Technical Report

Parallel Network File System Configuration and Best Practices for Clustered Data ONTAP 8.2

Bikash Roy Choudhury, NetApp

June 2013 | TR-4063

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

Since its introduction in 1984, the Network File System (NFS) has become a standard for network file sharing, particularly in UNIX® and Linux® environments. The NFS protocol has steadily evolved to adapt to new requirements and market changes.

Compute clusters running computer-aided engineering, seismic data processing, bioinformatics, and other science and engineering applications often rely on NFS to access shared data. However, because all files in a file system must be accessed through a single file server, NFS can result in significant bottlenecks for the aforementioned applications. One common way to scale performance is to “scale up” the performance of that single file server that gets CPU and memory limited at some point. Another way to scale performance is to “scale out” storage—clients connect to a single file server, but data is distributed across numerous servers using a clustered file system. For some applications this can increase performance, but it might not accelerate access to single large files or eliminate all bottlenecks. These limitations have created a strong need for an industry-standard method to “scale out” shared storage, analogous to the way servers are scaled out in a compute cluster.

NFS version 4 (NFSv4) addresses some of the limitations found in NFSv3 through compound operations and the use of delegations for improved client-side caching. Despite these enhancements, the single-server bottleneck remains. The latest update to the NFS protocol, NFS version 4.1, includes a specification for parallel NFS (pNFS), which has been designed to eliminate the single-server bottleneck.

1.1 Scope

This document covers the following topics:

- Introduction to parallel network file system (pNFS)
- Architecture of pNFS
- Implementation of pNFS on the NetApp® clustered Data ONTAP® 8.2operating system
- pNFS use cases
- Comparison of traditional NFS and pNFS
- Configuration and best practices for pNFS

1.2 Intended Audience and Assumptions

This technical report is for storage administrators, system administrators, and data center managers. It assumes basic familiarity with the following:

- NetApp FAS systems and the Data ONTAP operating system
- An understanding of protocols (NFS in particular)

2 Overview of Clustered Data ONTAP 8.2

Clustered Data ONTAP 8.2 is primarily built for a location-transparent cluster namespace**.** A collection of storage and networking resources from physical controllers in the cluster form a manageable entity called a Storage Virtual Machine (SVM)**.** All the storage resources in that SVM are accessible over all the networking resources contained in the SVM.

Depending on the location of the volume in the cluster namespace, the I/O requests from the client are served locally (both network and storage resources are on the same physical controller) or remotely (network and storage resources are on different physical controllers, and a cluster network hop is required to reach the storage). In most cases, a nonoptimized path to a remote volume could reduce the overall throughput of the cluster. Providing an optimized and direct data path from the client to the flexible volumes in the cluster namespace from the protocol level provides the possibility for consistent local data access.

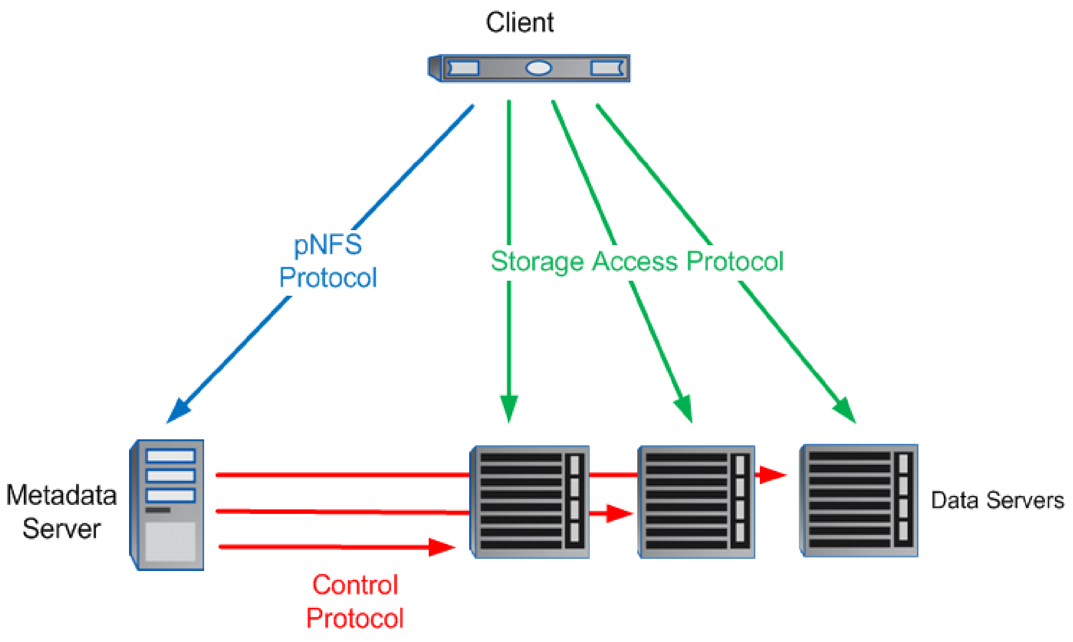
3 Overview of pNFS

Parallel NFS (pNFS) is a new standard part of NFS version 4.1. NFSv4.1 (RFC 5662) is a minor release of NFSv4. NFSv4.1 does not modify any NFSv4 features and functionalities—it simply enhances what is already in place. With traditional NFS versions 3, 4, and 4.1, metadata and data are shared on the same I/O path. With pNFS, for the first time, there is an NFS feature that handles metadata and data on different I/O paths.

A metadata server handles all metadata activities from the client while the data servers provide a direct path for data access. pNFS has three different implementations: files, blocks, and objects. NetApp supports only file implementation at this time. The client communicates with the metadata server over NFSv4.1 while access to the data servers is through files, blocks, or objects. Every storage vendor has a different way of implementing pNFS in their stack, and NetApp supports pNFS only in clustered Data ONTAP at this time. There is no Data ONTAP 7G/7-Mode support for pNFS. The sections that follow describe in detail the pNFS architecture in clustered Data ONTAP.

3.1 Architecture

Figure 1) pNFS block diagram.



The pNFS architecture includes three main components:

- The metadata server (MDS) that handles all nondata traffic. The MDS is responsible for all metadata operations such as GETATTRs, SETATTRs, LOOKUPs, ACCESS, REMOVE, RENAME, and so on. The MDS also provides information on the layout of files.
- Data servers that store file data and respond directly to client read and write requests. Data servers handle pure READ and WRITE I/O.
- One or more clients that are able to access data servers directly, based on information in the metadata received from the metadata server.

There are three types of protocols used between the clients, metadata server, and data servers:

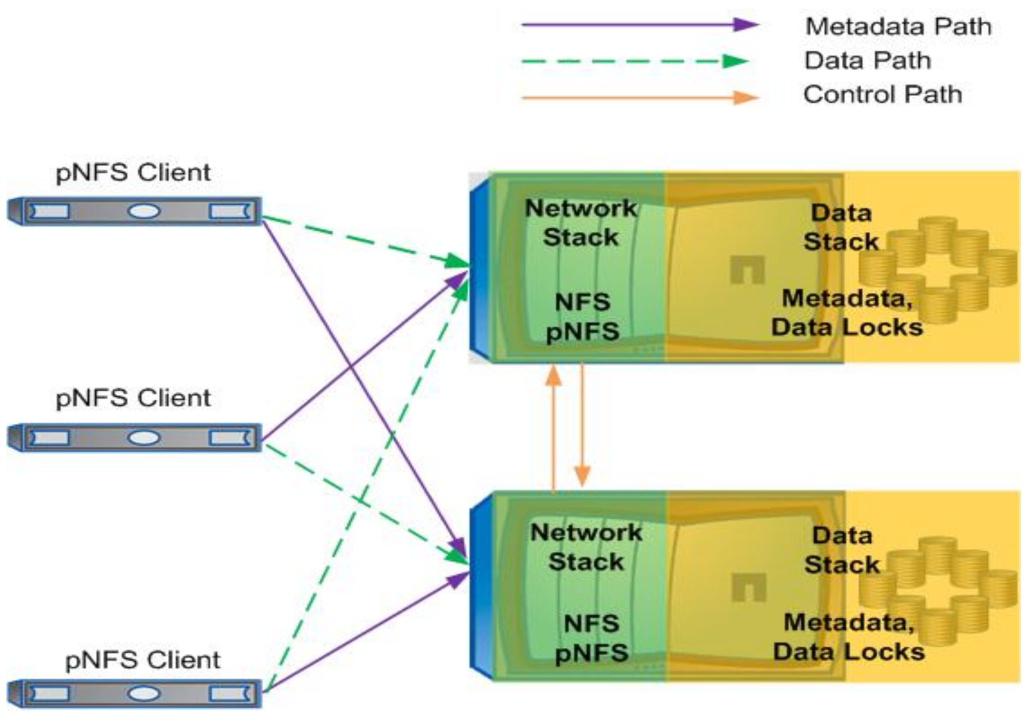
- A control protocol used between the metadata server and data servers to synchronize file system data.
- pNFS protocol, used between clients and the metadata server. This is essentially the NFSv4.1 protocol with a few pNFS-specific extensions. It is used to retrieve and manipulate layouts that contain the metadata that describes the location and storage access protocol required to access files stored on numerous data servers.
- A set of storage access protocols used by clients to access data servers directly. The pNFS specification currently has three categories of storage protocols: file based, block based, and object based. Data ONTAP 8.1 and later support file-based storage protocol and access the data servers over NFSv4.1.

The relationship between a client, a single server, and several storage devices for pNFS (server and client have access to all storage devices) is shown in the preceding figure.

3.2 Implementation of pNFS in Clustered Data ONTAP

.

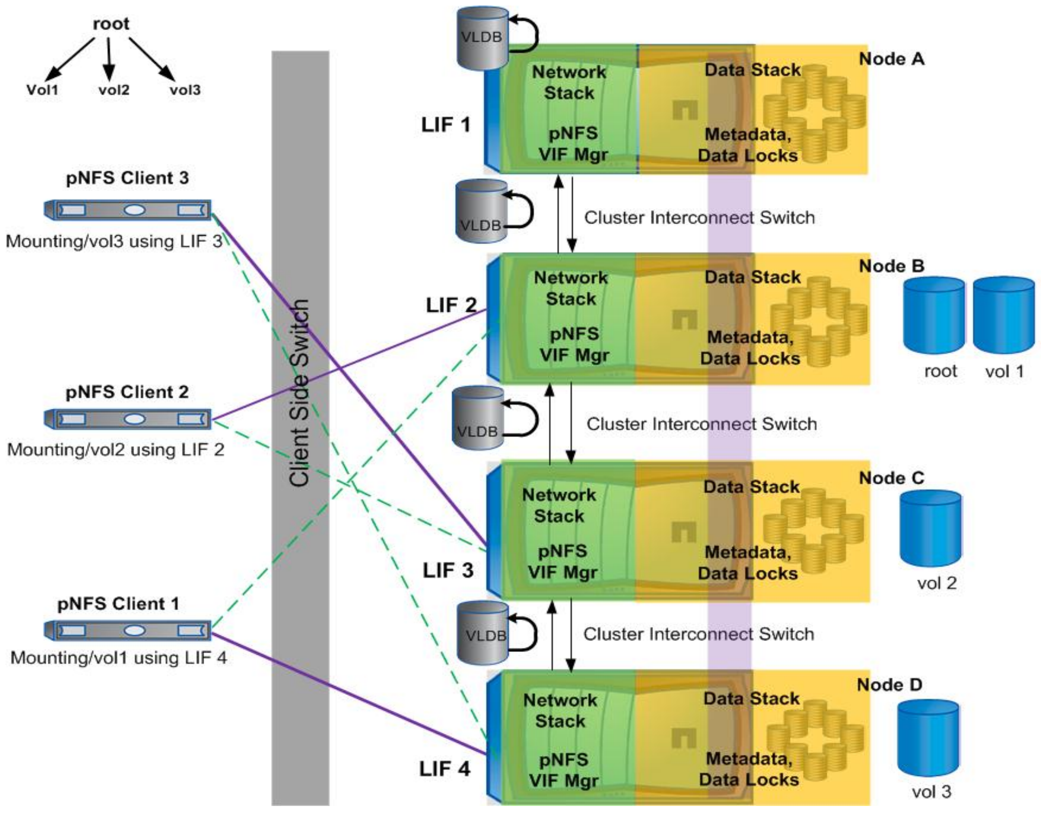
Figure 2) Implementation of pNFS in clustered Data ONTAP.



pNFS leverages clustered Data ONTAP technology and provides an optimized path to the storage that essentially eliminates a hop over the cluster network to serve data. Clients supporting pNFS always get their data served by the network addresses that are hosted on the same physical controller as the storage.

When the pNFS client mounts the Logical Interface (LIF) from the cluster namespace, the node that hosts the home port of that LIF becomes the metadata server. If that node also has a data volume in it then that node will also serve as a data server. That means that each node in a cluster namespace will behave as a metadata and data server for pNFS in clustered Data ONTAP implementation. The following example illustrates this behavior in more detail.

Figure 3) Workings of pNFS with clustered Data ONTAP.



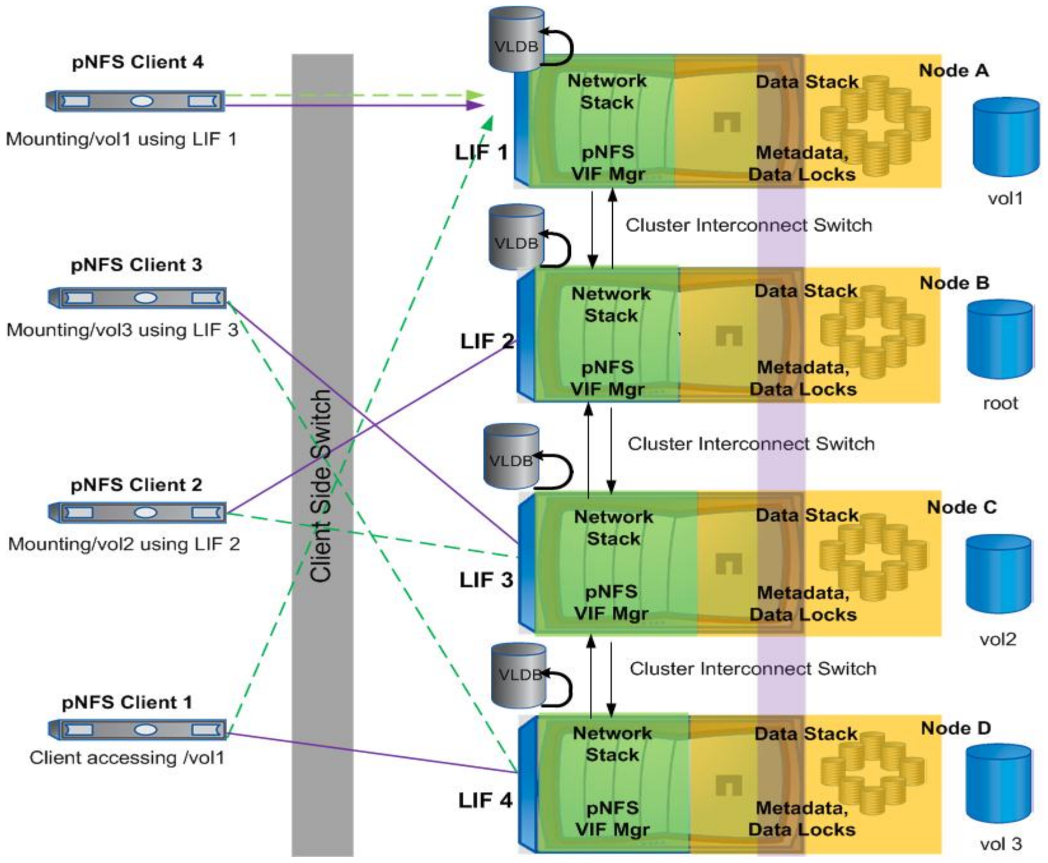
In the preceding figure, there are four nodes in the cluster: Node A, Node B, Node C, and Node D.

3.2.1 pNFS Local Data Access Scenarios

- pNFS Client 1 mounts the logical interface (LIF) from Node D (LIF 4). When Client 1 sends an OPEN request to open a file from the data Node D, it talks to the network stack of Node D. This becomes the metadata server for that pNFS client (Client 1), which mounts this LIF from Node D (LIF 4). If a file requested by the pNFS client exists in the corresponding data blade (D-blade), it is served locally from Node D. If the file requested by the client is in a volume that is in Node B, then that typically becomes a remote request for any regular NFSv3 and NFSv4 client. But with pNFS, the volume location database (VLDB) builds device information that contains location, layout of the file, and a path to reach that location. This information gets cached in the network stack (in this case, N stack of Node D) and is also passed on to the client. Once the client gets this information, it reaches out to the local network stack where the file resides (Node B).
- Similarly, pNFS Client 2 mounts the LIF from Node B (LIF 2). When Client 2 sends an OPEN request to open a file from Node B, it talks to the network stack (N stack) of Node B. This becomes the metadata server for pNFS Client 2 (LIF 2). The file requested by pNFS Client 2 is in a volume that is on Node C. VLDB now builds device information, which gets cached in the network stack of Node B, and this information is passed on to Client 2. Once this information is received by the client, it reaches out to the local network stack where the file resides (Node C).
- pNFS Client 3 mounts the LIF from Node C (LIF 3). When Client 3 sends an OPEN request to open a file from data Node C, it talks to the network stack (N-Stack) of Node C. LIF3 now becomes the metadata server for pNFS Client 3. The file requested by pNFS Client 3 is in a volume that is on Node D. VLDB then builds the necessary device information that gets cached in the network stack of Node

C, and this information is also passed on to Client 3. Once Client 3 gets this information, it reaches out to Node D's network stack, where the file resides.

Figure 4) Workings of pNFS with clustered Data ONTAP 8.1.



- Consider a scenario in which a `vol move` command is executed on /vol1, moving it from Node B to Node A. pNFS Client 1 has been accessing files on this volume (/vol1). With regular NFS, after a vol move operation, the client had to unmount or remount to access the file in the volume from the new location. With pNFS, this operation is nondisruptive. Once /vol1 is moved from Node B to Node A, VLDB updates the device information that is cached in the network stack of Node D. This information is also sent to Client 1. Once this information is received by the client, it reaches out to the updated location of the local network stack where the file resides (Node A).
- If Client 4 sends an OPEN request to open a file from the data Node A (LIF1) to communicate with the network stack of Node A, notice that the file requested by the pNFS Client 4 exists in the corresponding data stack of Node A and can be served locally by Node A.

4 Why Use pNFS with Clustered Data ONTAP?

Described below are the reasons why pNFS with clustered Data ONTAP makes sense for various workloads.

- **Efficient load balancing for business-critical applications.** As computing requirements of high-performance commercial environments keep growing, storage capacity, CPU utilization, and network bandwidth need to scale seamlessly. Although adding incremental storage capacity in a clustered Data ONTAP namespace is seamless, moving volumes to new storage for capacity growth, CPU, and network utilization rebalancing has to also be nondisruptive to business-critical applications running on the compute nodes, which require near-zero downtime. This could apply to an oil and gas scenario in which hundreds of compute nodes are tasked to complete numerous jobs.

With pNFS, any volume that is transitioned live within the cluster namespace does not affect the compute nodes accessing the volumes from a new storage node. The NFS file handle does not change when the file system is served from a different cluster node after the move within the namespace; therefore, the file system does not need to be unmounted or remounted. This efficient way of load balancing helps to address highly utilized CPU, memory, and network in clustered storage. Although additional storage nodes in the scale-out architecture provide additional CPU and memory resources, the intelligent pNFS client is updated with the new location of the file system, helping move the data access to the new location.

- **Effective resource utilization for applications with different workload characteristics.** With traditional NFS, metadata, data, and control share the same I/O path. As the data changes in its structure and grows in storage capacity, a single file system can have system resources saturated beyond a certain attainable performance. Applications implemented in electronic device automation (EDA) environments are very CPU intensive. The CPU on a single node in a cluster could most likely prove to be a bottleneck. While EDA demonstrates mostly sequential workload that drives high CPU utilization, there could also be workloads such as user home directories and scratch spaces for software build environments on the same storage that are mostly random in nature. Redistributing these workloads within a cluster namespace and providing an optimal access path to the file systems are always a challenge.

  Clients supporting pNFS, in addition to the necessary best practice recommendations for prevailing disk and aggregate sizing, can address heterogeneous workloads. A pNFS client can access file systems with different workload characteristics, from spinning disks or solid-state devices (SSDs), memory, and deduplication-aware extended cache. Irrespective of where the file system resides in the namespace, the pNFS clients access the volume on a direct data path that is isolated from the metadata and the control path. This provides parallelism to the application seeking access to file systems existing on different volumes through single or numerous NFS mounts, thereby improving the throughput and optimizing latency by distributing heterogeneous workload across numerous nodes.

- **Scale and provision storage to achieve quick service-level agreements (SLAs) for IT as a service (ITaaS).** With today's changing and diverse business requirements, achieving SLAs in a timely manner and shortening time to market are very critical. For cloud vendors who host ITaaS or platform as a service (PaaS), scaling, downsizing, and provisioning storage capacity require a quick turnaround time. While scale-out architecture provides the flexibility to provision storage on demand, volumes in the namespace could reside in different tiers of storage (SSDs, Fibre Channel, or SATA disks) to support different workload requirements (frequently used to less frequently used). Traditional NFS clients that mount volumes from storage will have to choose the correct cluster node at the time of mount to provide an optimized data path to meet the performance SLAs. If numerous data LIFs exist in an SVM, then choosing the correct path can prove to be challenging.

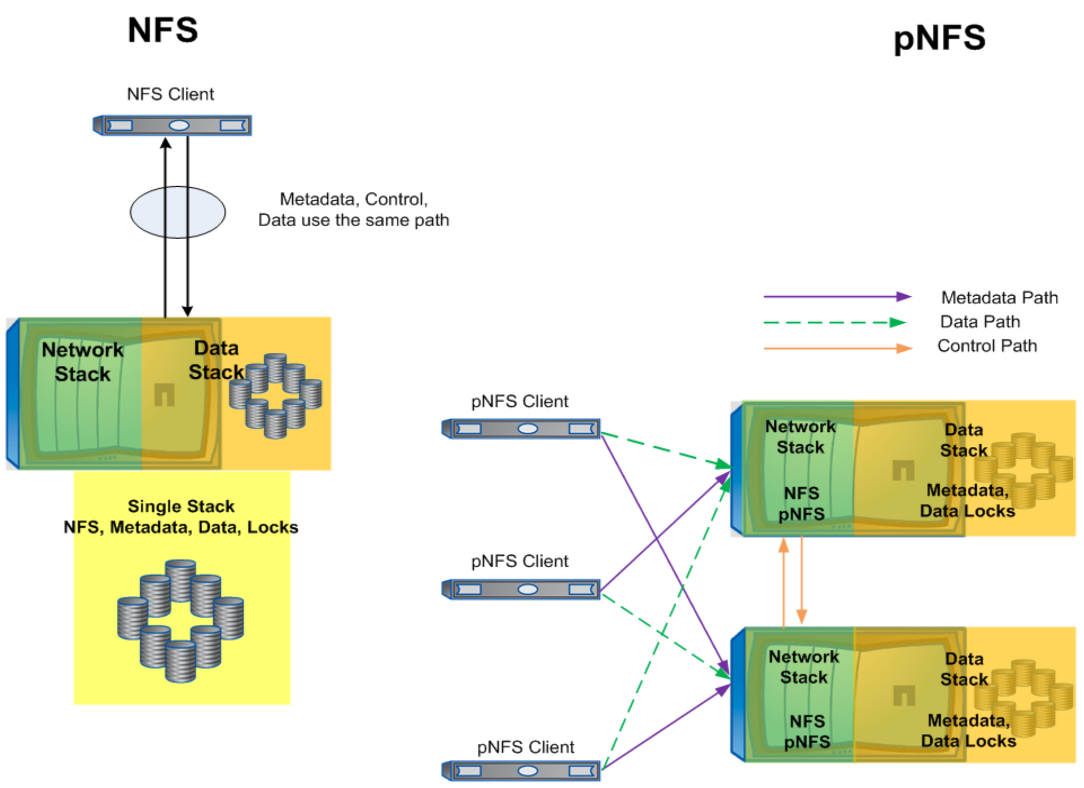
  NFSv4.1 and pNFS can provide flexibility for mounting the file system or volumes from anywhere in the cluster namespace. The volumes in the namespace that are remote to the mount point specified by the client can still have a direct data path for optimum performance. Clustered applications can be mounted over pNFS, while monolithic ones can still be mounted over NFSv3. File systems that are exported from storage can have clients mount over different varieties of NFS, so they can coexist without making any significant changes to applications accessing the data or client-side infrastructure. This reduces the overhead of frequent change management, which translates into time taken from SLAs. Numerous clients can simultaneously access different exported file systems. pNFS, as an open source, does not require additional software or drivers on the client that are proprietary to enable it. As long as the client and server support pNFS, a simple mount will suffice.

- **Unconventional NFS clients in hypervisors and databases to improve accessibility, reliability, and performance.** Traditionally, some of the leading and most popular hypervisors such as VMware® and databases such as Oracle® have a built-in NFSv3 client that performs remote procedure call (RPC) communication similar to kernel NFS. This NFS client is lightweight and is integrated into the independent vendor stack. Today, the majority of NFSv3 traffic to a hypervisor datastore or a database volume uses a single TCP session. This implies that even if there is link aggregation at the network layer, clients are still limited by a single TCP session. The performance scales linearly as the number of connections is increased. There is no multipath support in NFSv3 either, so a client is unable to have several connections to a single NFSv3 server. If this single TCP session breaks (such as with a link failure), there must be another standby path for the RPC communication to resume. This limits the accessibility, reliability, and performance between the client and the server.

NFSv4.1/pNFS features will have a positive impact when these nonconventional NFS clients start adopting NFSv4.1 in their stack. NFSv4.1 provides a session's model that allows numerous connections in one session. Presently, an NFS4.1 client in a hypervisor and database can have numerous connections in one session.

5 NFS Versus pNFS

Figure 5) NFS versus pNFS.



5.1 Challenges with Traditional NFS

You face the following challenges with traditional NFS.

- The NFS requests from the client shares the same I/O path for both metadata and data traffic. This becomes overwhelming at times, when applications with high metadata operations can saturate CPU cycles on a single controller head that also handles data reads and writes. When the number of users accessing the application increases, scalability becomes a challenge when a single controller head is responsible for handling these operations.
- With traditional NFS versions such as 3, 4, or 4.1, distributing workload within a single HA pair or between a namespace within a cluster setup requires numerous NFS mounts from the client to access the flexible volumes that are exporting the file systems. Managing these exported file systems could be an overhead for the administrators when the application requires access to additional

volumes that hold the data that is archived or less frequently used. Separate mounts would also be required to access volumes hosted from the SAS/FC disk or from SATA disk aggregates, respectively.

- Without adequate sizing and best practice adoption for different workloads, the data path from a regular NFS client would traverse an unoptimized path in a cluster to access volumes scattered over all the nodes in the namespace. This could impair optimum performance, with limited local access to the data volumes.
- With a single server design and a single IP address assigned to a single physical interface, the movement of data across nodes is a challenge. Copying and mirroring are the only options to move the data onto a different controller for network and workload balancing from a single HA pair. A different IP address from the new location might be required to mount on the NFS client. This will lead the client to unmount the old IP address and remount the updated IP address to access the volumes from the new location. This mechanism proves to be disruptive for applications that are less tolerant to downtime.

5.2 Advantages of pNFS

The following are the advantages of pNFS.

- **Eliminates the single controller bottleneck.**
  - Unlike traditional NFS, pNFS provides isolation of metadata and control from the data path. The metadata server (MDS) handles all nondata traffic such as GETATTRs, SETATTRs, ACCESS, LOOKUPs, and so on. Data servers (DSs) store file data and respond directly to client read and write requests. A control protocol is used to provide synchronization between the metadata server and data server. This eliminates the single controller bottleneck to handle all the NFS operations by spreading out the metadata operations to be handled by one server, while other numerous data servers handle the read or write operations.
  - With pNFS, any volume that is moved to a different node within the cluster namespace for load-balancing purposes is not disruptive and is completely transparent to the application. The NFS file handle does not change when the file system is served from a different cluster node after the move within the namespace; therefore, the file system does not need to be unmounted or remounted. This efficient system of load balancing helps to address highly utilized CPU, memory, and network in the clustered storage.
- **Improves scalability and manageability.** The pNFS client works seamlessly when the application tries to access flexible volumes hosting file systems from nodes that are added to the cluster namespace without any disruption. No additional mounts are required from the client for the application to access these volumes as long as the application identifies that these volumes are valid for its use. Typically in a scenario in which hierarchical storage management is deployed, aging files are automatically moved to volumes created in less expensive SATA disks. And when the application might need to access these volumes sporadically, pNFS along with the benefits of cluster namespace requires no additional mounts from the client to access the new volumes. The application can access the data without any additional mounts from the different storage tiers. While additional storage nodes in the scale-out architecture provide additional CPU and memory resources, the intelligent pNFS client is updated with the new location of the file system, thus providing a direct path to the data nodes.
- **Coexists with traditional NFS protocol.** The pNFS part of NFSv4.1 and other versions of NFS are open source and are driven by the Internet Engineering Task Force (IETF) community. Therefore, the different varieties of NFS can coexist at the same time, as long as the clients follow the appropriate RFCs. This means that the supported NFS clients can mount the same file system over NFSv3, NFSv4, and NFSv4.1/pNFS. However, if an NFSv4.x access control list (ACL) is set on a file or directory, then the permissions are enforced on the appropriate file or directory. While an NFSv4 client can list all the access control entries (ACEs), an NFSv3 client might not see the actual ACLs set on it, because it does not support ACLs.

- **Provides data locality.** pNFS provides a direct path to access volumes in a namespace as long as a valid network path is available to reach the cluster node. With pNFS, the unoptimized path to access any volume in the namespace is eliminated. pNFS leverages clustered Data ONTAP technology and provides an optimized path to the storage that essentially eliminates a cluster hop to serve the data; clients supporting pNFS always get their data served by the network addresses that are hosted on the physical controller as the storage.

6 Technical Specifications of pNFS

6.1 pNFS Workflow

The pNFS workflow includes two stages.

1. In the first stage, an eligible pNFS client uses any IP address or host name in the cluster namespace to mount the flexible volume/file system. During the mount process, the NetApp cluster identifies that the node hosting the LIF is capable of being a metadata and/or a data server. Once the mount is completed successfully, the node that hosted the IP address becomes the metadata server. The data volumes can be on the same node or on any other node in the namespace. No additional mounts are required by the client to identify the access paths to the data volumes.
2. In the second stage, the client tries to access a file from a data volume for a read or write operation. During this phase the client issues a LAYOUTGET call to the NetApp cluster node that is designated as the metadata server, followed by a GETDEVISEINFO call. Without a read or write operation, the client never requests a LAYOUTGET to the NetApp cluster.

The client follows the following sequence to read or write a file.

1. The client performs an OPEN request to open a file from the data stack of the metadata node.
2. A check is then performed to see if the file requested by the client exists in the corresponding data blade of the cluster node, from the volume location database.
   a. If the file exists in the corresponding data stack, then the client is served locally by the node. This means the same node can function as a metadata server as well as a data server.
   b. If the files exist on another cluster node, then the VLDB on the metadata node provides a layout of the file. The layout has information on the constituent or location of the volume.
   c. The virtual network interface (VIF) or network manager (VIFMGR) on the network stack in the cluster node that functions as the metadata provides information on the network path to get to the node that has the volume. This operation is initiated by the client as a GETDEVICEINFO call. As a reply to this request, the MDS on the NetApp cluster refers the client to the IP address of the LIF where the volume is located in the namespace. The client then directly accesses the LIF that is local to the volume in the data server.

      Note: The metadata server (MDS) can recall the layout at any point in time.
   d. This layout information is cached in the network stack and is also passed on to the client.
   e. Once the pNFS client gets the layout information, it reaches out to the cluster node in the namespace using its local network path where the requested file physically resides. Once the client performs a CLOSE operation on the file, it returns the layout.

For further information on pNFS protocol operations, refer to this section in the appendix.

7 Enabling and Mounting File System over pNFS

7.1 Enable pNFS on SVM

To enable pNFS, enable NFSv4.0, NFSv4.1, and pNFS as shown in the command that follows. Enabling NFSv4.0 is required, since it is responsible for handling the locking mechanism of the files. Additionally, since NFSv4.1 is a minor version of NFSv4, then the underlying NFS structure is required. A valid mechanism is required to map the users and the group information between the client and NetApp storage while mounting a file system over NFSv4.x protocols. Refer to TR-4067 for further details.

```
clus_107 ::> vserver nfs modify –vserver vs1 -v4.0 enabled –v4.1 enabled –v4.1-pNFS enabled –v4-
id-domain tme2k8.iop.eng.netapp.com
```

Verification:

```
clus_107::> nfs show -vserver vs1
                         Vserver:      vs1
              General NFS Access:      true
                          NFS v3:      enabled
                        NFS v4.0:      enabled
                    UDP Protocol:      enabled
                    TCP Protocol:      enabled
             Spin Authentication:      disabled
            Default Windows User:      administrator
             NFSv4.0 ACL Support:      enabled
 NFSv4.0 Read Delegation Support:      disabled
NFSv4.0 Write Delegation Support:      disabled
         NFSv4 ID Mapping Domain:      tme2k8.iop.eng.netapp.com
   NFSv4.1 Minor Version Support:      enabled
                  Rquota Enable:       disabled
    NFSv4.1 Parallel NFS Support:      enabled
             NFSv4.1 ACL Support:      disabled
            NFS vStorage Support:      disabled
```

7.2 Set Up the pNFS Client

Follow these steps to set up the pNFS client:

1. Choose a supported pNFS client. RHEL 6.4 currently is the GA release from Red Hat that officially supports pNFS.
2. Mount the NFS share with `-o vers=4.1`

```
[root@ibmx3650-svl26 /]# mkdir /pnfs

[root@ibmx3650-svl26 ~]#  mount -t nfs -o vers=4.1 172.17.40.171:/pnfs1 /pnfs
[root@ibmx3650-svl26 ~]# lsmod|grep nfs
nfs_layout_nfsv41_files    19480  1
nfs                    415143  2 nfs_layout_nfsv41_files
lockd                   73534  1 nfs
fscache                 53874  1 nfs
auth_rpcgss             44917  1 nfs
nfs_acl                  2647  1 nfs
sunrpc                 260521  11 nfs_layout_nfsv41_files,nfs,lockd,auth_rpcgss,nfs_acl


[root@ibmx3650-svl26 ~]# mount
172.17.40.171:/pnfs1 on /pnfs type nfs
(rw,vers=4,minorversion=1,addr=172.17.40.171,clientaddr=172.17.44.42)
```

```
[root@ibmx3650-svl25 /]# cd /home/root10/mnt/vs1/pNFSpath01

[root@ibmx3650-svl25 pNFSpath01]# ls -al
total 8
drwxrwxrwx 2 nobody nobody 4096 Mar 27 03:33 .
drwxr-xr-x 3 root   root   4096 Mar 27 02:55 ..

[root@ibmx3650-svl25 pNFSpath01]# touch foo
[root@ibmx3650-svl25 pNFSpath01]# ls -al
total 8
drwxrwxrwx 2 nobody nobody 4096 Mar 27 03:34 .
```

```
drwxr-xr-x 3 root   root   4096 Mar 27 02:55 ..
-rw-r--r-- 1 nobody nobody    0 Mar 27 03:34 foo

[root@ibmx3650-svl25 pNFSpath01]# dd if=/dev/zero of=/ home/root10/mnt/vs1/pNFSpath01/foo
bs=1024k count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB) copied, 0.768461 s, 136 MB/s
```

Note: In RHEL 6.4 the file layout driver - "`alias nfs-layouttype4-1 nfs_layout_nfsv41_files`" is enabled by default. If pNFS were to be disabled and only NFSv4.1 used, then this driver could be included in the `/etc/modprobe.d/blacklist.conf` file. This will disable pNFS, but the file system will still be accessed over NFSv4.1 if `-o vers=4.1` is used to mount from RHEL 6.4.

7.3 Check the Status of pNFS on Clustered Data ONTAP 8.2

When the pNFS client communicates the status to the clustered Data ONTAP pNFS stack, the following checks can be performed to validate the sessions:

1. In the following example, 172.17.40.171 serves as the metadata server, and 172.17.40.173 serves as the data server.

```
bumblebee::> set diag

Warning: These diagnostic commands are for use by NetApp personnel only.
Do you want to continue? {y|n}: y

bumblebee::*> vserver nfs pnfs devices show
Vserver Name     Mapping ID      Msid            Mapping Status  Generation
---------------  --------------- --------------- --------------- -------------
vs1               1               2147484676      available       1
vs1               2               2147484694      available       1
2 entries were displayed.

bumblebee::*> vserver nfs pnfs devices mappings show
Vserver Name    Mapping ID      Dsid            LIF IP
--------------  --------------- --------------- --------------------
vs1              1               1028            172.17.40.171
vs1              2               1046            172.17.40.173
```

2. Use the `statistics` option to verify the working of pNFS. LAYOUTGET and GETDEVICEINFO are the two counters that get invoked during a pNFS-related operation.

```
bumblebee::*> statistics start -object nfsv4_1 -instance vs1 -counter *_total
Statistics collection is being started for Sample-id: sample_35

bumblebee::*> statistics show -object nfsv4_1 -instance vs1 -counter *_total

Object: nfsv4_1
Instance: vs1
Start-time: 5/10/2013 11:22:35
End-time: 5/10/2013 11:22:39
Cluster: bumblebee
Number of Constituents: 48 (complete_aggregation)
    Counter                                                     Value
    -------------------------------- --------------------------------
    access_total                                                   14
    backchannel_ctl_total                                           0
    bind_conn_to_session_total                                      0
    close_total                                                     2
    commit_total                                                    0
    compound_total                                               3597
    create_session_total                                            6
    create_total                                                    0
    delegpurge_total                                                0
    delegreturn_total                                               0
    destroy_clientid_total                                         0
    destroy_session_total                                          3
    exchange_id_total                                              6
    free_stateid_total                                             0
    get_dir_delegation_total                                       0
    getattr_total                                                103
    getdeviceinfo_total                                            1 ←
    getdevicelist_total                                            0
    getfh_total                                                   14
    layoutcommit_total                                             0
    layoutget_total                                                1 ←
    layoutreturn_total                                             0
    link_total                                                     0
    lock_total                                                     0
    lockt_total                                                    0
    locku_total                                                    0
    lookup_total                                                  10
    lookupp_total                                                  0
    null_total                                                     1
    nverify_total                                                  0
    open_downgrade_total                                           0
    open_total                                                     2
    openattr_total                                                 0
    putfh_total                                                 1699
    putpubfh_total                                                 0
    putrootfh_total                                               10

Object: nfsv4_1
Instance: vs1
Start-time: 5/10/2013 11:22:35
End-time: 5/10/2013 11:22:39
Cluster: bumblebee
Number of Constituents: 48 (complete_aggregation)
    Counter                                                    Value
    -------------------------------- --------------------------------
    read_total                                                     0
    readdir_total                                                  4
    readlink_total                                                 0
    reclaim_complete_total                                         6
    remove_total                                                   0
    rename_total                                                   0
    restorefh_total                                                2
    savefh_total                                                   2
    secinfo_no_name_total                                          0
    secinfo_total                                                  0
    sequence_total                                              3582
    set_ssv_total                                                  0
    setattr_total                                                  1
    test_stateid_total                                             0
    verify_total                                                   0
    want_delegation_total                                          0
    write_total                                                 1600
```

53 entries were displayed.

7.4 pNFS Improvements in RHEL 6.4

- DirectIO or the O_DIRECT is an OPEN() system call argument in the Linux kernel and is normally used by applications that do not want to use page cache or memory buffers on the hosts running Red Hat Enterprise Linux (RHEL). A good example would be when Oracle Real Application Cluster (RAC) nodes would like to write the new or modified blocks from the Shared Global Area of the database to NetApp storage directly, bypassing the page cache or the memory buffers of the host. This reduces additional buffering on the RAC nodes and improves performance by providing a zero copy mechanism from the application buffer to the disks in the storage subsystem. The application determines if it would use DIRECTIO to communicate synchronously or asynchronously using the POSIX Asynchronous IO (AIO) interface inside the kernel to improve performance.

Prior to RHEL 6.4, the regular READ/WRITE path shared the same code while the DIRECTIO used a different code path. Due to the disparate code path between the normal READ/WRITE and DIRECTIO, the fixes or patches applied to the normal path did not apply to the DIRECTIO path and vice versa. Similarly the DIRECTIO code path was not available for pNFS. It only benefitted applications that used NFSv3 and NFSv4 protocols.

The READ vector (readv) and WRITE vector (writev) are more efficient ways of reading into and writing from several memory buffers compared to normal read() and write() operations. One of the most important characteristics of the readv/writev is to coalesce the vector buffers that are not contiguous into an array and convert them into one buffer and perform a single read or write operation. This eliminates the overhead from the trivial way of performing numerous writes into different buffers; copy them into one block of memory using "memcpy," following with a single write operation. Prior to RHEL 6.4, the process of coalescing vector buffers was unavailable for applications that use DIRECTIO with NFSv3 or NFSv4. This posed issues for hypervisors like Red Hat KVM that use DIRECTIO and coalescing of vector buffers to improve performance. The normal read and write operations that use the regular page cache always had the ability to coalesce the vector buffers into large reads and writes.

In RHEL 6.4, the READ/WRITE code path was rewritten and was merged with DIRECTIO. Now the normal READ/WRITE operations that use page cache and the DIRECTIO path share the same code. Therefore, fixes applied to either one will have a global impact. The change in the READ/WRITE code helped to apply the DIRECTIO feature for applications that use pNFS. By reusing the page cache code, the support for DIRECTIO was achieved for pNFS without having to change the pNFS layout drivers. Now, RHEL 6.4 has the capability to use DIRECTIO on all it data paths as required by the applications using pNFS. The readv/writev buffers are also effectively coalesced in large reads and writes, thereby improving the performance of the applications that use DIRECTIO.

- The Linux kernel has an inherent behavior of tagging GETATTRs to a write operation for consistency for file systems accessed over NFSv4 and NFSv4.1. This is known as a "post-attribute operation." When NetApp clustered Data ONTAP 8.2 receives an NFSv4.x write request, it generates two operations: one for the WRITE and the other for the GETATTR. This proved to be a performance deterrent for applications that run over NFSv4.x. This also impacted the ability to do "zero-copy" operations in clustered Data ONTAP 8.2 for NFSv4.x WRITE requests. This issue is now fixed in clustered Data ONTAP 8.2 and patches are rolled into RHEL 6.4. With the fixes available in RHEL 6.4 a single NFSv4.x WRITE to clustered Data ONTAP 8.2 will perform a single operation by prefetching a GETATTR.

  Clustered Data ONTAP 8.2 now allows "zero-copy" operations for file systems accessed over NFSv4.x by borrowing memory buffers from the network and adding them to the buffer cache. The post-attribute patches in RHEL 6.4 would allow applications sending WRITE operations to clustered Data ONTAP over NFSv4.x using the DIRECTIO path to drop all the GETATTRs. The GETATTRs that are tagged to a WRITE operation are also dropped when delegations are enabled.

  These RHEL fixes are mainly targeted when using DIRECTIO or delegations. If the setup does not have these conditions, the client would perform a WRITE and check for the change attributes. Regular WRITE operations that do not use DIRECTIO or delegations will tag the WRITE with a GETATTR, which would produce two operations. The reason why the fix only applies to DIRECTIO and delegation scenarios is that DIRECTIO bypasses the page cache. Cache consistency is not required, and, in the case of delegations, cache consistency already exists.

- However, these fixes do not affect NFSv3 because post attributes are part of the NFSv3 WRITE operation. NFSv3 performs "zero-copy" operations in clustered Data ONTAP 8.2 and does not send any post attributes like GETATTR for a write operation. pNFS also does not send any post attributes for DIRECTIO for WRITE operations.

8 Best Practices

- Each physical node in the cluster should have at least one logical interface configured. This allows the client to get a direct data path to the volumes residing anywhere in the namespace. If there is no LIF setup on the nodes, I/O requests will get to remote volumes on a nonoptimized path.
- RHEL 6.4 and Fedora 17 are the only clients at this time that support pNFS. The right choice of clients has to be used for pNFS implementation.
- NetApp does not recommend performing a “LIF migrate” operation on the metadata node unless there is a strong requirement to do so. Proper caution must be taken to perform a LIF migrate on the metadata node. With a LIF migrate, a new node will become the metadata server. A proper evaluation of workload and resources on the new node is required before migrating the LIF from the original node if the same LIF is used to mount over other versions of NFS (v3 or v4) from different clients.
- In the clustered Data ONTAP pNFS implementation, every node in the cluster is a metadata and data server. In the case of a metadata-intensive workload, NetApp recommends that different IP addresses from cluster nodes in the namespace be mounted on the client for the application to connect to spread the metadata access across different nodes rather than making a single node in a cluster namespace a bottleneck. Another way to approach this is to use an external round robin domain name server (DNS) or On-Box DNS load balancing, which can resolve a host name into several IP addresses in the namespace. Setting up these features is documented in this KB article: https://kb.netapp.com/support/index?page=content&id=10https://fieldportal.netapp.com/Core/DownloadDoc.aspx?documentID=84837&contentID=10856913801&locale=en_US. This allows the client to mount different IP addresses to spread the metadata load across the nodes in the cluster. In this process the application can access a volume that is local to that node or on a different node, but would still have direct data access.

The command to check the pNFS client for the LAYOUT information is:

```
[root@ibmx3650-svl26 ~]# nfsstat -c
Client rpc stats:
calls      retrans    authrefrsh
2959       0          2959

Client nfs v4:
null         read         write        commit       open         open_conf
0         0% 0         0% 1600     54% 0         0% 2         0% 0         0%
open_noat    open_dgrd    close        setattr      fsinfo       renew
0         0% 0         0% 2         0% 1         0% 6         0% 0         0%
setclntid    confirm      lock         lockt        locku        access
0         0% 0         0% 0         0% 0         0% 0         0% 11        0%
getattr      lookup       lookup_root  remove       rename       link
19        0% 6         0% 2         0% 0         0% 0         0% 0         0%
symlink      create       pathconf     statfs       readlink     readdir
0         0% 0         0% 4         0% 0         0% 0         0% 4         0%
server_caps  delegreturn  getacl       setacl       fs_locations rel_lkowner
10        0% 0         0% 0         0% 0         0% 0         0% 0         0%
exchange_id  create_ses   destroy_ses  sequence     get_lease_t  reclaim_comp
0         0% 3         0% 3         0% 1         0% 1275     43% 2         0%
layoutget    layoutcommit layoutreturn getdevlist   getdevinfo   ds_write
3         0% 1         0% 1         0% 0         0% 0         0% 0         0%
ds_commit
0%
```

- On the storage cluster, run `systat` on individual nodes in the cluster to provide more accurate data traffic information.

  The following is a comparison of traffic when a file system is accessed over NFSv3 and pNFS.

  – NFSv3 data when the volumes are remote.

```
bumblebee::> sysstat -u 1
 CPU    Total      Net    kB/s    Disk    kB/s    Tape    kB/s  Cache  Cache    CP   CP  Disk
        ops/s       in     out    read   write    read   write    age    hit  time   ty  util
 32%     2576    90018  166136       0       0       0       0     0s    97%    0%   -     0%
 33%    2516   91774 166703       0      0       0      0     0s   100%    0%  -     0%
 32%    2482   89035 164300       8     32       0      0     0s    96%    0%  -     2%
 34%    2433   88732 164168       0      0       0      0     0s    99%    0%  -     0%
 33%    2499   85571 164528       0      0       0      0     0s    99%    0%  -     0%
 32%    2503   85095 163010       8     24       0      0     0s   100%    0%  -     2%
 32%    2558   91610 166362       0      0       0      0     0s   100%    0%  -     0%

bumblebee::> sysstat -u 1

 CPU   Total      Net   kB/s    Disk   kB/s    Tape   kB/s  Cache  Cache    CP  CP  Disk
       ops/s       in    out    read  write    read  write    age    hit  time  ty  util
 13%    2451      698  79258       0      8       0      0     2s   100%    0%  -     1%
 19%    2566      736  84913     700   1456       0      0     2s   100%   10%  T     7%
 18%    2639      756  86880       0      0       0      0     2s   100%    0%  -     0%
 14%    2492      659  75231       0      0       0      0     2s   100%    0%  -     0%
 18%    2570      758  86376      24     24       0      0     2s   100%    0%  -     2%
 13%    2538      701  80325       0      8       0      0     2s   100%    0%  -     1%
 18%    2515      704  81529       8      0       0      0     2s   100%    0%  -     0%
 17%    2429      662  75598      24     24       0      0     2s   100%    0%  -     2%
 15%    2653      813  92852       0      0       0      0     2s   100%    0%  -     0%
```

Notice that the NFSv3 traffic on node "clus-01" is double that of "clus-02." This implies that there is direct data traffic from the local volume from "clus-01" and also from the volume on "clus-02" through an unoptimized path.

– pNFS data to volumes on a direct path:

```
clus-01> sysstat -u 1
 CPU   Total      Net   kB/s    Disk   kB/s    Tape   kB/s  Cache  Cache    CP  CP  Disk
       ops/s       in    out    read  write    read  write    age    hit  time  ty  util
 24%    3107     3206  98338       0      0       0      0     0s    96%    0%  -     0%
 21%    3145     3240  99083       8     32       0      0   >60     97%    0%  -     2%
 24%    3097     3251  96651       4      0       0      0     0s    93%    0%  -     0%
 23%    3251     3317 100931       0      0       0      0   >60     98%    0%  -     0%
 19%    3178     3224  98068       8     16       0      0   >60     98%    0%  -     1%
 23%    3190     3324  98691    1360   2964       0      0     0s   100%   11%  T     5%
 22%    3204     3264  99377       0      8       0      0     0s    99%    0%  -     2%
 21%    3211     3294  99376       8     16       0      0     0s    99%    0%  -     1%

clus-02> sysstat -u 1
 CPU   Total      Net   kB/s    Disk   kB/s    Tape   kB/s  Cache  Cache    CP  CP  Disk
       ops/s       in    out    read  write    read  write    age    hit  time  ty  util
 22%    3132     3212  98688       0      0       0      0     0s    98%    0%  -     0%
 21%    3124     3201  98097      24     24       0      0     0s   100%    0%  -     2%
 22%    3151     3268  99570       0      8       0      0     0s    98%    0%  -     1%
 19%    3112     3168  97711       8      0       0      0     0s    99%    0%  -     0%
 22%    3272     3339 103119      24     24       0      0     0s    97%    0%  -     2%
 20%    3082     3137  96475       0      0       0      0     0s    99%    0%  -     0%
 22%    3189     3261 100315       0      0       0      0     0s    98%    0%  -     0%
 19%    3066     3142  97202      24     32       0      0     0s    98%    0%  -     5%
 14%    1787     1822  56049       8      0       0      0     0s   100%    0%  -     0%
```

Where a file system is mounted over pNFS, the data traffic is uniformly spread across two cluster nodes, and the data access paths directly access the volumes. There is no optimized data path.

8.1 NFSv4.x Referrals Versus pNFS

What Are Referrals?

Clustered Data ONTAP introduces NFSv4 referrals. NFSv4 referrals are not available in Data ONTAP 7G/7-Mode. A referral is a mode of I/O path redirection that directs a client to the location of the volumes that it needs to access from a node while the client is mounting from a different node in the namespace. NFSv4 referrals work within the namespace. They cannot allow a client to mount a node from one namespace and communicate with another node in a different namespace. NFSv4 referrals cannot cross namespace boundaries. Referrals are effective when clients mount file systems over NFSv4. The volumes in a namespace are connected to each other by way of a junction path. A client request transparently traverses the junction paths in a namespace to reach to a volume location. This is handled dynamically within the namespace. However, referrals cannot traverse junction paths.

Referrals Versus pNFS

Both referrals and pNFS work toward providing an optimized data path to volumes located in different storage nodes in the namespace. However, pNFS and referrals do not work hand in hand.

With referrals, if a volume moves to another aggregate on another node, the NFSv4 clients have to unmount and remount the file system manually to understand the new location of the volume. Referrals are effective during mounts. Environments that use automounters extensively are an ideal use case for NFSv4 referrals. After a period of inactivity, when the client unmounts and mounts the file system, NFS referrals start to enable clients to provide a direct data path to volumes that are moved to different nodes in the namespace.

Another use case in which NFSv4 referrals can be applied is one in which applications that OPEN and CLOSE constantly but still provide a direct data path to volume. Any web application would be a good candidate for using NFSv4 referrals.

With pNFS, when a volume is moved from one node to another, the client does not have to unmount and remount to access the new location of the volume. The file handle from the original mount is still preserved even after the volume is moved. The locks and the state information of the files are now available in the new volume location. After the volume is moved, the volume location information on the cluster is updated, and no further OPEN request is required from the client to send out the new LAYOUT information. The pNFS stack in clustered Data ONTAP traverses junction boundaries to identify different types of volumes: flexible volumes, FlexCache® volumes, data protection (SnapMirror®) volumes, and load-sharing mirror volumes.

pNFS does not support FlexCache volumes or load-sharing volumes. pNFS and referrals cannot be used both at the same time.

Appendix

Clustered Data ONTAP Terminology

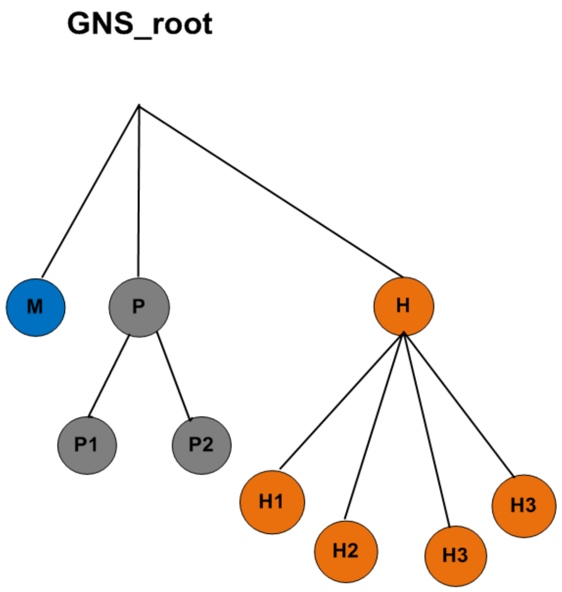
- **LIF.** A logical interface is essentially one or more IP addresses with associated characteristics, such as a home port, a list of ports to which to fail over, a firewall policy, a routing group, and so on.
  - Client network access is through LIFs dedicated to the SVM.
  - A SVM can have more than one LIF. You can have many clients mounting one LIF or one client mounting several LIFs.
  - In case a volume is moved from one node to another within a single namespace across nodes, NetApp recommends migrating the LIF to the new location on the volume to provide more data locality and for load balancing.
- **Storage Virtual Machine (SVM).** An SVM is a logical file system namespace capable of spanning beyond the boundaries of physical nodes in a cluster.
  - Clients can access virtual servers from any node in the cluster, but only through the associated LIFs.
  - Each SVM has a root volume under which additional volumes are mounted, extending the namespace.
  - It can span numerous physical nodes.
  - It is associated with one or more logical interfaces; clients access the data on the virtual server through LIFs.

- **Vifmgr.** The VIF (virtual interface) manager is responsible for creating and monitoring VIFs as well as handling failover of VIFs to other network ports, either local or remote. Data VIFs may fail over to any port in the cluster on either the local node or remote nodes. Cluster and management VIFs may only fail over to ports on the same node as the failure.

Cluster Namespace

A cluster namespace is a collection of file systems hosted from different nodes in the cluster. Each Storage Virtual Machine (SVM) has a file namespace that consists of a single root volume. A cluster can have more than one SVM. The SVM namespace consists of one or more volumes, linked through junctions that connect from a named junction node in one volume to the root directory of another volume. The global namespace is mounted at a single point in the cluster. The top-level directory of the global namespace within a cluster is a synthetic directory containing entries for the root directory of each SVM namespace in the cluster. All volumes belonging to the SVM are linked into the global namespace in that cluster.

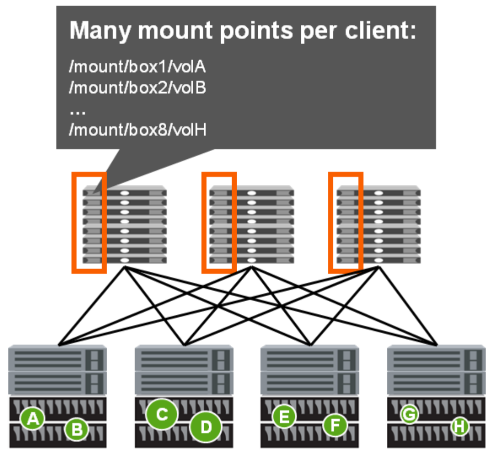
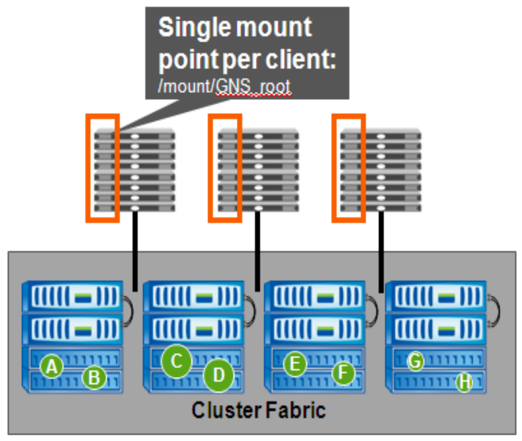
Figure 6) Cluster namespace root.



Benefits of a Global Namespace

Table 1 lists the benefits of a global namespace.

Table 1) Benefits of a global namespace.

Without a Global Namespace	With a Global Namespace
 Many mount points per client: /mount/box1/volA /mount/box2/volB ... /mount/box8/volH A B C D E F G H	 Single mount point per client: /mount/GNS_root A B C D E F G H Cluster Fabric
Change mapping for thousands of clients when moving or adding data	Namespace unchanged as data moves
Difficult to manage	Much easier to manage
Very complex to change	Much easier to change
Not scalable	Seamlessly scales to petabytes

pNFS Protocol Operations

Table 2 lists the pNFS operations.[1]

Table 2) pNFS protocol operations.

Operation	Description
LAYOUTGET	Obtains the data server map from the metadata server.
LAYOUTCOMMIT	Servers commit the layout and update the metadata 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	Client requests the list of all data servers participating in the storage cluster.
CB_LAYOUTREC	

Refer to the Interoperability Matrix Tool (IMT) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.

[1] http://www.snia.org/sites/default/education/tutorials/2009/spring/networking/JoshuaKonkle_pNFS_Parallel_Storage_Grid_Enterprise_Computing.pdf