Name: Baliganahalli Narayana Murthy, Sunil
Course: CSCI 7000 Computer storage systems
Date: Dec 12, 2016

pNFS - Parallel Network File system

Abstract:

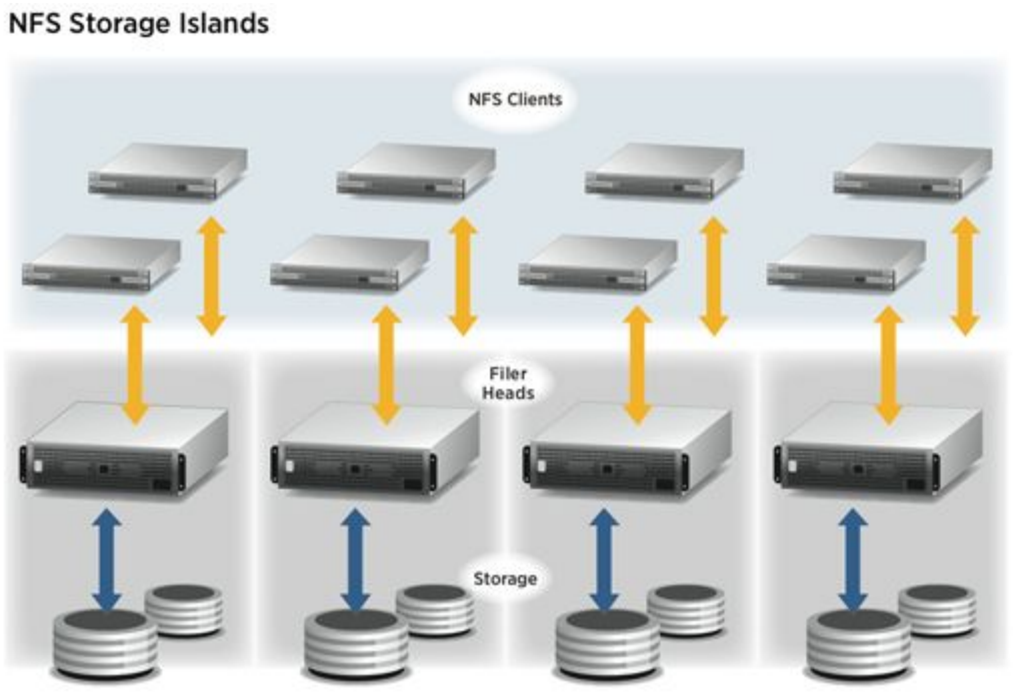
In the recent years, data is growing at an enormous pace. This outburst of information or data abundance has posed some critical challenges like how to store voluminous data to meet the business goals, how to efficiently retrieve it etc. At the same time, it's important to come up with a solution that's cost effective, efficient in performance, low latency, reliable and scales with the growth of data. NFS (network file system) is one of the distributed storage solutions for storing the data on the server. NFS is an effective protocol for storing and retrieving the data from the NAS (Network attached storage). However as the data grows NFS server faces performance, latency, and bandwidth bottleneck issues.

In this paper, I will briefly discuss one of the solution to the above problem for managing storage using pNFS (parallel NFS). pNFS provides a high-performance IO. It was first introduced in NFSv4.1 release to overcome the limitations of NFSv3 and earlier versions by providing direct access to the storage system for NFSv4 clients.

Introduction

In recent years, there has been a tremendous inflow of information. The data is getting generated at a rate of 2.5 quintillion bytes or 2.5 exabytes per day. Around 90% of the total data in the world has been created in last two years [1]. As per International data corporation (IDC), data will grow from 4.4 ZB to 44 ZB by 2020. This explosive growth in the data has resulted in a search for a storage solution that's cost-effective, easy to manage, low latency while retrieving the data(reads), good performance (reads and writes over the network) etc. Some may say why not use Network file system (NFS) protocol. However, NFS does not scale out but only scales up with certain limitations. To understand why NFS is not a good fit for high-performance IO we should understand a bit of history about NFS first. The NFS was first introduced in 1986 by Sun microsystems and has been a great success ever since. The IETF standards have been continuously changing the specifications of NFS to meet the changing requirements of the industry. NFS was intended for sharing files across networks of workstations.

In traditional network file system, architecture consists of a filer head which is placed in front of a storage media (spinning disks, tape etc). Filer's head exposes the filesystem via NFS. An NFS clients can now mount the remote storage usually called network attached storage (NAS) as if it were a local media. In NFS all the requests have to go through a single server. Thereby funneling the I/O requests through a single NFS server. The problem arises when a large number of clients want to access the file systems or the data set becomes enormous. This funneling of the IO requests results in overloading the NFS server and network bandwidth. This could significantly hurt the performance in a high IO systems since NFS server sits in between the NFS client and storage media.



Courtesy: Pansas [3]

One way to fix the NFS limitation is to avoid NFS server getting in the data path between clients and storage. And let the NFS client direct access to the storage devices. This idea led to the start of Parallel filesystems like pNFS, Lustre, GFS, IBM GBFS. So, this solution scales out effectively, does not hurt the performance in high IO bound systems, and no intermediate devices when client wants to access data from storage device.

Background

There are a lot of parallel filesystems like IBM GBFS, Google’s GFS, etc. But all of these are proprietary - the clients have to pick a vendor and use their filesystem thereby causing vendor lock-in issues. Clients won’t be able to migrate or use other file systems without incurring a loss of capital. This conflict led to the development of standards for a parallel filesystem. The IETF incorporated all these new concepts and extended the capabilities of NFS to create what's known as pNFS standards.

NFSv4 also addressed some of the limitations of NFSv3 through compound operations and the use of delegations to improve the client-side caching. However, the single server problem wasn’t addressed. With NFSv4.1, single server bottleneck issue was addressed by introducing parallelism in IO.

pNFS Overview

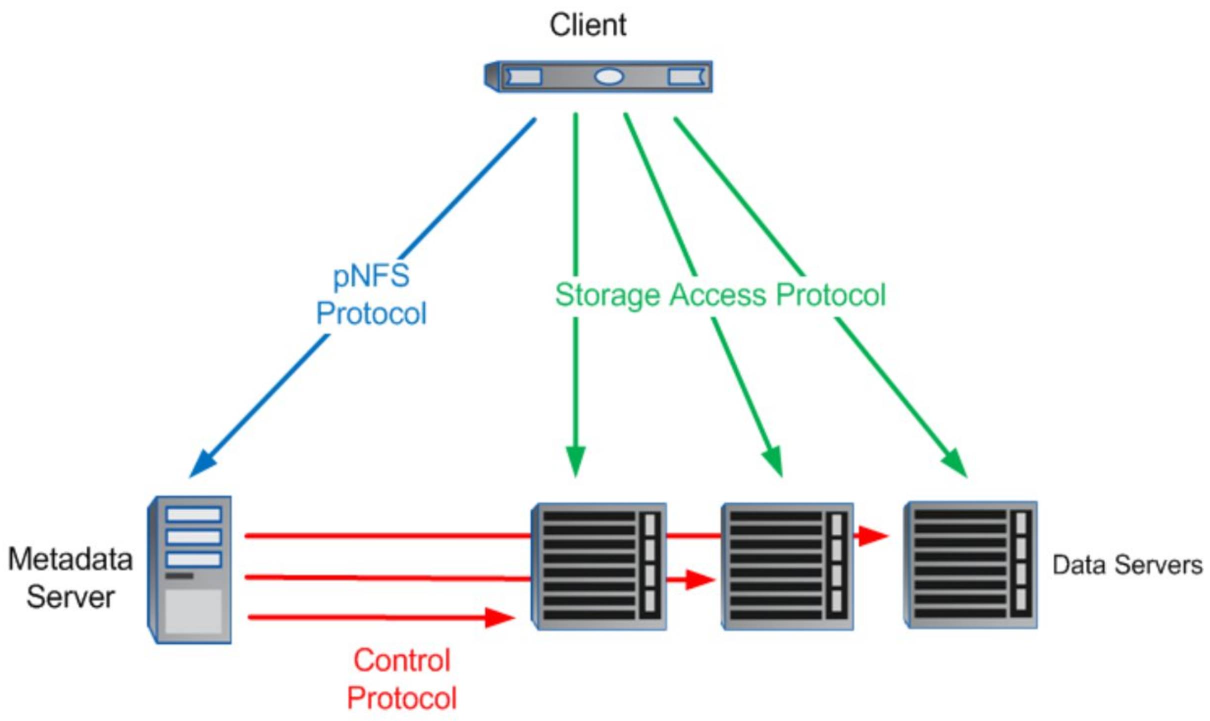
Parallel NFS is a subset of the standard NFS version 4.1. NFSv4.1 that follows the request for comments(RFC) 5662 is a minor release of NFSv4. It defines a way for the clients to directly communicate with storage media for reads or writes. This is precisely the same idea behind almost all parallel filesystems like Lustre, GFS, PanFS etc. The pNFS follows the same syntax as the earlier good old NFS versions, the standards are open and the vendor can write the implementation by referring to the NFS v4.1 specifications.

pNFS which is a part of NFSv4.1 was designed with the intention of seamless integration with the existing application or operating systems. With traditional NFS versions 3, and v4 where the metadata and data are shared on the same IO path. Whereas in version 4.1 (pNFS) the metadata and data are shared on a different IO path. This eliminates the funneling of reads and writes. Metadata server is responsible only for handling all the metadata activities from the NFS v4.1 client and data server provide direct access to the storage media (data) to the clients. pNFS has three different implementations - file, object, and blocks.

In addition to support for parallel IO and scalability, few other notable new features were introduced with NFSv4.1 like - a single well-known port to handle all operations like open, close, locking, resource quota management - which eases the deployment process, aggregating multiple operations using *COMPOUNDS* procedures, client side caching using delegations - to cut down the number of requests sent to server, support for TCP-only transfer protocol (NFSv2 and v3 supported UDP-only), sessions, stateful servers etc.

pNFS Architecture

This section briefly discusses the architecture of pNFS and various components involved.



pNFS block diagram

Courtesy: NetApp [7]

The main components in the pNFS architecture are -

- Metadata server (MDS): pNFS standards defines the NFSv4.1 protocol between clients and NFS metadata servers. MDS handles all metadata information and is responsible for handling the NFS/pNFS operations like GETATTR, LOOKUP, OPEN, CLOSE etc. MDS also sends control protocol to manage the data servers.

- Data servers: Data servers are responsible for storing the client data. These data servers respond directly to the clients read or write operations.

- Clients: Clients retrieved the metadata information from the MDS. Based on the information received they are able to access the data servers directly.

There are three types of protocols in action between MDS, data servers, and clients.

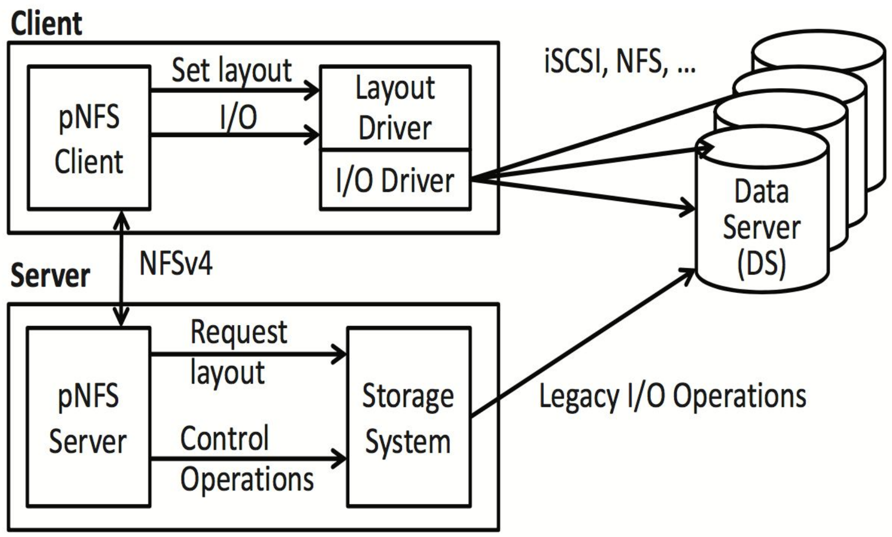
- Control protocol: Control protocol is a management protocol used by MDS to manage the storage devices to synchronize the file system data. Control protocols manage activities such as allocation and deallocation of storage, management of states required by the storage devices to perform client access control. Control protocols depend on the type of protocols used by the storage devices such SCSI object-based storage device (OSD) over iSCSI, NFS.

- pNFS protocol: Is used between clients and MDS. It's same as NFS protocol with a few addition of pNFS-specific operations. It's used to retrieve and manipulate the metadata information. This metadata defines the storage location and the protocol to be used for communicating with a specific data server. For eg: SCSI object-based storage device (OSD) over iSCSI, Network file system(NFS), SCSI block commands over Fiber channel (FC).

- Storage access protocol: Is used by NFS clients to access the data servers directly. pNFS specification supports three storage protocols - File storage, block storage, and object storage.

pNFS Layout

The following diagram shows detailed interactions happening between server, client and data servers. The pNFS server lets the clients directly communicate with the storage devices by transferring the metadata information of the file(s) - also known as the layout information. The layout describes the file name, owner of the file, how the files are distributed in the storage, how it is striped across multiple drives, access control lists(ACL) and other file attributes. The layout also contains the storage access protocol which specifies the protocol to be used while communicating with a specific storage media. MDS is responsible for managing these layouts, support file mirroring and striping. MDS returns the layout to the client upon request however, it may also perform actions like recall, revoke, modify the layout when required.



pNFS detailed Architecture   Courtesy: [2]

The IETF standards committee for pNFS defines three storage access protocols - File, block and object storage such as SCSI, iSCSI, FoC etc. However, the vendor’s are free to add their own storage access protocols.

File storage - File storage is implemented using traditional NFS protocol such as SMB/CIFS etc. Here the files are organized into folders and subfolders. This used in network-attached storage where file sharing is required. The MDS layout information contains the list of servers that hold the pieces of the files. A file layout type would contain an array of tuples of device ID and file handle.

Block storage - Block storage is used in storage attached network (SAN). Here the files are treated as data blocks and these blocks are striped across the drives. Layout information maps the file blocks to the physical blocks on the drives. A block storage layout type would contain information such as device Id, block numbers, block count and size. Blocks storage protocol usually contains the SCSI block command set.

Object storage - Unlike file or block storage, object storage is not organized into files or folders. Every object contains data and exists at the same level. An object layout type is an array of the tuple <device ID, object ID> and aggregation map. Protocols that support object storage are OSD v1, OSD v2, OSD over iSCSI or fiber channel etc.

When a pNFS client wants to access a file on the storage, it first retrieves the metadata/ layout information from the MDS. Once it receives the layout, the layout driver on the pNFS clients selects a specific driver that supports the protocol understood by the storage server to initiate the writes or reads.

pNFS Operations

All NFS operations like *LOOKUP, GETATTR* etc are supported in pNFS as well. In this section, we will look at some of the pNFS specific operations.

GETDEVICEINFO: Every storage device has to be uniquely identified to access it. The pNFS layout contains only the device ID however it does not identify the device uniquely. We also need information like LUN (logical unit number), volume label and others. Since the layout doesn't contain the full identity of the storage, a new operation called *GETDEVICEINFO* was introduced in pNFS. This operation retrieves the complete identity of the storage device like IP address, port, file server etc.

GETDEVICELIST: This operation retrieves the mappings of multiple storage devices associated with a metadata server.

LAYOUTGET: The clients need the metadata information to access the storage devices directly. They can make use of *LAYOUTGET* to get pull this information from the MDS. Clients can also provide information to the MDS with *LAYOUTGET* operation such as specify a sharing mode - shared or exclusive. In a shared mode - other clients can access the files, whereas in an exclusive mode only one client has the permission to read/ write to the file.

LAYOUTCOMMIT: The layout commit operation indicates that the client has completed writing to the file using the layout retrieved from MDS. This operation commits data to the storage device and updates the layout information on metadata server so that the changes are visible globally.

SETATTR / GETATTR: The client can set or get parts of the information from the entire layout using *SETATTR* or *GETATTR* respectively. For eg: To set the owner of the file, modify the access time of the files, change access control list etc.

LAYOUTRETURN: When the clients are done accessing the file(s). They can give up or return the layout information to the MDS using *LAYOUTRETURN* operation. After this operation, the clients should not access the storage devices with a cached copy of layout information. If they want to access it again, they should do so by getting the latest copy layout using *LAYOUTGET*. Layouts can be returned by the clients - voluntarily using *LAYOUTRETURN* or recalled by the server using *CB_LAYOUTRECALL*.

pNFS Callback Operations

CB_LAYOUTRECALL: This operation recalls the layout or all layouts for a given file. This command forces the clients to relinquish their control over the specified file. The clients will return layout to the server using *LAYOUTRETURN*.

CB_SIZECHANGED: Indicates the clients that the file size associated with the specific file handle has been changed. Upon receiving this notification, the client(s) should update their layout information.

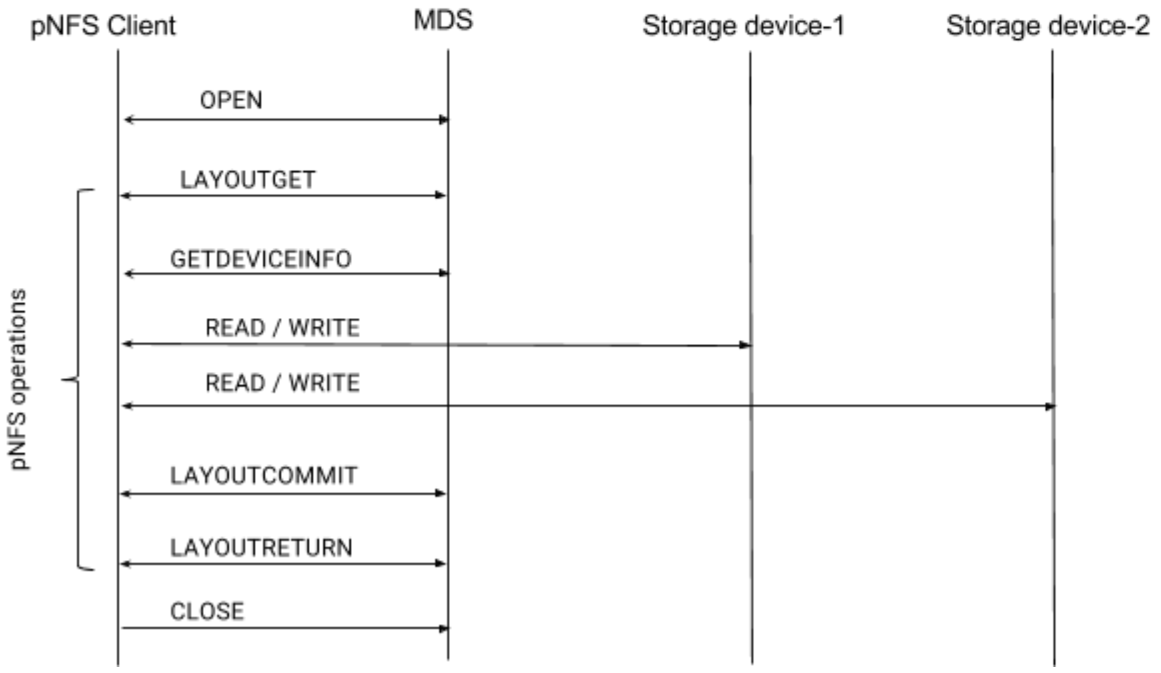
CB_NOTIFY_DEVICEID: This operation will notify the clients about the changed device mappings.

CB_RECALLABLE_OBJ_AVAIL: Informs the previously denied *LAYOUTGET* requested client, that the layout is now available.

pNFS workflow

This section briefly describes the workflow of pNFS. To access a file on the storage device, the client should open the file using NFS OPEN operation on the MDS. The MDS will check if the file exists on the devices if the requested client has access to the file etc. The client can get the layout information using LAYOUTGET after successful OPEN operation.

After receiving the layout, the client will get the full identity of the storage device using GETDEVICEINFO. The client will select an appropriate layout driver to translate the reads / writes from pNFS client to the reads / writes understood by the storage devices.



pNFS workflow - Sequence diagram

The above diagram shows the pNFS workflow as a sequence diagram. The OPEN and CLOSE are regular NFS operations, and everything else in the diagram is pNFS operations. When the client is done accessing the file, it can update the MDS layout information using LAYOUTCOMMIT and return its layout to the server using LAYOUTRETURN. After layout return the client can close the file, making the layout available for the other clients.

Performance/ Experimental results

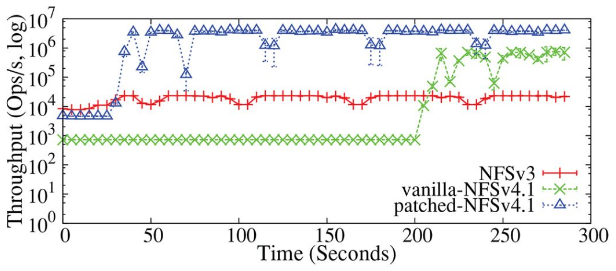
Here we will briefly look at the performance aspects of NFSv4.1 (pNFS) in comparison with its predecessor, NFSv3. The experimental results cited in this section are original work of Ming Chen, Dean Hildebrand, Geoff Kuenning, Soujanya, Shankaranarayana, Bharat Singh, and Erez Zadok. The main theme of the paper [11] is to perform a comprehensive performance comparison between stateless-NFSv3 server and stateful-NFSv4.1 server, under varying workloads and latency.

The experimental setup consists of one server and five client machines of identical hardware and software configurations [12]. They used BenchMaster, a benchmark framework that generates random workloads concurrently. It also collects system statistics using iostat, vmstat and reads procfs entries. To simulate real network loads over short and long distances, a delay of 1ms - 40ms was injected into the network.

Throughput

For data-intensive workloads that operate on one big remote file, both NFSv3 and NFSv4.1 performed well with almost similar throughput. However on a metadata intensive workloads that operate on a large number of small files - pNFS performed poorly on startup. This was due to a system bug in Linux pNFS implementation. The bug was fixed with a patch, resulting in an improved overall performance (shown in triangles in Fig-1). In the graph Fig-1, a little after the 40 seconds there’s a big boost in the throughput. This was due to the pNFS delegation feature (client side caching) which reduces the number of requests sent to the server.

In their experiment, they observed a “Winner-loser” pattern for both NFSv3 and v4.1 caused by the server’s use of multi-queue network interface card (NIC). The clients were usually clustered into winners and loser. Where the winners get roughly twice the throughput of the losers.



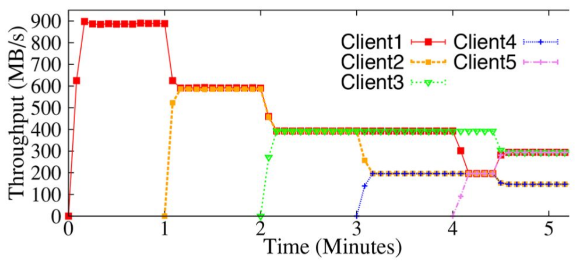


Fig-1: Aggregate throughput of 5 clients in 10ms delay   Fig-2: Individual throughputs of 5 clients

In Fig-2, the clients are started sequentially with 1-minute delay. It’s evident from the graph that client-2 started at second ended up as loser and client-5 started last ended up as a winner. This reveals that there exists no correlation between client start order and it’s chances of ending up as a winner or loser.

Statefulness

This test characterizes the effect of introducing states to pNFS server. Fig-3 shows the number of requests made to both NFSv3 and v4.1 server on a zero and 10ms latency network. From the results here, it’s obvious that NFSv4.1 requires more interactions with the server to maintain the states thereby reducing the throughput. To counteract the effect, NFSv4.1 takes advantage of *COMPOUND* procedures and *DELEGATION*.

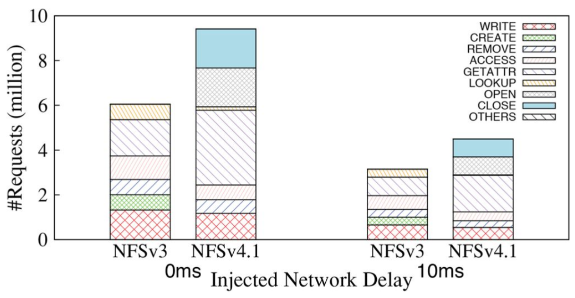


Fig-3: Number of requests to NFS servers

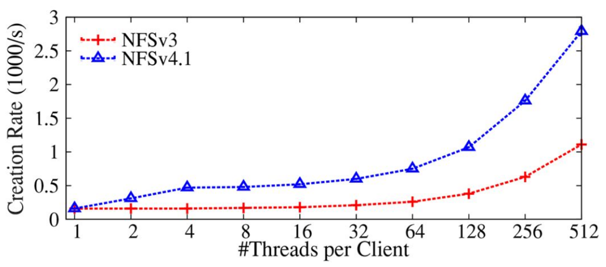
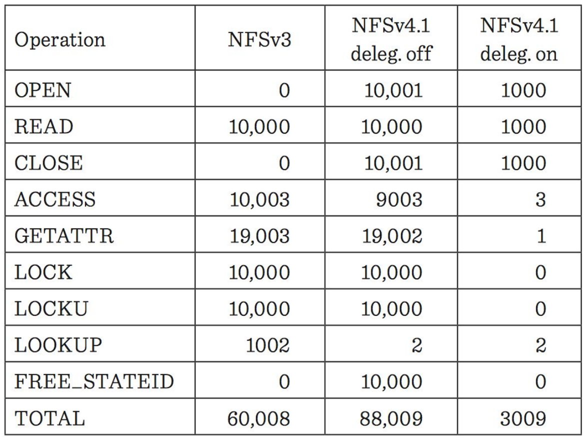


Fig-4: Empty file creation rate

Fig-4 shows the performance graph of creating an empty file with increasing number of client threads. pNFS outperforms NFSv3 due to the fact that pNFS server maintains states, this allows network layer to combine multiple RPC messages.

Delegations

To reduce the number of requests sent to the pNFS server, client side caching was introduced. In NFSv4.1, the server delegates the file to a particular client for a limited time. During this time, the client need not revalidate the file attributes or contents. Though this will improve performance most of the time, it will incur an expensive cost if there are any conflicting file operations.



Operation	NFSv3	NFSv4.1 deleg. off	NFSv4.1 deleg. on
OPEN	0	10,001	1000
READ	10,000	10,000	1000
CLOSE	0	10,001	1000
ACCESS	10,003	9003	3
GETATTR	19,003	19,002	1
LOCK	10,000	10,000	0
LOCKU	10,000	10,000	0
LOOKUP	1002	2	2
FREE_STATEID	0	10,000	0
TOTAL	60,008	88,009	3009

Fig-5: Operations performed by NFSv3 and v4.1 clients.

The fig-5 shows the number of NFS operations performed by NFSv3 and v4.1 clients with the delegation on and off. With delegations, only OPEN, READ, and CLOSE operations interacts with the server and all other operations can be processed locally until the leased time expires.

Conclusion

pNFS is an ideal solution for distributed clusters with high-performance requirements like HPC storage servers and others. By separating the data path and metadata path, pNFS gains network bandwidth and concurrency. pNFS being a subset of NFSv4.1, which makes it backward compatible with the preceding version of NFS such as v3 and v2. What this means is that the existing clients, the user application can gain performance boost without any changes. Like previous versions of NFS, pNFS is interoperable with a variety of operating systems such as - FreeBSD, Linux, MacOS, Windows etc.

Future of pNFS

NFSv4.2 is being drafted by the IETF. NFSv4.2 addresses some of the shortcomings of pNFS in NFSv4.1. Businesses have already started placing a request to improve the network bandwidth, processors, storage capacity etc. Solid state flash storage is emerging and is much faster than the rotating magnetic disks. pNFS can incorporate SSD's to the data server to increase the performance, lower the latency.

References:

[1]. Storage tends File & Object based storage by IBM

[2]. pNFS-based Software-Defined Storage for Information Lifecycle Management by Kuo Sheng Deng, Chin Feng Lee, Jerry Chou, Yi Chen Shih, Shang Hao Chuang, Po Hsuan Wu.

[3]. pNFS standards & Latest Developments by Brent Welch from Panasas

[4]. Overview of NFSv4 by SNIA, ethernet storage forum.

[5]. Parallel NFS (pNFS): The war is close by Konstantin S. Solnushkin

[6]. A Brief History of pNFS by Garth Gibson

[7]. Parallel Network File System Configuration and Best Practices for Data ONTAP 8.1 Cluster-Mode by Amrutha Naik, Bikash Roy Choudhury, NetApp

[8]. pNFS.com by Panasas

[9]. pNFS operations by IETF - Internet draft

[10]. pNFS block/volume layout Request for comment : 5662 - Internet draft

[11]. Linux NFSv4.1 Performance Under a Microscope by Ming Chen, Dean Hildebrand, Geoff Kuenning, Soujanya, Shankaranarayana, Bharat Singh, and Erez Zadok.

[12]. Is NFSv4.1 Ready for Prime Time?