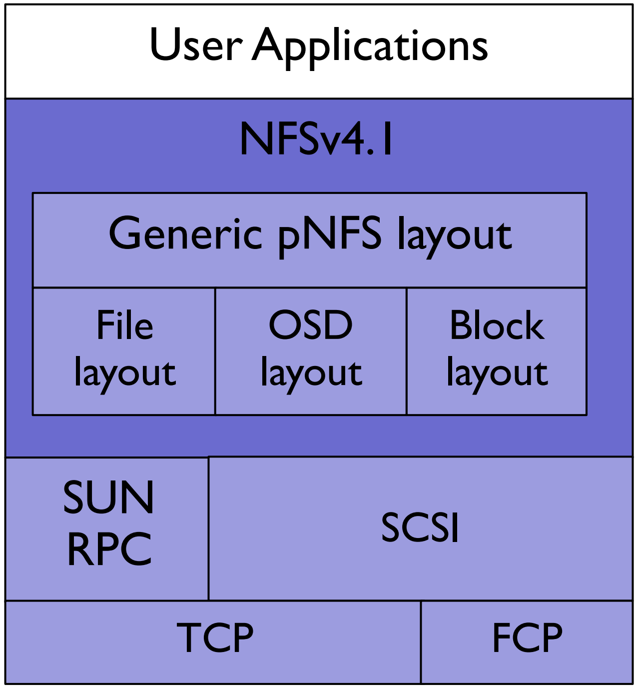


pNFS Filesystem Implications

- Files, blocks, objects can co-exist in the same storage network
  - Can access the same filesystem; even the same file
- NFS flexible enough to support unlimited number of storage layout types
  - Three IETF standards, files, blocks, objects
  - Others evaluated experimentally
- NAS vs SAN; no-one cares any more
  - IETF process defines how you get to storage, not what your storage looks like
  - Each vendor's pNFS implementation is different… but they all meet the standard

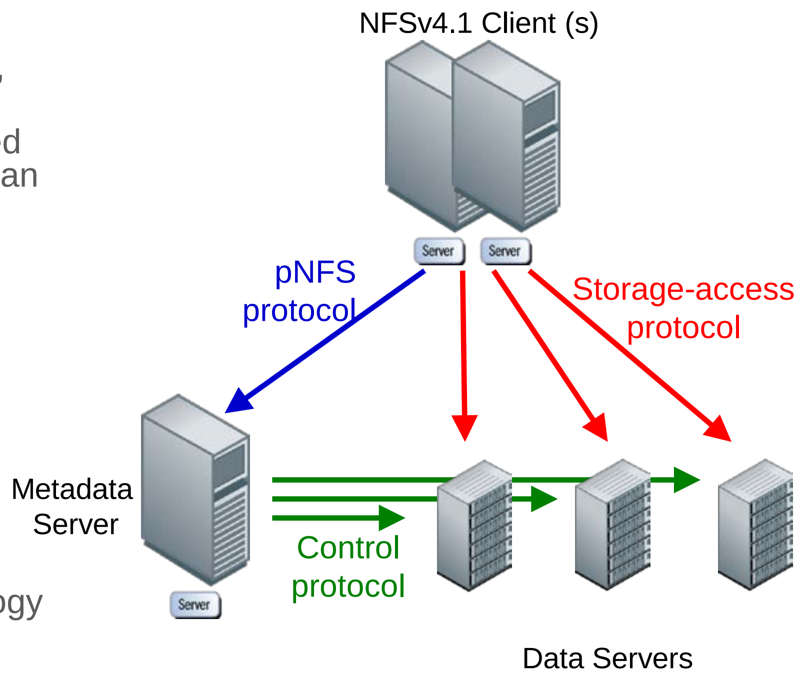
Relationship of pNFS to NFSv4.1

- RFC 3530bis – Network File System (NFS) Version 4 Protocol
  - NFSv4 (updated from RFC 3530 based on experience)
- RFC 5661 – Network File System (NFS) Version 4 Minor Version 1 Protocol
  - Specifies Sessions, Directory Delegations, and parallel NFS (pNFS) for files
- RFC 5663 - Parallel NFS (pNFS) Block/Volume Layout
- RFC 5664 - Object-Based Parallel NFS (pNFS) Operations
- pNFS is dependant on session support, which is only available in NFSv4.1



pNFS Terminology

- Metadata Server; the MDS
  - Maintains information about location and layout of files, objects or block data on data servers
  - Shown as a separate entity, but commonly implemented on one or across more than one data server as part of an array
- pNFS protocol
  - Extended protocol over NFSv4.1
  - Client to MDS communication
- Storage access protocol
  - Files; NFS operations
  - Objects: OSD SCSI objects protocol (OSD2)
  - Blocks; SCSI blocks (iSCSI, FCP)
- Control protocol
  - Not standardised; each vendor uses their own technology to do this
- Layout
  - Description of devices and sector maps for the data stored on the data servers
  - 3 types; files, block and object
- Callback
  - Asynchronous RPC calls used to control the behavior of the client during pNFS operations



pNFS Operations

- Client requests layout from MDS
- Layout maps the file/object/block to data server addresses and locations
- Client uses layout to perform direct I/O to the storage layer
- MDS or data server can recall the layout at any time using callbacks
- Client commits changes and releases the layout when complete
- pNFS is optional
  - Client can fall back to NFSv4

- pNFS operations
  - LAYOUTCOMMIT Servers commit the layout and update the meta-data maps
  - LAYOUTRETURN Returns the layout or the new layout, if the data is modified
  - GETDEVICEINFO Client gets updated information on a data server in the storage cluster
  - GETDEVICELIST Clients requests the list of all data servers participating in the storage cluster
  - CB_LAYOUT Server recalls the data layout from a client if conflicts are detected

pNFS Pre-requisites

- NFSv4.1 and pNFS capable server
  - Contact your NAS vendor for availability
  - Commercial products available for all of files, blocks and object types
  - Open source Linux pNFS server in development
    - http://wiki.linux-nfs.org/wiki/index.php/PNFS_Development
- pNFS capable client
  - Linux to date
  - See previous BrightTalks
    - Part3 – Planning for a Smooth Migration

pNFS Files Mount

- RHEL6.4 pNFS mount
  - mount -o minorversion=1 server:/filesystem /mnt
- Check
  - (output edited)

```
/proc/self/mountstats

device 172.16.92.172:/filesystem mounted on /mnt with fstype
nfs4 statvers=1.1
opts: …,vers=4.1, …
nfsv4: …,sessions,pnfs=nfs_layout_nfsv41_files
…
```

pNFS Client Mount

```
47 27.086618  172.17.40.185   172.17.40.171   NFS   282 V4 Call (Reply In 48) EXCHANGE_ID
48 27.086762  172.17.40.171   172.17.40.185   NFS   266 V4 Reply (Call In 47) EXCHANGE_ID
49 27.086883  172.17.40.185   172.17.40.171   NFS   330 V4 Call (Reply In 51) CREATE_SESSION
50 27.087003  172.17.40.171   172.17.40.185   NFS   146 V1 CB_NULL Call (Reply In 53)
51 27.087032  172.17.40.171   172.17.40.185   NFS   194 V4 Reply (Call In 49) CREATE_SESSION
```

```
⊞ Ethernet II, Src: Netapp_20:7a:42 (00:a0:98:20:7a:42), Dst: IntelCor_2b:40:06 (00:1b:21:2b:40:06)
⊞ Internet Protocol Version 4, Src: 172.17.40.171 (172.17.40.171), Dst: 172.17.40.185 (172.17.40.185)
⊞ Transmission Control Protocol, Src Port: nfs (2049), Dst Port: 1007 (1007), Seq: 29, Ack: 261, Len: 200
⊞ Remote Procedure Call, Type:Reply XID:0x634bd45a
⊟ Network File System, Ops(1): EXCHANGE_ID
    [Program Version: 4]
    [V4 Procedure: COMPOUND (1)]
    Status: NFS4_OK (0)
  ⊞ Tag: <EMPTY>
  ⊟ Operations (count: 1)
    ⊟ Opcode: EXCHANGE_ID (42)
        Status: NFS4_OK (0)
        clientid: 0x6387220000000004
        seqid: 0x00000001
      ⊟ eir_flags: 0x00060100
          0... .... .... .... .... .... .... .... = EXCHGID4_FLAG_CONFIRMED_R: Not set
          .0.. .... .... .... .... .... .... .... = EXCHGID4_FLAG_UPD_CONFIRMED_REC_A: Not set
          .... .... .... .1.. .... .... .... .... = EXCHGID4_FLAG_USE_PNFS_DS: Set
          .... .... .... ..1. .... .... .... .... = EXCHGID4_FLAG_USE_PNFS_MDS: Set
          .... .... .... ...0 .... .... .... .... = EXCHGID4_FLAG_USE_NON_PNFS: Not set
          .... .... .... .... .... ...1 .... .... = EXCHGID4_FLAG_BIND_PRINC_STATEID: Set
          .... .... .... .... .... .... .... ..0. = EXCHGID4_FLAG_SUPP_MOVED_MIGR: Not set
          .... .... .... .... .... .... .... ...0 = EXCHGID4_FLAG_SUPP_MOVED_REFER: Not set
        eia_state_protect: SP4_NONE (0)
```

172.17.40.185 – IP address of the pNFS client
172.17.40.171 – IP address of the server

Client and Server handshake to determine respective Capabilities. The Cluster replies with MDS and DS flags set, indicating capability for both

pNFS Client to MDS

```
117 44.370851  172.17.40.185   172.17.40.171   NFS   418 V4 Call (Reply In 118) OPEN DH:0x7f69f7d7/testfile5
118 44.470682  172.17.40.171   172.17.40.185   NFS   566 V4 Reply (Call In 117) OPEN StateID:0xa36e
119 44.470856  172.17.40.185   172.17.40.171   NFS   338 V4 Call (Reply In 120) SETATTR FH:0x4c99adea
120 44.471391  172.17.40.171   172.17.40.185   NFS   318 V4 Reply (Call In 119) SETATTR
121 44.477141  172.17.40.185   172.17.40.171   NFS   342 V4 Call (Reply In 122) LAYOUTGET
122 44.477244  172.17.40.171   172.17.40.185   NFS   306 V4 Reply (Call In 121) LAYOUTGET
123 44.477406  172.17.40.185   172.17.40.171   NFS   274 V4 Call (Reply In 124) GETDEVINFO
124 44.477501  172.17.40.171   172.17.40.185   NFS   218 V4 Reply (Call In 123) GETDEVINFO
129 44.477982  172.17.40.185   172.17.40.173   NFS   110 V4 NULL Call (Reply In 130)
130 44.478154  172.17.40.173   172.17.40.185   NFS    94 V4 NULL Reply (Call In 129)
```

```
    Status: NFS4_OK (0)
    sessionid: 00000004638722000000000000000000
    seqid: 0x00000017
    slot ID: 0
    high slot id: 0
    target high slot id: 15
    status: 0
⊟ Opcode: GETDEVINFO (47)
    Status: NFS4_OK (0)
    layout type: LAYOUT4_NFSV4_1_FILES (1)
    device index: 0
  ⊟ r_netid: tcp
      length: 3
      contents: tcp
      fill bytes: opaque data
  ⊟ r_addr: 172.17.40.173.8.1
      length: 17
      contents: 172.17.40.173.8.1
      fill bytes: opaque data
  [Main Opcode: GETDEVINFO (47)]
```

The OPEN and SETATTR are sent to the MDS

MDS LAYOUT to pNFS Client

```
121 44.477141 172.17.40.185   172.17.40.171   NFS   342 V4 Call (Reply In 122) LAYOUTGET
122 44.477244 172.17.40.171   172.17.40.185   NFS   306 V4 Reply (Call In 121) LAYOUTGET
123 44.477406 172.17.40.185   172.17.40.171   NFS   274 V4 Call (Reply In 124) GETDEVINFO
124 44.477501 172.17.40.171   172.17.40.185   NFS   218 V4 Reply (Call In 123) GETDEVINFO
129 44.477982 172.17.40.185   172.17.40.173   NFS   110 V4 NULL Call (Reply In 130)
130 44.478154 172.17.40.173   172.17.40.185   NFS    94 V4 NULL Reply (Call In 129)
```

```
    Status: NFS4_OK (0)
 ⊟ Opcode: LAYOUTGET (50)
    Status: NFS4_OK (0)
    return on close?: No
  ⊟ stateid
      [StateID Hash: 0x28fd]
      seqid: 0x00000001
      Data: 032287634f000e0000000000
  ⊟ Layout Segment (count: 1)
      offset: 0
      length: 18446744073709551615
      IO mode: IOMODE_RW (2)
      layout type: LAYOUT4_NFSV4_1_FILES (1)
      device ID: 01010100160400800000000001000000
      nfl_util: 0x00010000
      first stripe to use index: 0
      offset: 0
    ⊞ File Handles (count: 1)
  [Main Opcode: LAYOUTGET (50)]
```

Before reading or writing data, the pNFS client requests the layout

The map of data servers and file handles is returned

pNFS Client DEVICEINFO from MDS

```
117 44.370851  172.17.40.185   172.17.40.171   NFS   418 V4 Call (Reply In 118) OPEN DH:0x7f69f7d7/testfile5
118 44.470682  172.17.40.171   172.17.40.185   NFS   566 V4 Reply (Call In 117) OPEN StateID:0xa36e
119 44.470856  172.17.40.185   172.17.40.171   NFS   338 V4 Call (Reply In 120) SETATTR FH:0x4c99adea
120 44.471391  172.17.40.171   172.17.40.185   NFS   318 V4 Reply (Call In 119) SETATTR
121 44.477141  172.17.40.185   172.17.40.171   NFS   342 V4 Call (Reply In 122) LAYOUTGET
122 44.477244  172.17.40.171   172.17.40.185   NFS   306 V4 Reply (Call In 121) LAYOUTGET
123 44.477406  172.17.40.185   172.17.40.171   NFS   274 V4 Call (Reply In 124) GETDEVINFO
124 44.477501  172.17.40.171   172.17.40.185   NFS   218 V4 Reply (Call In 123) GETDEVINFO
129 44.477982  172.17.40.185   172.17.40.173   NFS   110 V4 NULL Call (Reply In 130)
130 44.478154  172.17.40.173   172.17.40.185   NFS    94 V4 NULL Reply (Call In 129)
```

```
    Status: NFS4_OK (0)
    sessionid: 0000000463872200000000000000000
    seqid: 0x00000017
    slot ID: 0
    high slot id: 0
    target high slot id: 15
    status: 0
  Opcode: GETDEVINFO (47)
    Status: NFS4_OK (0)
    layout type: LAYOUT4_NFSV4_1_FILES (1)
    device index: 0
    r_netid: tcp
      length: 3
      contents: tcp
      fill bytes: opaque data
    r_addr: 172.17.40.173.8.1
      length: 17
      contents: 172.17.40.173.8.1
      fill bytes: opaque data
  [Main Opcode: GETDEVINFO (47)]
```

Meta-data node provides the pNFS client with the IP information for the DS. In this example – 172.17.40.173

Information is cached for life of the layout or until recalled (for example, when the data is moved)

pNFS Client Uses Direct Data Path

123	44.477406	172.17.40.185	172.17.40.171	NFS	274 V4 Call (Reply In 124) GETDEVINFO
124	44.477501	172.17.40.171	172.17.40.185	NFS	218 V4 Reply (Call In 123) GETDEVINFO
129	44.477982	172.17.40.185	172.17.40.173	NFS	110 V4 NULL Call (Reply In 130)
130	44.478154	172.17.40.173	172.17.40.185	NFS	94 V4 NULL Reply (Call In 129)
132	44.478663	172.17.40.185	172.17.40.173	NFS	282 V4 Call (Reply In 133) EXCHANGE_ID
133	44.478784	172.17.40.173	172.17.40.185	NFS	266 V4 Reply (Call In 132) EXCHANGE_ID
134	44.478918	172.17.40.185	172.17.40.173	NFS	330 V4 Call CREATE_SESSION
163	60.480000	172.17.40.185	172.17.40.173	NFS	330 V4 Call (Reply In 206) CREATE_SESSION
169	64.476795	172.17.40.185	172.17.40.171	NFS	242 V4 Call (Reply In 170) SEQUENCE
170	64.476916	172.17.40.171	172.17.40.185	NFS	150 V4 Reply (Call In 169) SEQUENCE
191	76.480717	172.17.40.185	172.17.40.173	NFS	330 V4 Call CREATE_SESSION

```
⊟ Network File System, Ops(2): SEQUENCE GETDEVINFO
    [Program Version: 4]
    [V4 Procedure: COMPOUND (1)]
    Status: NFS4_OK (0)
  ⊞ Tag: <EMPTY>
  ⊟ Operations (count: 2)
    ⊞ Opcode: SEQUENCE (53)
    ⊟ Opcode: GETDEVINFO (47)
        Status: NFS4_OK (0)
        layout type: LAYOUT4_NFSV4_1_FILES (1)
        device index: 0
      ⊟ r_netid: tcp
          length: 3
          contents: tcp
          fill bytes: opaque data
      ⊟ r_addr: 172.17.40.173.8.1
          length: 17
          contents: 172.17.40.173.8.1
          fill bytes: opaque data
    [Main Opcode: GETDEVINFO (47)]
```

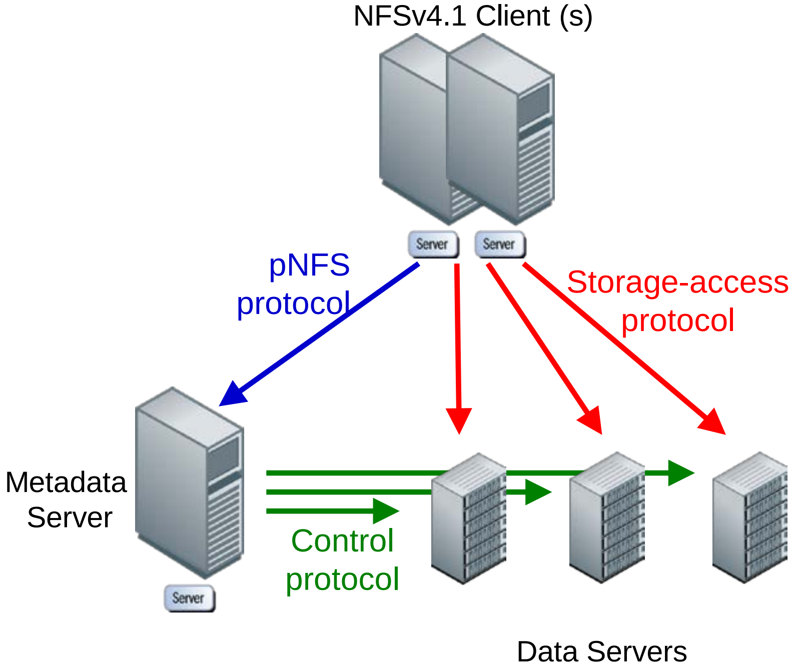
Now the pNFS client is reaching out to the remote volume on a direct path using IP address 172.17.40.173.

In Summary: The Benefits of pNFS

NFSv4.1 (pNFS) can aggregate bandwidth

- Modern approach; relieves issues associated with point-to-point connections

- pNFS Client
  - Client read/write a file
  - Server grants permission
  - File layout (stripe map) is given to the client
  - Client parallel R/W directly to data servers
- Removes IO Bottlenecks
  - No single storage node is a bottleneck
  - Improves large file performance
- Improves Management
  - Data and clients are load balanced
  - Single Namespace



Linux Client and NFSv4.1

- Upstream (Linus) Linux NFSv4.1 client support
  - Basic client in Kernel 2.6.32
  - pNFS support (files layout type) in Kernel 2.6.39
  - Support for the 'objects' and 'blocks' layouts was merged in Kernel 3.0 and 3.1 respectively
- Full read and write support for all three layout types in the upstream kernel
  - Blocks, files and objects
  - O_DIRECT reads and writes supported

Linux Client and NFSv4.1 (cont.)

- pNFS client support in distributions
  - Fedora 15 was first for pNFS files
  - Kernel 2.6.40 (released August 2011)
- Red Hat Enterprise Linux (RHEL)
  - “Technical preview" support for NFSv4.1 and for the pNFS files layout type in version 6.2, 6.3
  - Full support in RHEL6.4
- Ubuntu, SUSE & other distributions
  - Possible to upgrade to NFSv4.1
- No support in Solaris
  - Both server and client are NFSv4 only

Summary/Call to Action

- NFS has more relevance today for commercial, HPC and other use cases than it ever did
  - Features for a virtualized data centers
- Developments driven by application requirements
- Adoption slow, but will continue to increase
  - NFSv4 support widely available
  - New NFSv4.1 with client & server support
  - NFS defines how you get to storage, not what your storage looks like
- Start using NFSv4.1 today
  - NFSv4.2 nearing approval
  - pNFS offers performance support for modern NAS devices
- Planning is key
  - Application, issues & actions to ensure smooth implementations
- pNFS
  - First open standard for parallel I/O across the network
  - Ask vendors to include NFSv4.1 and pNFS support for client/servers
  - pNFS has wide industry support
  - Commercial implementations and open source

Attribution & Feedback

The SNIA Education Committee thanks the following individuals for their contributions to this Tutorial.

Authorship History

Alex McDonald, April 2013

Updates:
Alex McDonald, March 2014

Additional Contributors

Tom Haynes, NetApp

Please send any questions or comments regarding this SNIA Tutorial to ***tracktutorials@snia.org***

Additional Material

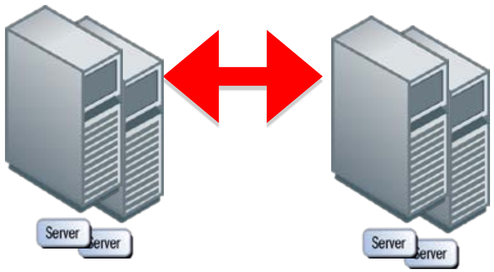
- Future NFSv4.2 and FedFS (in addendum slides)
  - New features in NFSv4.2
  - FedFS: Global namespace; IESG has approved Dec 2012

Other NFS Performance Capabilities

- Trunking (NFSv4.1 & pNFS)
  - A single data server connection limits data throughput based on protocol
  - Trunking “bundles” connections into a single pipe
    - Open multiple sessions via different physical Ethernet connections to the same file handle/data server resource
  - Expands throughput and can reduce latency
  - No implementations as yet

New Features in NFSv4.2

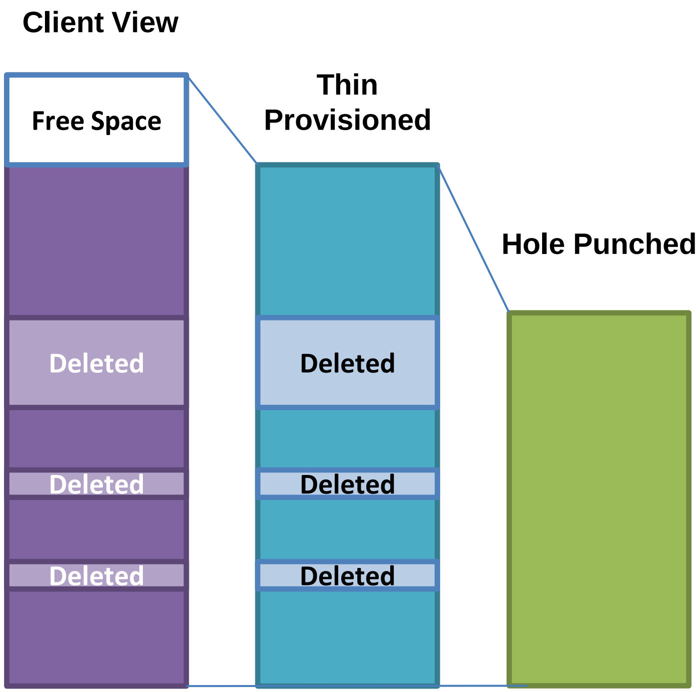
- Server-Side Copy (SSC)
  - Removes one leg of the copy
  - Destination reads directly from the source



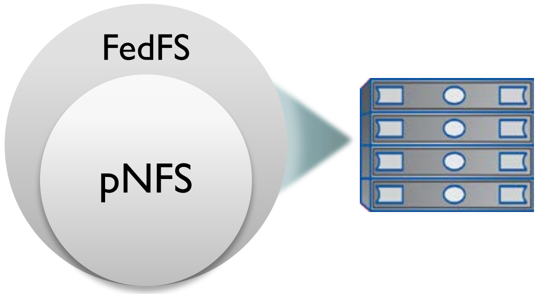
- Application Data Blocks
  - Allows definition of the format of file
  - Examples: database or a VM image.
  - INITIALIZE blocks with a single compound operation
    - Initializing a 30G database takes a single over the wire operation instead of 30G of traffic.

New Features in NFSv4.2

- Space reservation
  - Ensure a file will have storage available
- Sparse file support
  - “Hole punching” and the reading of sparse files
- Labeled NFS (LNFS)
  - MAC checks on files
- IO_ADVISE
  - Client or application can inform the server caching requirements of the file



Federated File System: FedFS



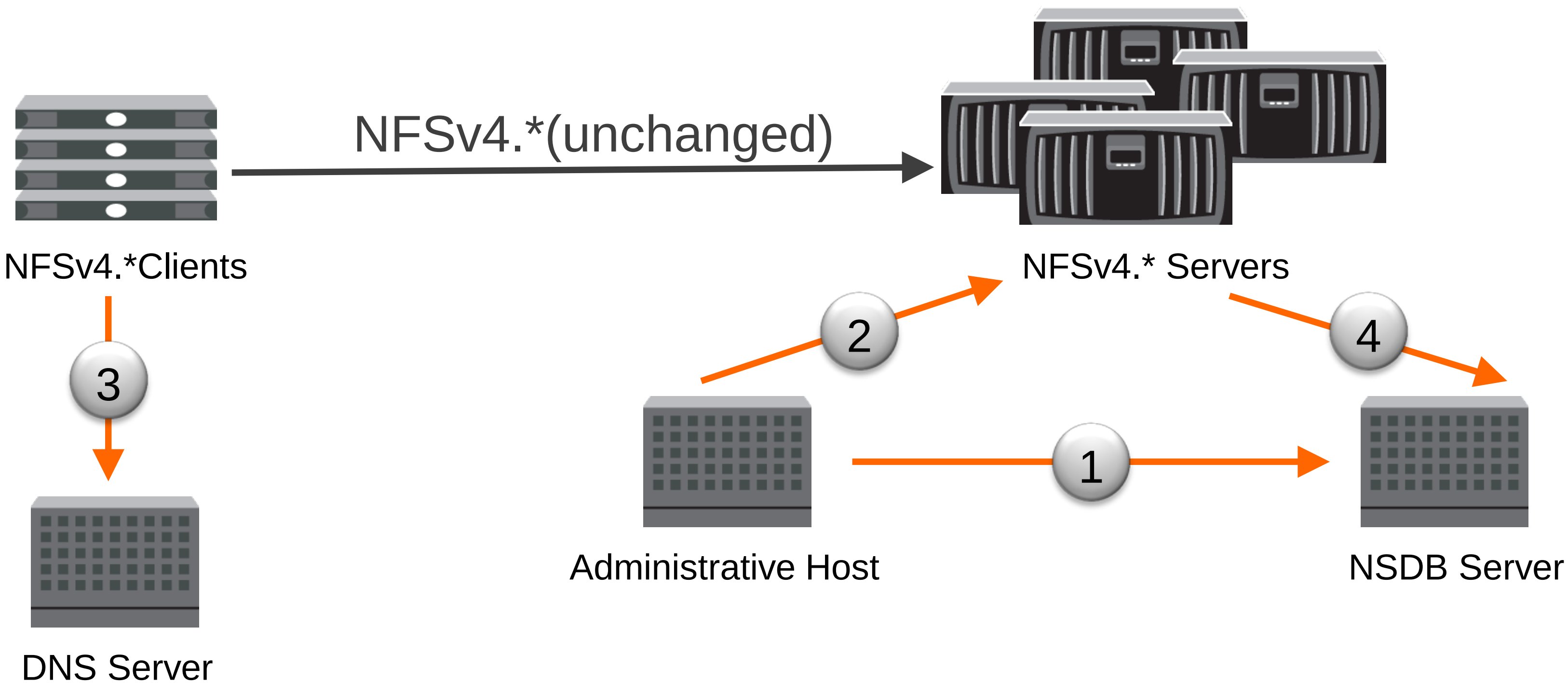
- Federated File System
  - Uniform namespace that has local and geographically global referral infrastructure
  - Accessible to unmodified NFSv4 clients
  - Addresses directories, referrals, nesting, and namespace relationships
- Client finds namespace via DNS lookup
  - Sees junctions (directories) and follows them as NFSv4 referrals

What is FedFS?

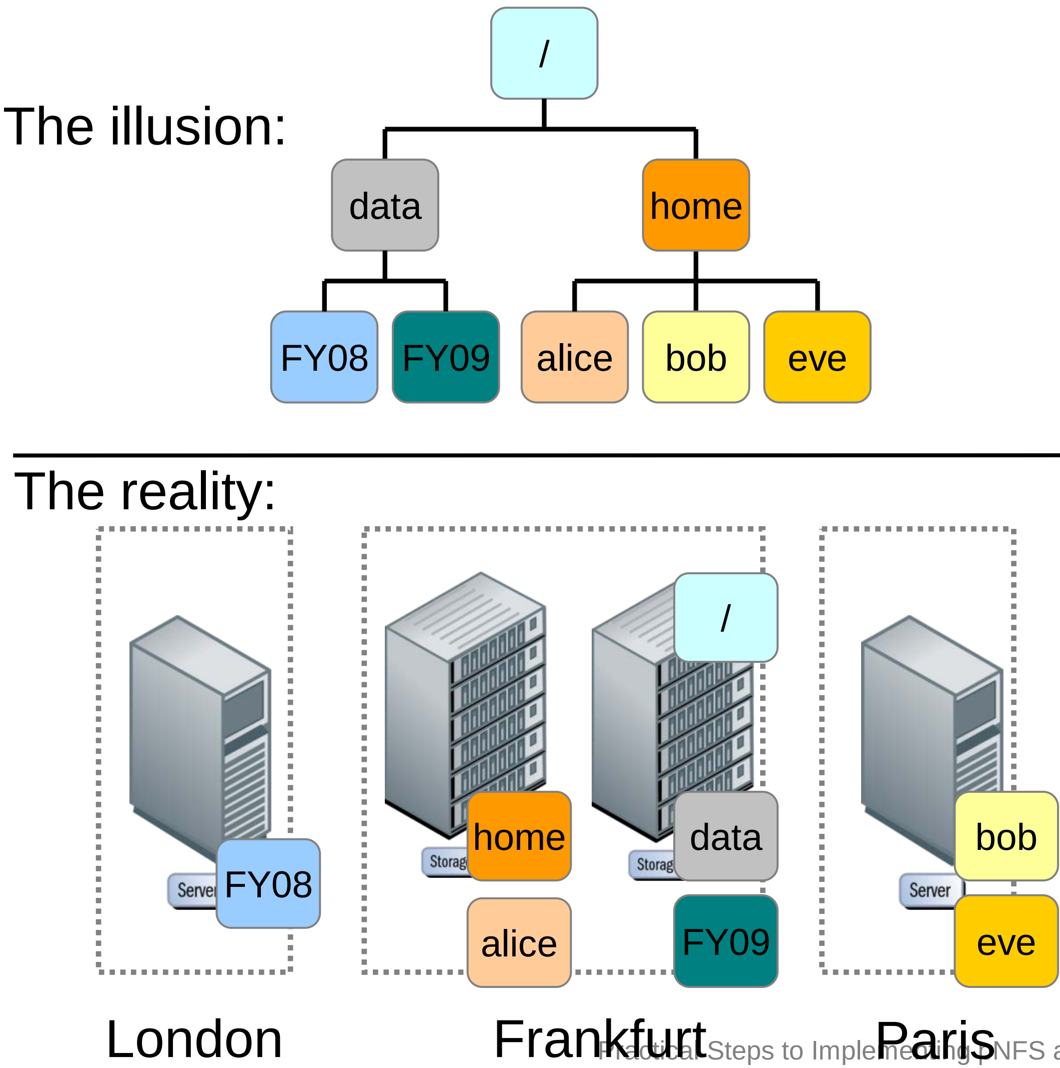
- FedFS is a set of open protocols that permit the construction of a scalable, cross-platform federated file system namespace accessible to unmodified NFSv4[.1] clients.
- Key points:
  - Unmodified clients
  - Open: cross-platform, multi-vendor
  - Federated: participants retain control of their systems
  - Scalable: supports large namespaces with many clients and servers in different geographies

FedFS Protocols

Namespace Management	Namespace Navigation
1 NSDB Management (LDAP)	3 Namespace discovery (DNS)
2 Junction Management (ONC RPC)	4 Junction resolution (LDAP)

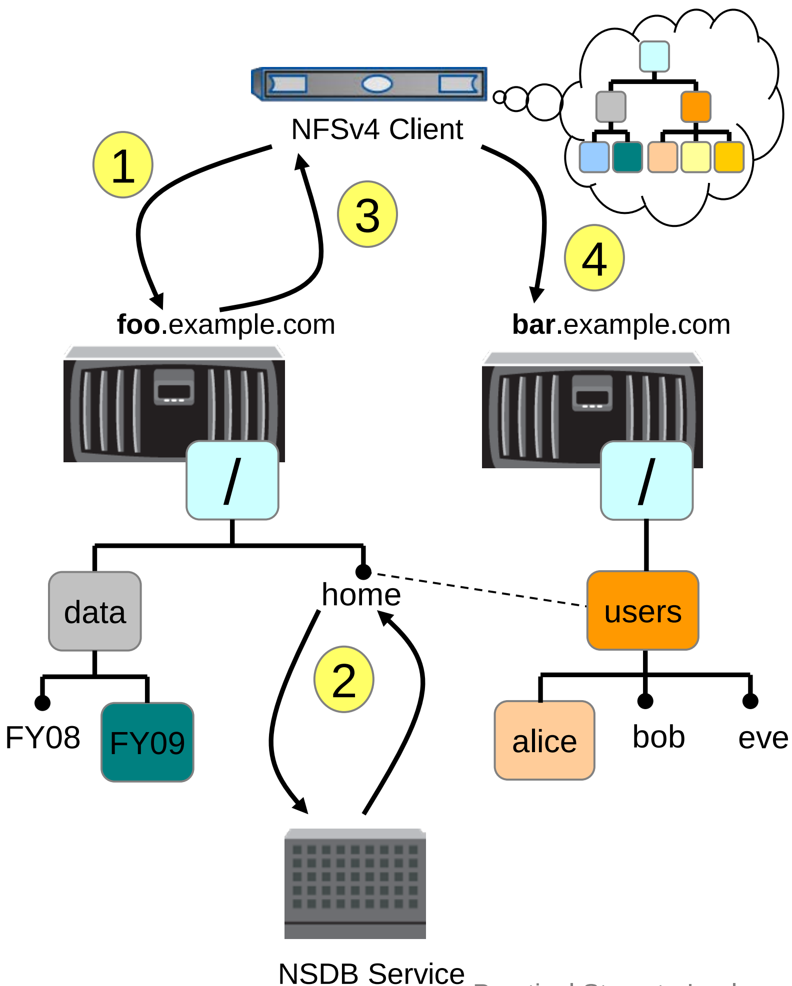


FedFS Example



- The user and application software see a simple, hierarchical namespace
- Behind the scenes, simple management operations allow data mobility for high performance, high reliability, and high availability

FedFS Example



The user requests `/home/alice`:

1. The client attempts to access `/home/alice` on server **foo**.
2. Server **foo** discovers that home is a namespace junction and determines its location using the FedFS NSDB service.
3. Server foo returns an NFSv4 referral to the client directing it to server **bar**'s `/users`.
4. The client accesses `/users/alice` on server **bar**.

Benefits of FedFS

- Simplified management
  - Eliminates complicated software such as the automounter
- Separates logical and physical data location
  - Allows data movement for cost/performance tiering, worker mobility, and application mobility
- Enhances:
  - Data Replication
    - Load balancing or high availability
  - Data Migration
    - Moving data closer to compute or decommissioning systems
  - Cloud Storage
    - Dynamic data center, enterprise clouds, or private internet clouds.