pNFS

Parallel Network File System

Author:
Thijs Stuurman

email: Thijs.Stuurman@os3.nl

Supervisor:
Jan Meijer

June 30, 2008

UNIVERSITEIT VAN AMSTERDAM

Masters program System and Network Engineering

The Norwegian research network

Abstract

Parallel NFS may become the standard as NFS has long been. As networks, storage and the need for bandwidth has grown over the years NFS has been left behind. With pNFS being standardized by the IETF and development investment and support from major players in the field of storage solutions, problems faced by large scale distributed storage platforms may be solved. Therefore UNINETT is interested in pNFS for the NorStore project. This report gives an overview to make clear what pNFS is, how it works and how it might be used. For this the pNFS IETF drafts, available source code and a test implementation have been used. In the current state, pNFS will not be ready for production use for the coming year but will certainly change storage platforms in the future.

1 Acknowledgments

I would like to thank Jan Meijer, Maurits van der Schee and JP Velders for their assistance during my research. A special thanks goes out to everybody on the pNFS mailinglist for answering questions and their hard work in general on pNFS.

2 Preface

This report has been written as part of my study *System and Network Engineering* at the University of Amsterdam. Two research projects are done throughout the year of which this is the second. I researched pNFS which is an extension on NFSv4 and which is still being drafted. Jan Meijer defined and supervised the project, he works for UNINETT which is the Norwegian NREN (National Research and Education Network). The work was done in the Netherlands at the university lab.

Contents

3 Introduction

Computer systems are not able to keep up with the demand for resources such as CPU power and storage I/O. To solve most of these problems computer systems are being clustered together. Networked storage I/O has long been dominated by the NFS (Network File System) RPC (Remote Procedure Call) protocol which has been originally developed by Sun Microsystems. NFS is an open standard and any vendor can create their own implementation which is then interoperable with others. Computing environments which are using the NFS protocol are being limited by the single NFS head which creates a bottleneck between fast storage and large amounts of computing power. Several proprietary alternatives such as IBM's GPFS[8] and Panasas[18] PanFS have been built to make I/O available to clusters using multiple connections to eliminate the bottleneck. Recent developments with NFSv4[9] have brought forth pNFS (Parallel Network File System)[19]. This protocol is being standardized as NFSv4.1[10]. pNFS provides clients with scalable end-to-end performance and the flexibility to interoperate with a variety of clustered storage service architectures. Because pNFS is an open standard there is a lot of interest from the High Performance Computing community and scientific institutes who deal with large amounts of data on a daily basis. Implementations of pNFS are currently being developed and tested.

In Norway they are currently working on a new national distributed storage platform, the NorStore project[31], which will be used for science and research of natural sciences[32]. The infrastructure will consist of heterogeneous computing systems, networks and storage systems. Large amounts of data will be moved and pNFS has been created to make it possible to transfer large amounts of data at high speeds making it a topic of interest.

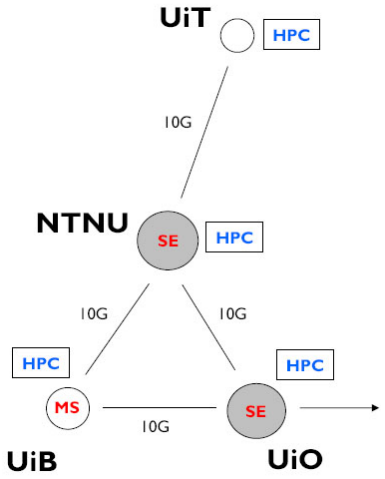

Figure 1: The initial NorStore network overview.

Figure 1 shows the initial setup in which the following parties were involved:

- UNINETT
- UNINETT Sigma
- UiO (University of Oslo)
- NTNU (Norwegian University of Science and Technology)
- UiB (University of Bergen)
- UiT (University of Tromsø)

As can be seen in figure 1, sites are connected using high bandwidth, high latency data links of 10 gigabit. The first two storage elements each hold 600 terabytes of storage and this shows just the beginning. Large amounts of data need to be moved for which optimal data path utilization is a must.

3.1 Research questions

The first week of the project consisted of reading up on related material and planning for the following three weeks. The main research question inferred from the original research description is as follows:

> Is pNFS capable of transferring the large amounts of data required in the NorStore context?

To answer this question, several sub questions were defined:

- What is and how does pNFS work?
- What are the capabilities of pNFS?
- What are the limitations of pNFS?
- How does pNFS compare to proprietary solutions?
- What is the current state of the pNFS development?
- How can the NorStore project utilize pNFS?
- How does pNFS behave in a number of scenarios?

The hypothesis was as follows:

> pNFS will be capable of transferring large amounts of data but is currently still missing good integration with underlying file systems and under heavy development. Therefore it is currently not ready for production use.

To answer these questions and analyse pNFS, both the draft 23[23] of NFSv4.1 and a PoC (Proof of Concept) based on Linux with spNFS (Simple pNFS)[2] have been used. Because the NorStore infrastructure will be used for a variety of uses such as HPC, astronomy, biology, chemistry, earth science and physics work for which both large and small files are relevant. Based on those uses, the following scenarios have been used to test the behaviour of pNFS:

1. Transferring large amounts of small files from and towards the client.
2. Transferring large files from and towards the client.
3. Transferring data through different administrative network domains.

3.2 Scope

- The research is limited to the specifications of pNFS as defined in the NFSv4 minor version 1 by the IETF[23]. Current implementations and PoCs will only be used to gain experience and insights but will not influence any conclusions which are specific to the pNFS protocol.
- Aspects such as details about the underlying file systems and security will not be examined.
- A PoC will be used to test an implementation of pNFS to get a better understanding of the protocol and its current development and implementation status.
- The PoC has been setup using a local private Ethernet network with speeds up to 200Mbit. There were no available resources to test using a high speed, high latency Internet connection.

3.3 In this document

The next section describes related work. Section 6.1 and 6.2 gives insight on what pNFS is. Section 6.3 will explain how pNFS works which include the scenario testing. Section 6.4 gives and overview of who works on pNFS, how far development has progressed and what is to come or expect in the future. Section 8 describes why and how pNFS is of use when it comes to the NorStore context. This document then ends with a conclusion and a description for future work.

4 Related work

Dean Hildebrand *et al.*[7] conducted a study as to how pNFS behaves and performs with large files and small writes. An early prototype of pNFS was used and introduced a write threshold to overcome the parallel file system inefficiencies. Dean Hildebrand *et al.*[5] published a paper in 2007 about the pNFS architecture using PVFS2 and GPFS. Their early benchmarks have shown promising bandwidth use results and interoperability. Unfortunately, the PVFS2 layout is no longer supported by the current pNFS software. Dean Hildebrand *et al.*[4] conducted a study on using pNFS across the TeraGrid[29] WAN (Wide Area Network) where clients accessed GPFS back-end storage. They concluded that pNFS achieved very impressive read performance but write performance have yet to be demonstrated. GPFS is proprietary software and for this report updated pNFS drafts and software along with other back-end software has been used.

5 Background

This section will describe some of todays storage solutions. Storage environments can be set up using a variety of techniques such as a NAS (Network Attached Storage) or SAN (Storage Area Network) with software solutions as NFS and specialized file systems.

5.1 Current storage solutions

There are two forms of storage solutions available, namely distributed and parallel file systems. Distributed file systems are single file systems which are shared through a network to many users. Parallel file systems make use of multiple storage servers to stripe data to gain higher performance and fault tolerance.

An example of a simple distributed file system would be a single server running Linux on a hard disk partition using the Ext3[1] file system which then exports[2] part of it using NFS to other users on the network. This setup is able to scale to the extend of the network throughput and system load of this single server. The storage can be expanded to as many disks the server can accommodate and connect to its I/O busses. More expensive options include adding external storage through perhaps another private network. This means that even several extra servers may export their storage through NFS towards the server which in turn exports the whole to the network users. An example of a typical simple NFS setup is depicted in figure 2.

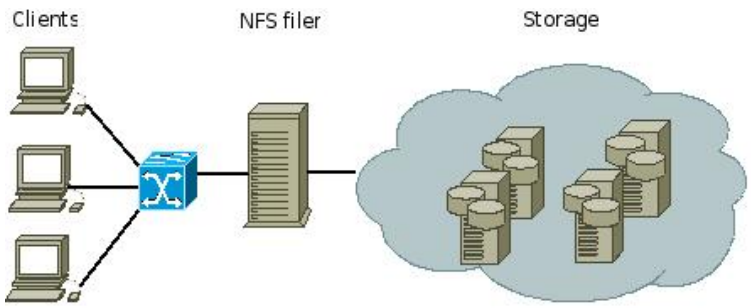

Figure 2: Single NFS head-filer example.

There are two big draw backs to this kind of setup:

- Single point of failure, the head-filer
- Single data flow through point, the head-filer

There are cluster setups which use several head-filers with each their own storage space between which users may be divided to compensate on the drawbacks. These head-filers are aware of each others data and interconnected using a private network as depicted in figure 3. When a user connected to head-filer *A*

[1] Third extended filesystem, a default filesystem used by many Linux distributions.
[2] Using NFS, the transfer of the filesystem over the network to other machines.

requests data which is actually located at head-filer B, A will retrieve the data through the private network from B and then relay it to the client. Tuned for a specific environment, this kind of setup scales well beyond what a single head-filer could have handled. The draw backs however still exist. Clients still communicate with a single head-filer and when a head-filer goes offline, data for which it was responsible will not be accessible or even lost. Data loss may be prevented using RAID (Redundant Arrays of Inexpensive Disks) solutions but the data will still be inaccessible.

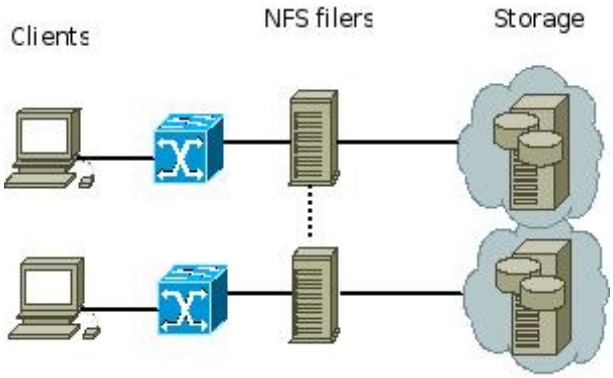

Figure 3: NFS cluster setup example.

Certain environments require even more, namely HPC (Higher Performance Computing) demands low latency and very much bandwidth and high availability. A single head-filer simple does not have the memory, CPU or I/O bandwidth to meet these demands. These environments may consist out of hundreds to thousands of nodes demanding access to terabytes or even petabytes of data. To accommodate these requirements clustered parallel file systems are built. These file systems are parallel in the way that separate data storages are used to form one namespace. Files written to this namespace may be striped, mirrored or a combination of these kind of RAID techniques over the available storage locations. Clients are able to directly contact these storage locations in parallel. This allows the available storage and bandwidth to grow significantly as more storage nodes can be added and these can then directly be used by the clients.

A few examples of a parallel file system are IBM's GPFS[8], EMC MPFS[6] , Lustre[28], Panasas PanFS[17] and PVFS2[21]. These are the actual file systems unlike NFS which is more like a virtual file system. Note that these are all different implementations of a parallel file system with each their own client software for parallel access. Some of these systems are closed source, proprietary, and require licenses. All of these however do support access using NFS or CIFS (Common Internet File System) through a single gateway machine.

6 Research

This section gives more detailed information on pNFS. Information has come from both the pNFS draft as using the PoC with utilities such as `tcpdump`[3] and wireshark[4] with a pNFS patch which can be found in the linux-nfs[13] git[5] repository.

6.1 What is pNFS

pNFS is an optional protocol extension for NFSv4, NFSv 4.1, which is currently in the form of an IETF (Internet Engineering Task Force) draft. Regular NFS operates in a serial fashion where all data flows through a single NFS front-end server. NFS it self is not a very complicated protocol as it is designed to effectively show a virtual file system to another computer over the network. It is not tied to a particular real file system which the server would use such as Ext3. Any directory hierarchy on the NFS server can be exported to clients over the network, even if that particular part comes from another back-end storage system.

pNFS separates the directory hierarchy and the back-end storage. To be more precise, it separates the metadata and the actual file data. Like NFS the pNFS server still exports a part of its directory hierarchy. A pNFS enabled client operates on the directory hierarchy and filenames just as in a regular NFS situation. When actual file data I/O is requested by a client it will receive information in the form of a layout from the pNFS server called the MDS (Meta-Data Server). These layouts may be heavily cached by the client for improved performance. The draft does not mention anything on scaling the MDS itself, clustering metadata or high availability setups but these features can still be implemented. The information in a layout is then interpreted using a layout driver to translate for example specific file data byte ranges in to I/O requests specific for the back-end storage. The client then uses an I/O driver to access the back-end storage DS (Data Server) directly which can then be done in a parallel distributed over multiple DSs. Figure 4 gives an overview of how a pNFS setup would be like. Several pNFS clients communicate with the MDS and then talk in parallel to the back-end storage servers. Whenever needed, regular NFS can be used through the MDS.
The layout driver may be supplied by the vendor of the specific back-end system but several drivers are already available. More information on the layouts is available in section 6.4. As there are multiple layout drivers available, a client may load several at once to communicate with different storage platforms in parallel making it interoperable.

[3] A network data packet capture program.
[4] A network data packet analyzer.
[5] A distributed revision control / software code management project created by Linus Torvalds, initially for the Linux kernel development. (Source: Wikipedia)

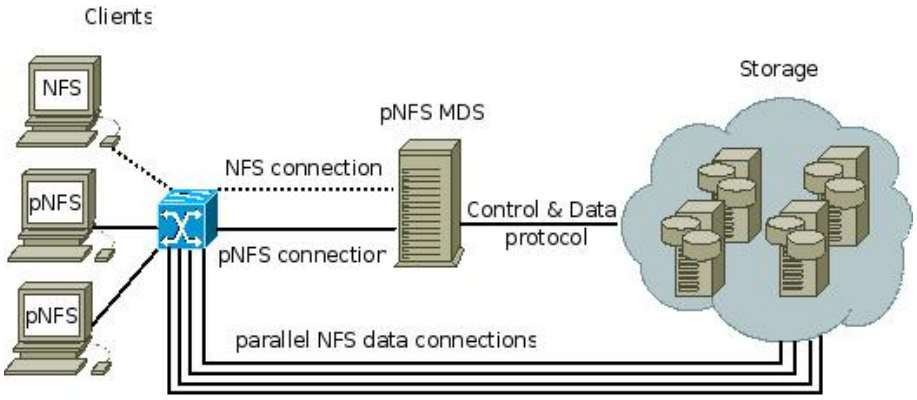

Figure 4: pNFS setup

6.2 pNFS's role

As mentioned before, NFS is the *only* standard network file system. With the addition of pNFS there is a chance to standardize parallel network file systems. However, this is an open and free standard for which a company can not ask money when being used with their parallel file system solution. Why would they, as competitors, support this new standard? A quote from the Chairman of Nokia's Board of Directors[33] sums it up quite nicely:

> Open standards and platforms create a foundation for success. They enable interoperability of technologies and encourage innovativeness and healthy competition, which in turn increases consumer choice and opens entirely new markets.

Companies involved with the development of pNFS are some of the industries leading parties such as Sun Microsystems (Solaris, original NFS developers), IBM, EMC, NetApp[14] and Panasas. Infact, large portions of the Linux implementation of pNFS is based on Panasas's own proprietary DirectFLOW[16] product which shows how big the backing by the industry is. CITI (Center for Information Technology Integration) which is a research and development center in the School of Information at the UMICH (University of Michigan) is also a huge drive behind drafting and standardizing NFSv4.

A likely future scenario will be where clients will access an array of different storage solutions in a serial and/or parallel fashion using open source standardized well developed and interoperable software, NFS and pNFS. Both because it is *required* that regular NFS can access the export through the gateway indirectly for continued interoperability.

6.3 How pNFS works

A number of different scenarios can be made using pNFS as the front-end towards the client(s). Figure 4 shows the outline of a pNFS setup in general. The following sections will describe which layouts are available for communication

with the back-end storage and what exactly happens when a pNFS mount[6] is made.

6.3.1 Available layout types

Currently the following layout types have been defined and are available:

- 1 `LAYOUT4_NFSV4_1_FILES`
- 2 `LAYOUT4_OSD2_OBJECTS`
- 3 `LAYOUT4_BLOCK_VOLUME`

The numbering of the file system layout types will be maintained by IANA as described in the pNFS draft. Currently these are not yet defined at IANA.

The `LAYOUT4_NFSV4_1_FILES` is used when the back-end storage system consists of DSs which can speak NFSv4. This layout is the default and is available as part of the pNFS draft.

The `LAYOUT4_BLOCK_VOLUME` is written by engineers from EMC based on their own distributed file system known as EMC Celerra HighRoad or MPFS (Multi-Path/Multi-Protocol File System). It defines how pNFS should communicate with block/volume devices using protocols such as fiber channel or iSCSI.

The `LAYOUT4_OSD2_OBJECTS` is written by engineers from Panasas to enable pNFS to communicate with an OSD (Object Storage Device). According to the draft, this layout is even aware of RAID and supports numerous algorithms. A layout driver for Panasas OSDs is available in the NFS git repositories.
With these three layout types, pNFS clients will be able to speak to a wide range of storage solutions. The `LAYOUT4_NFSV4_1_FILES` is being used to speak to DSs using NFSv4.1 which is being used for the pNFS implementations using GFS2, PVFS, spNFS and GPFS. `LAYOUT4_OSD2_OBJECTS` follows the OSD T10 protocol, like `LAYOUT4_BLOCK_VOLUME` it can speak to compatible back-end storage using the standardized protocols such as iSCSI or Fiber Channel T11. A possible scenario would be that a block volume file system as Lustre[28] may adapt its MDS software to be pNFS compatible and use this layout for the pNFS client to communicate with the storage. Unfortunately, Lustre is based on OSD storage but does not follow the T10 standard and can thus not make use of this layout type. This is an example of a limitation of pNFS, in these cases a separate layout will have to be defined, worked through the NFS working group and registered at IANA. Currently Sun Microsystems who owns Lustre do not have any plans for a Lustre specific layout type.

6.3.2 Mounting with pNFS

Clients mount an exported file system on the MDS. The NFSv4 communication does this from the client using TCP towards port 2049 which is the default NFS port number assigned by IANA (Internet Assigned Numbers Authority) on the MDS. The NFS discovery takes place after which the client hands information about its RPC and NFS software version and possible authentication information. The next step is the ID exchange and creating the actual NFS session.

[6] mount: attatching the NFS made available file system to the local file system for easy access.

After this several calls are made to exchange extra information about values and attributes from which one attribute on the MDS is `fs_layout_type` which indicates that the client may request the server for it's supported file system layout types.

In case the client supports the files layout, it will request associated devices using `GETDEVLIST`. The spNFS PoC setup indicated in this case that the maximum number of devices was set to 16, which is not a really big deal. The devices are identified by IDs which adds an extra layer of abstraction. A device ID maps to a group of storage devices, thus the earlier defined maximum of 16 is not the maximum of individual systems which can be accessed in parallel. For example, the PoC uses two DSs which form one device. The MDS would tell the client in that case that the `DEVLIST` contains 1 device. The client requests this specific device ID along with information on the requested layout type and the maximum value of real devices (individual DSs for example). The PoC had its maximum set to 512. The client then receives the actual information to access the storage. When NFS servers or Object storage devices are used this kind of information could be a list of IP addresses and ports to use. Block storage devices could be addressed using a volume label.

6.3.3 Transferring files

When a file is being transferred, the pNFS data flow looks as shown in table 1 using a 1 megabyte file and 2 DSs using the PoC setup:

Step	Flow			Operation	Packet size (bytes)
1	Client	--->	MDS	OPEN	385
2	Client	<---	MDS	Reply OPEN	566
3	Client	--->	MDS	LAYOUTGET	286
4	Client	<---	MDS	Reply LAYOUTGET	318
5	Client	--->	MDS	(parallel DS I/O)	*
6	Client	--->	MDS	LAYOUTCOMMIT	306
7	Client	<---	MDS	Reply LAYOUTCOMMIT	322
8	Client	--->	MDS	CLOSE	266
9	Client	<---	MDS	Reply CLOSE	326

Table 1: pNFS communication flow.

Some of these sizes may vary and most of the data consists out of NFS calls to exchange attribute data, sequence information and other non pNFS relevant information.
Each step in more detail:

Step 1 & 2: OPEN

These two steps are the same as those used by regular NFS. The filename and information are metadata and thus communicated about with the MDS.

Step 3: LAYOUTGET

The client requests a layout for which it will send along additional information:

Layout availability:
Register to expect a call-back in case of resource exhaustion.
Layout type:
The requested layout type.
I/O mode:
The clients I/O intention (read, read/write).
Offset:
Defines the start data offset of the part the layout should cover.
Length:
Length which the client intends to write. When uploading a file it is set to the used stripe size, at download it is set to the 32768 (bytes) which is the default read and write size for NFS since v3.
Minimum length:
Used to define the minimum range the layout must cover.
Stateid:
The stateid which was returned by the MDS on the OPEN operation.
Maxcount:
This field specifies the maximum layout size which the client can handle. The PoC had this value set to 4096 bytes.

The LAYOUTGET operation required 56 bytes of the 286 bytes of the full Ethernet package.

Step 4: Reply LAYOUTGET

The server returns the requested layout to the client which holds the following information:

Device ID:
The device ID identifies a group of storage devices. If unknown to the client it can use GETDEVICEINFO to request the information from the MDS.
nfl_util:
Tells the client how data is stored, be it either dense or sparse.
First stripe index & pattern offset:
These values are used to define where data should be written and in which pattern.
nfl_fh_list:
An array which contains information on the which filehandle to use in certain situations. The filehandle may be the same as on the MDS or different in which case it may be the same on all DSs or even different on all servers.

Because the PoC uses regular NFS storage as back-end, the layout contains a filehandle for each DS even though the used filenames are the same on each DS. The size of the layout was 128 bytes of which each filehandle takes up 32 bytes. The maximum layout size was set to 4096 bytes, this means that $4096 - 64 = 4032/32 = 126$ filehandles fit in the file layout.

Step 5: parallel DS I/O

After the file layout communication is done the client may transfer data to and from the DSs by itself in parallel. It depends on the pNFS server implementation if writes to DSs are committed to each individual DS or to the MDS. In case of spNFS as used with the PoC the commits went through the DSs. This means that while the actual data is being transferred the MDS is not contacted.

Step 6: LAYOUTCOMMIT

After the I/O has finished, the MDS is contacted to commit the handlings done for the specific layout. Information send during the commit includes the filehandle of the file on the MDS and an offset and length which makes up the amount of data which was changed or added. The commit takes up 48 bytes.

Step 7, 8 & 9: Reply LAYOUTCOMMIT, CLOSE, Reply CLOSE

The MDS will reply on the LAYOUTCOMMIT with verification information on if the size changed and what the data length is. This operations requires 16 bytes. Because the regular NFS operation OPEN was used to begin the whole process, CLOSE is used to end it.

6.3.4 Removing files

Removing a file will be handled through the MDS which uses its control protocol to free up data. The client will then return the layout to the MDS. In case data is removed for which the client holds a layout, it will be recalled by the MDS.

6.3.5 Changing data

When data is changed, possible outstanding layouts may be recalled. A file which has been modified may not result in a layout recall for a client when its cached version is set for read only.

6.4 pNFS availability and status

This section will give an overview of the pNFS development status and availability based on information from the linux-nfs wiki[13] and additional sources such as the pNFS mailing list. Some information was gained by emailing involved parties. Development of pNFS takes place in the NFSv4 working group of the IETF[9]. The latest drafts are available at the NFSv4 Status Page[10] and the IETF will have their 72nd meeting somewhere around July and August of 2008 in Europe.

The current draft of NFSv4.1 is `draft-ietf-nfsv4-minorversion1-23` which is dated 2008-05-12. This draft also includes information about the file based layout. There are 4 drafts available, table 2 gives an overview on which type of layout is involved and who wrote the draft.

Draft	Layout	Involved parties
draft-ietf-nfsv4-minorversion1-23[23]	File based	NetApp Sun Microsystems
draft-ietf-nfsv4-pnfs-block-09[3]	Block/Volume based	EMC
draft-ietf-nfsv4-pnfs-obj-09[1]	Object based	Panasas
draft-ietf-nfsv4-minorversion1-dot-x[24]	-	NetApp Sun Microsystems

Table 2: Available pNFS drafts from the NFSv4 Status Pages[10].

The two drafts on block/volume and object based are only about those layout types. The dot-x draft describes the XDR[7] description for NFSv4 minor version 1 which is not relevant for this research but part of the pNFS drafts.

Actual software development is done by the parties listed in table 3.

Layout	Parties
File based	Sun (Solaris) IBM (GPFS) DESY (dCache) NetApp (spNFS) Red Hat (GFS) CITI/UMICH (Linux)
Object based	Panasas
Block based	EMC

Table 3: Parties involved in pNFS development.

The people from CITI/UMICH are responsible for the most of the Linux client and layout code. They first used PVFS2 as the back-end storage to build the first pNFS prototype but the source code is now outdated and no longer supported by PVFS2. Anyone is free to pick it up and re-implement the latest development code to make pNFS functional again using PVFS2 as the storage backend but nobody is currently pursuing this. Originaly both the NFSv4.1 file layout and a separate PVFS2 layout to talk the native PVFS2 storage protocol to the DSs were used but the work was stopped. The people at CITI are currently collaborating with Red Hat to convert their GFS2[26] client in to a

[7] External data representation, see the glossary in section 13 for more information.

pNFS server. During the time spend on research, no working code was available for testing but it is to be expected soon.

Sun microsystems made a pNFS implementation on OpenSolaris using their ZFS (Zettabyte File System) file system as back-end storage. Their latest available release is dated March 24 2008, based on pNFS draft 19 and built on onnv85[8] which dates to 2008-01-29. They do have a link up on their pNFS status page[27] towards draft 23 which indicates they keep up with the development.

IBM is working on making their GPFS[8] cluster storage back-end compatible with pNFS. This means that clients would no longer need to pay for licenses because they no longer need to use proprietary drivers to use the GPFS storage. The back-end storage may still be proprietary but the problem for the clients is solved by using pNFS.

DESY who is responsible for dCache is working on pNFS support. There has been confusion between Pnfs (Pretty normal file system) and pNFS but dCache will provide native NFSv4.1 support. Version 1.8.0-16 will contain the current snapshot of NFSv4.1 development so people can test it out.

NetApp has created their own client test bed called spNFS[2]. The spNFS software runs in userspace and uses regular NFSv4.1 servers for back-end storage. Their software is fully functional and has been used to test pNFS but currently lacks the abilities which real clustered file systems offer. The only functionality offered is the striping of data using a pNFS file layout towards multiple NFSv4.1 servers. As noted in the slides[2] the software can help drive pNFS adaptation and might be able to take the place of the future easy to use pNFS server for mainline Linux.

To ensure the quality all these implementations and developments and to allow for interoperability testing numerous so called *bakeathons* and *Connectathons* are often held where any vendor who is implementing NFSv4 minor version 1 is welcome to come and test their implementations against those of other vendors. Unfortunately any information regarding these meetings is confidential according to the terms and conditions to which participants have to agree. However, a talk at FRUUG (Front Range UNIX Users Group) in June of 2007 by Sun Microsystems[22] notes that interoperability between vendors has been demonstrated.

[8] OpenSolaris Nevada build 85

6.4.1 Linux pNFS Road Map

The latest Linux pNFS development uses kernel version 2.6.25, changes and additions are made almost daily. According to the Linux pNFS development gantt chart[12], development will last well in to 2009. At the end of august the final specifications should be used in the code but patch submits and testing have been set to continue in 2009.

The current official pNFS patches are for kernel version `2.6.18.3` by CITI. Unfortunately the changes are big and there was no patch history suitable for submission in to the kernel development tree. Benny Halevy of Panasas is using the latest kernel (now at 2.6.25) to re-patch it with small chunks at a time while discussing these on the mailinglist and updating code to the latest draft to end up with a patch history which is suitable for submission for review before it can be added to the official kernel tree.

All of the current development is kept in the linux-nfs git repository tree which on 2008-06-19 looked as follow:

- nfs41
  - pnfs
    - pnfsd-lexp
    - pnfs-block
    - spnfs
    - pnfs-gfs2
    - panlayout

A recent development is **pnfsd-lexp** by Benny Halevy of Panasas which makes it possible to export any local file system over pNFS using the file layout. The server will be both MDS and DS which is useful for development and debugging.

The NFSv4.1 file based layout is currently the layout which is being used for the GFS2, spNFS and GPFS implementations. According to the Linux pNFS development gantt chart[12] they also wish to develop an in-kernel pNFS file system. This pNFS file system is also supposed to be submitted along with the pNFS client implementation to the official kernel git tree. According to CITI the current possibilities are GFS2 and spNFS but no further information was available.

6.5 pNFS behaviour

According to the proposal, spNFS has been used to test the behaviour of pNFS using a PoC. During the project another PoC has been set up for more extensive testing. An overview is depicted in figure 5, the setups consist out of the following hardware:

- Dell 850 running 2.6.18-xen #1 SMP

  Intel(R) Pentium(R) D CPU 3.00GHz

  2G RAM

  4 XEN VMs running Debian Linux with kernel 2.6.25-pnfs #2 SMP Thu Jun 5

  256MB RAM per VM

- 4x IBM X330s Debian Linux with kernel 2.6.25-pnfs #4 SMP Sat Jun 21

  Dual Intel(R) Pentium(R) III (Coppermine) 1.00Ghz

  2G RAM

  1x Client machine with a 1000Mbit NIC

  1x MDS machine with a 100Mbit NIC

  2x DS machine with a 100Mbit NIC

Both the Dell VMs as the IBM machines were set up running the same pNFS kernel from the spNFS git repository. The setup was pretty stable but pNFS would give an occasional problem on mount time which could disable pNFS in favour of NFS. As a few hours of bug hunting and several similar odd error reports on the pNFS mailinglist did not resolve the issue the tests were continued while monitoring kernel messages to avoid bogus results. Several updates where available soon after but were not applied because new bugs were being introduced as well. All the default parameters were used for NFS and the network configuration.

The setup using the IBM machines were used for the scenario testing running on a private network switch.

6.5.1 Transferring a lot of small files

This test has been conducted using PostMark[11] by NetApp. To compare the performance, the MDS has been used for both pNFS as regular NFSv4. PostMark was used with the following settings:

- set numbers 500
- set transactions 1000

The test performed the following file operations:

- 1027 created

  Creation alone: 500 files

  Mixed with transactions: 527 files
- 502 read
- 498 appended
- 1027 deleted

  Deletion alone: 554 files

  Mixed with transactions: 473 files

The operations resulted in the following data transfer:

- 2.90 megabytes read
- 6.20 megabytes written

The results are from an average of 4 runs each where the numbers represent the amount of operations per second. The deviation between the runs was between 1 and 2 seconds:

Operations	NFSv4	spNFS 4096	spNFS 8192	spNFS 16384
Total time	23	161.25	147.5	128
Seconds of trans.	13.25	101.25	97	79.75
trans. per second	74.75	9	10	12
Files created	44.25	6	6.25	7.5
- Creation alone	73	11	13	14.25
- Mixed with trans.	39.25	5	5	6
read	37.25	4	5	6
append	37.25	4	5	6
deleted	44.25	6	6.25	7.5
- deleted alone	207.25	35	39	37.5
- mixed with trans.	35.25	4	4.25	5

Table 4: Postmark pNFS behaviour comparison.

The performance while using pNFS is far worse then native NFSv4. This is not a surprise as the Linux pNFS client code is still in development and

far from being ready for testing. The spNFS daemon is put together only for development and debugging of the pNFS protocol and in no way optimized for any performance testing.

The improvement in performance as the stripe size gets bigger is caused by the fact that out of the two DSs, any file with a size less then the stripe size will have its data put on only the first DS. This means the second DS was rarely used resulting in less operations overall.

According to the draft, future implementations of pNFS will support a minimum threshold which can be used to decide if pNFS or native NFS through the MDS will be used for small transfers. This feature may drastically improve overall performance on smaller files as no layout information is needed. The control protocol between the MDS and the DSs may then be used to load balance the smaller files to avoid hotspots. Exactly how these situations are handled will depend on future pNFS implementations, control protocols and back end storage systems.

6.5.2 Transferring a large file

For this test I used the Linux command `dd` to write a 1 gigabyte file from and towards the pNFS cluster 3 times in a row each.

Transferring towards the cluster resulted in a disappointing average of 7.7MB per second. This was without any peaks or lows, the transfer speed remained almost constant which might point to a problem not related to pNFS. Receiving data maximized the potential of the DSs and resulted in an average of 21.7MB per second.

Because commits were done towards the DSs with the PoC setup, files ranging from 100MB to 2G all resulted in the same data flow from and towards the MDS. No additional load is generated as the size of the layouts are not related to the file size.

Unfortunately I was unable to solve the slow write problem due to time constrains but do not expect it to be a problem which might return in future tests with updated software.

6.5.3 Transferring data through different administrative domains

Transferring traffic between different administrative domains means that these must be able to connect. Firewalls may block connections and in case of pNFS the used storage protocol must be supported over TCP/IP when using the Internet in between.

Mounting the pNFS cluster uses the standard NFS port 2049 as registered at IANA. With the PoC setup, the DSs are mounted directly as well through the same port. Transferring data resulted in no problems and can be seen of as any regular NFS server which is being mounted from another network, except that now a few more connections are made to the DSs which must be accessible from the outside through port 2049.

7 Usage scenarios for pNFS

In any environment, pNFS will require a back-end storage system consisting out of 2 or more DSs. Therefore I doubt pNFS will reach home and small business networks. Basically, any environment with a serious storage platform and requirements for bandwidth beyond that of a single head-filer will classify.

I expect to see most of the current big storage environments to adopt pNFS if they are used by multiple users. Current storage solutions already support NFS through one of the head-filers and with the use of pNFS those systems which require so much bandwidth that parallel access was used through the use of proprietary software will now be able to use pNFS. This way license problems for clients will be solved and there will be less vendor lock-in. No changes in the storage platform are required as any of the currently used interconnections and protocols will be supported.

When just data needs to be transferred at high speeds, other solutions such as GridFTP[30] may be used. In comparison pNFS will offer far more as it comes with all the features of NFSv4 such as a unified namespace and has layout drivers to directly communicate with back-end storage devices. A feasible scenario would be for example that pNFS is used by users of the systems (be it real or machines in a HPC cluster) and GridFTP by administrators and background backup and copy processes. Yet I see no reason why pNFS would not be used for those types of operations as well. It may even be used together where A is a pNFS client who runs a GridFTP server when the other side B may or can not directly mount the pNFS MDS of A.

8 10Gbit and 600 PB of data

The challenge is to transfer large amounts of data while utilizing the 10Gbit connection to its fullest to obtain the most desired condition. The fastest NICs on the market today use PCI Express which has more then enough bandwidth to fully utilize the 10Gbit line speed unlike the earlier PCI-X cards which were limited to 8.512Gbit/second. Advances in bus architectures eliminates this hardware as a potential bottleneck. However, earlier research [15] has shown that transfer speeds are being limited by software rather then hardware. Because pNFS sets up transfers in parallel, benchmark tests in earlier research (see section 4) such as at the Bandwidth Challenge at Supercomputing 2007 in Reno, Nevada[4] have shown that pNFS is capable of fully utilizing a 10Gbit connection. As described in section 7 pNFS could also be one of the links in the chain to achieve the goal of transferring the data.

9 Considerations

There were no available resources to test the PoC using a high speed, high latency Internet connection. It is important that future work will be done using such resources. The latencies and implementations of high speed high latency networks may have an impact/influence on the performance and utilization of the available resources. Because of these variables, the pNFS protocol might not scale as well as expected.

10 Conclusion

The main research question was the following:

> Is pNFS capable of transferring the large amounts of data required in the NorStore context?

As described in the section 9 it was not possible to test using a network infrastructure similar to that of the NorStore project. The pNFS software is not even ready for real tests as the specifications are still in a draft form and available implementations are far from ready. However, by looking at the facts gathered in this report and information from previous related work in section 4, there is little reason why it would not work sufficiently on high bandwidth connections up scaling towards tens of gigabits per second if the available hardware permits it. Together with the findings done in the other sections I conclude that the original hypothesis still stands:

> pNFS will be capable of transferring large amounts of data but is currently still missing good integration with underlying file systems and under heavy development. Therefore it is currently not ready for production use.

11 Future Work

Further research should take place when pNFS has crystallized in to the Linux kernel and implementations are ready for testing. To make sure pNFS is useful in high latency high bandwidth environments, field tests are required. Attention should be paid to both pNFS and the network. High speed high latency networks can behave very differently then expected and default settings will probably be far from optimal. A nice addition to such a test would be two different back-end storage systems to test the interoperability.

12 References

References

[1] Benny Halevy, Brent Welch & Jim Zelenka (Panasas, Inc.) . Object-base pNFS Operations (draft 09), June 19, 2008. http://tools.ietf.org/pdf/draft-ietf-nfsv4-pnfs-obj-09.pdf.

[2] Dan Muntz, Mike Sager & Ricardo Labiaga. spNFS, A Simple pNFS Server. http://www.connectathon.org/talks08/dmuntz-spnfs-cthon08.pdf.

[3] David L. Black, Stephen Fridella & jason Glasgow (EMC). pNFS Block/Volume Layout (draft 09), June 11, 2008. http://tools.ietf.org/pdf/draft-ietf-nfsv4-pnfs-block-09.pdf.

[4] Patricia Kovatch & Phil Andres (National Institute for Computational Sciences) & John White (Revision3) Dean Hildebrand, Marc Eshel & Roger Haskin (IBM). Deploying pNFS across the WAN: First Steps in HPC Grid Computing. 2008. http://www.linuxclustersinstitute.org/conferences/archive/2008/PDF/Hildebrand_98265.pdf.

[5] Peter Honeyman Dean Hildebrand, Wm. A. (Andy) Adamson (Center for Information, and University of Michigan) Technology Integration. pNFS and Linux: Working Towards a Heterogeneous Future. 2007. http://www.citi.umich.edu/techreports/reports/citi-tr-07-1.pdf.

[6] EMC. EMC Celerra Multi-Path File System (MPFS). http://www.emc.com/collateral/software/data-sheet/h2006-celerra-mpfs-mpfsi.pdf.

[7] Dean Hildebrand, Lee Ward, and Peter Honeyman. Large files, small writes, and pnfs. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, pages 116–124, New York, NY, USA, 2006. ACM.

[8] IBM. IBM General Parallel File System. http://www-03.ibm.com/systems/clusters/software/gpfs/index.html.

[9] IETF. Network File System Version 4 (nfsv4) Charter. http://www.ietf.org/html.charters/nfsv4-charter.html.

[10] IETF. NFSv4 Status Pages. http://tools.ietf.org/wg/nfsv4/.

[11] J. Katcher. Postmark: A new file system benchmark. *Technical Report TR3022, Network Appliance*, Oct. 1997.

[12] linux-nfs.org. Linux pNFS Development Gantt Chart. http://spreadsheets.google.com/pub?key=pGVvgce8dC-WWbowI9TSmEg.

[13] Linux-nfs.org. NFSv4. http://wiki.linux-nfs.org/wiki/index.php/Main_Page.

[14] NetApp. NetApp. http://www.netapp.com/.

[15] Niels Visser & Daniël Hilster. G5 Performance Report. https://www.os3.nl/_media/2003-2004/asp/reports/dh_nv-g5-performance-report.pdf.

[16] Panasas. DirectFLOW. http://www.panasas.com/directflow.html.

[17] Panasas. Panasas PanFS Parallel File System. http://www.panasas.com/panfs.html.

[18] Panasas. Parallel Storage Clusters. http://www.panasas.com/.

[19] Panasas. pNFS. http://www.pnfs.com/.

[20] W. Curtis Preston. *Using SANs and NAS*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[21] PVFS. Parallel Virtual File System, Version 2. http://www.pvfs.org/.

[22] Sam Falkner (Sun Microsystems, Inc.). pNFS, June, 2007. http://www.fruug.org/Archive/2007-10/fruug-pnfs.pdf.

[23] Spencer Shepler(Sun Microsystems), Mike Eisler & David Noveck (NetApp). NFS Version 4 minor Version 1 (draft 23), May 12, 2008. http://tools.ietf.org/pdf/draft-ietf-nfsv4-minorversion1-23.pdf.

[24] Spencer Shepler(Sun Microsystems), Mike Eisler & David Noveck (NetApp). NFSv4 Minor Version 1 XDR Description (draft 06), May 12, 2008. http://tools.ietf.org/pdf/draft-ietf-nfsv4-minorversion1-dot-x-06.pdf.

[25] Hal Stern. *Managing NFS and NIS*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[26] Steven Whitehouse (Red Hat, Inc.). The GFS2 Filesystem, 2007. http://ols.108.redhat.com/2007/Reprints/whitehouse-Reprint.pdf.

[27] Sun Microssytems, Inc. OpenSolaris Project: NFS version 4.1 pNFS. http://opensolaris.org/os/project/nfsv41/.

[28] Sun Microsystems. Lustre. http://wiki.lustre.org/index.php?title=Main_Page.

[29] TeraGrid. TeraGrid. http://www.teragrid.com/.

[30] The globus alliance. Grid Ecosystem - GridFTP. http://www.globus.org/grid_software/data/gridftp.php.

[31] UNINETT SIGMA. NorStore - Norwegian Storage Infrastructure. http://www.norstore.no/.

[32] Wikipedia. Natural science. http://en.wikipedia.org/wiki/Natural_sciences.

[33] Wikipedia. Open standard. http://en.wikipedia.org/wiki/Open_standards.

13 Glossary & Acronyms

Glossary

export Using NFS, the transfer of the filesystem over the network to other machines.

Ext3 Ext3 (third extended filesystem) is the default filesystem used by many Linux distributions.

git A distributed revision control / software code management project created by Linus Torvalds, initially for the Linux kernel development. (Source: Wikipedia)

iSCSI A protocol for sending SCSI commands using TCP/IP.

metadata Data which describes other data. Examples include filename, size, owner and location.

mirrored Storing the data twice on two separate devices for higher availability and possible improved read performance.

SCSI Small Computer System Interface, a communication standard for a range of storage devices.

striped Distributing data across several storage devices for improved performance.

T10 T10 is a Technical Committee, they are responsible for many SCSI command set standards (e.g., SPC-4, SBC- 3, SSC-3, MMC-6, SMC-3, OSD-2, RBC, etc.). These standards are used by almost all modern I/O interfaces, including SCSI, SAS, Fibre Channel, SSA, IEEE 1394, USB, and ATAPI (ATA). More information can be found on http://www.t10.org/.

XDR *External data representation* XDR is a data representation layer to unify differences in data representation which may exist in a heterogeneous network.

List of Acronyms

CIFS ..... Common Internet File System

CITI ..... Center for Information Technology Integration

DS ..... Data Server

FRUUG ..... Front Range UNIX Users Group

HPC ..... Higher Performance Computing

IANA ..... Internet Assigned Numbers Authority

IETF ..... Internet Engineering Task Force

MDS ..... MetaData Server

MPFS ..... Multi-Path/Multi-Protocol File System

NAS ..... Network Attached Storage

NFS..... Network File System

NREN ..... National Research and Education Network

OSD ..... Object Storage Device

Pnfs ..... Pretty normal file system

pNFS ..... Parallel Network File System

PoC ..... Proof of Concept

RAID ..... Redundant Arrays of Inexpensive Disks

RPC ..... Remote Procedure Call

SAN ..... Storage Area Network

spNFS ..... Simple pNFS

T11 ..... T11 is a Technical Committee who is responsible for Fibre Channel interfaces. More information can be found on http://www.t11.org/.

UMICH..... University of Michigan

WAN ..... Wide Area Network

ZFS..... Zettabyte File System

A Appendix

A.1 Figures

A.1.1 PoC

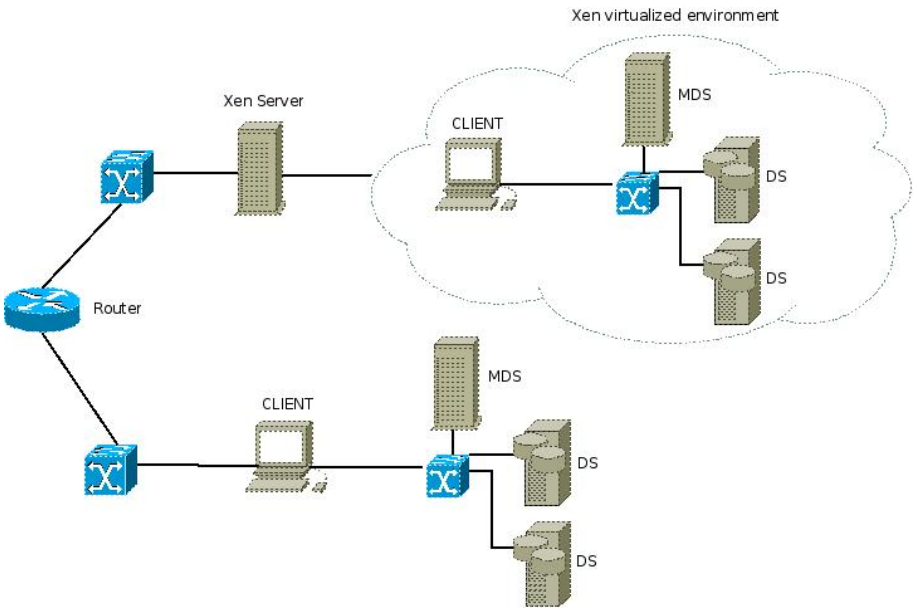

Figure 5: An overview of the PoC setup.