Statement of Work

NFSv4 Extensions for Performance and Interoperability

Center for Information Technology Integration
March 1, 2008 – February 28, 2009

Summary

This is a statement of work to be performed by CITI in a project sponsored by EMC. The major goal of this project is to complete the implementation of pNFS block layout capability in the Linux NFSv4.1 client. The driver will implement storage class-independent pNFS extensions and layout management API following specifications from the IETF specifications of pNFS protocol operations and block based pNFS.

To advance pNFS and NFSv4.1 standardization, CITI will implement sessions, reboot recovery, and other NFSv4.1 requirements in the Linux client and server. To aid implementation and testing by pNFS block layout developers, CITI will enrich the Python-based PyNFS client and server to support newly specified pNFS features.

The ultimate metric for success is pNFS client support for block layout in the mainline Linux kernel. Acceptance into the mainline kernel is a process involving peer review by and consensus among Linux developers and maintainers, so the principal deliverables are Linux software submitted for peer review and active participation in the consensus process.

Background

pNFS is an extension to NFSv4[1] that allows clients to overcome NFS scalability and performance barriers. Like NFS, pNFS is a client/server protocol implemented with secure and reliable remote procedure calls. A pNFS server manages storage metadata and responds to client requests for storage layout information. pNFS departs from conventional NFS by allowing clients to access storage directly and in parallel. By separating data and metadata access, pNFS eliminates the server bottlenecks inherent to NAS access methods.

While the pNFS protocol allows parallel access to files stored in any kind of back end, the IETF working group focuses on access to NFS servers, object storage, and block storage. The generic aspects of the pNFS protocol are described in a working document[2] that is moving toward standardization as the NFSv4.1 specification. That document also describes the pNFS file layout protocol. Mechanisms for access to object[3] and block[4] storage are described in separate documents, which are also moving toward standardization.

By combining parallel I/O with the ubiquitous standard for Internet filing, pNFS promises unprecedented benefits:

- State of the art performance
- Massive scalability
- Interoperability across standards-compliant application platforms
- Insulates storage architects from the risks of deploying best-of-breed technologies

pNFS makes the design of innovative and specialized storage systems future-proof.

---

[1] "Network File System (NFS) version 4 Protocol," S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, RFC 3530 (April 2003). http://www.ietf.org/rfc/rfc3530.txt

[2] "NFS Version 4 Minor Version 1," S. Shepler, M. Eisler, and D. Noveck (eds.), http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-minorversion1-19.txt (January 29, 2008)

[3] "Object-based pNFS Operations," B. Halevy, B. Welch, and J. Zelenka, http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-pnfs-obj-04 (September 5, 2007)

[4] "pNFS Block/Volume Layout," D.L. Black, S. Fridella, and J. Glasgow, http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-pnfs-block-05.txt (November 18, 2007)

Heard on the street

In my view, [the pNFS] standards effort is admirable and should be supported across the storage industry. Potential benefits for users include improved sustained performance, accelerated time to results (solution) and parallel storage capability with standard highly reliable performance. It offers more choice of parallel I/O capabilities from multiple storage vendors, freedom to access parallel storage from any client, as well as mix and match best of breed of vendor offerings. It also contains lower risk for the user community, since client constructs are tested and optimised by the operating system of vendors whilst the customer is free from vendor lock-in concerns. In short, it extends the benefits of the investment in storage systems.

Christopher Lazou, *Parallel Storage: A Remedy for HPC Data Management*, HPC Wire, January 18, 2008

Initially, a pNFS standard solution will be more applicable to HPC markets; however, pNFS will certainly prove beneficial to commercial markets with applications that demand increasing levels of computing and data resources.

Sal Capizzi, *Parallel NFS for Increased Storage Performance and Scalability*, Ideas Insights, June 7, 2007

pNFS is going to commoditize parallel data access. In 5 years we won't know how we got along without it.

Robin Harris, *StorageMojo*, October 15, 2007

pNFS development

CITI is one of a handful of primary contributors to the Linux-based, open source implementation of NFSv4.1 and pNFS. A partial list of organizations and engineers contributing to NFSv4.1 and pNFS is shown in Appendix 1.

CITI's NFS effort has had numerous federal and industry sponsors. Long-term stability is provided through CITI's participation in the SciDAC Petascale Data Storage Institute,[5] a five-year program funded by SciDAC.[6] The majority of CITI's NFSv4.1 funding comes from one-year agreements with industry partners. At this writing, CITI's industry partners include EMC, IBM, and Network Appliance. Google is likely to sponsor NFSv4 work at CITI in 2008. Discussion is also under way with Red Hat, LSI Logic, and SGI. Recent sponsors of NFS research and development at CITI include the National Science Foundation, Los Alamos, Sandia, and Lawrence Livermore National Labs, Sun, Polyserve, and SGI.

CITI and EMC

Since 1999, the University of Michigan's Center for Information Technology Integration has been the lead developer for the open source, Linux-based reference implementation of NFSv4, the Internet standard for distributed filing. For much of this period, CITI and EMC engineers have collaborated on the specification, design, and rigorous interoperability testing of NFSv4.

In 2006, EMC began sponsoring software development at CITI, focused on advancing the implementation of NFSv4 and later the pNFS block layout client in NFSv4.1. Within the scope of this partnership, engineers at CITI and EMC have succeeded in implementing the elements of the pNFS protocol that enable block layout and succeeded in tests if CITI's Linux client implementation against a EMC's Celerra server. In addition, CITI extended PyNFS, its Python-based implementation of the NFSv4 client and server, to meet the needs of project developers. CITI has also addressed issues critical to completing the NFSv4.1 specification and implementation.

Part of the challenge in developing pNFS block layout is tracking changes in the IETF specifications and the Linux kernel. In the past 13 months, the specification of pNFS evolved in a series of 14 drafts, the block layout specification saw six drafts, and the Linux kernel has moved from 2.6.20 to 2.6.24.

Development plans are necessarily flexible, not only to accommodate the rapid pace of change in protocol specification and in the base Linux kernel, but also in recognition of the consensus process that governs acceptance of patches into the mainline Linux kernel, which sometimes requires realigning the architecture of working code in the direction of Linux kernel evolution.

---

[5] PDSI is a five-year collaboration (in its second year) between researchers at Carnegie Mellon University Parallel Data Lab, National Energy Research Scientific Computing Center, Pacific Northwest National Laboratory, Oak Ridge National Laboratory, Sandia National Laboratory, Los Alamos National Laboratory, University of Michigan (CITI), and the Storage Systems Research Center at University of California, Santa Cruz. http://www.pdsi-scidac.org/

[6] Scientific Discovery through Advanced Computing, Office of Science, Department of Energy. http://www.scidac.gov/.

Status and goals

At this point, the block layout client provides basic pNFS functionality:

- Read and write through block layout driver with error recovery over NFS.
- Integration with the Linux kernel dm (device mapper) layer.
- Basic client layout segment and extent cache.

In the past year, CITI also tackled some necessary software engineering issues:

- In moving from the 2.6.18.3 Linux kernel to tracking the latest Linus git tree (2.6.24 at this writing), we completely rewrote the generic pNFS client.
- The block layout code is now organized as a branch off the pNFS git tree.
- CITI helped draft and review the IETF draft specifications for NFSv4.1 and pNFS block layout.
- CITI extended the PyNFS suite to continue to support block layout development and testing.
- CITI completed the NFSv4.1 directory delegations implementation.

For 2008, our goal is to complete the specification and implementation of the block layout client:

- Continue to refine block layout specification so that it reaches last call.
- Deliver a functional and complete implementation of the NFSv4.1 client and the block layout client.
- Continue to extend the PyNFS suite for development and testing

A complete and functional NFSv4.1 and block layout implementation is necessary for approval of code patches by Linux maintainers and inclusion into the mainline kernel. CITI's goal is to see that accomplished in 2009, and to see NFSv4.1 and pNFS present in Linux distributions in 2010.

Tasks

NFSv4.1 introduces (and mandates) *sessions*, a communication model that provides a client context for every RPC request. pNFS uses sessions to ensure ordering in a request stream and to prevent race conditions when callbacks are broken.

The following sessions components in the generic Linux pNFS client are incomplete and untested:

Task	Description	Dependencies
S1	Session recovery.	
S2	Callback channel implementation.	Complete
S3	NFSv4.1 back channel security using machine credentials.	S2
S4	NFSv4.1 back channel security using secret state verifiers.	S1, S2, S3

The generic pNFS client also requires resolution of these issues:

Task	Description	Dependencies
C1	Implementation of LAYOUTGET, LAYOUTRETURN, and CB_LAYOUTRECALL await completion of the sessions callback channel.	S2
C2	Support for CB_RECALL_ANY, RECLAIM_COMPLETE, and CB_RECALLABLE_OBJ_AVAIL operations.	S2
C3	Integration of block layout requirements into the generic client.	S2, B2
C4	Implement new NFSv4.1 draft 19–21 pNFS features and behavior such as pNFS device notification, CB_NOTIFY_DEVICEID, and layout stateid processing.	S2
C5	Implementation of reboot recovery.	S2, C2

Completing the block layout module requires the following:

Task	Description	Dependencies
B1	Revise the current implementation, based on draft 3, to comply with the latest specification, draft 6.	Ongoing
B2	The block layout implementation must be extended to support large server block sizes.	
B3	CITI and EMC have been invited to present a position paper "Parallel NFS Block Layout Module for Linux" at the Linux Storage and Filesystem Workshop to be held on February 25-26, 2008 in San Jose. (See Appendix II.) This is a small, tightly focused, invitation-only workshop that brings together Linux developers, maintainers, and researchers interested in advances in the Linux file and storage subsystems that can find their way into the mainline kernel and into Linux distributions in the 1–2 year timeframe. CITI will revisit the block layout client implementation based on that architectural review.	See Appendix III
B4	Support for complex volume topologies uses the Linux device mapper (dm). This code needs to be reviewed to meet performance and quality requirements.	B3

Task	Description	Dependencies
B5	The layout cache implementation assumes only one dm device. To support copy-on-write, we need to extend the implementation to support at least two devices.	B3
B6	The dm device must be extended to support the asynchronous CB_NOTIFY_DEVICEID callback operation.	S2, C4, B3
B7	The block layout client must implement a timed lease I/O fencing mechanism to insulate against network partition.	

The Python-based NFSv4.1 client and server developed in earlier phases of work prove extremely valuable for testing completeness and correctness of the Linux pNFS block layout client and the Celerra-based block layout metadata server. Tracking the IETF work requires:

Task	Description
P1	Update PyNFS client and server from NFSv4.1 draft 13 and block layout draft 3 to support new protocol features in the latest drafts, currently drafts 21 and 6, respectively.
P2	Enhance the block server implementation to pass full Connectathon tests. (See "Linux kernel consumer requirements" below.)

Milestones

A schedule for milestones follows naturally from the need to complete tasks for testing at the Connectathon in May 2008, at the fall 2008 Bakeathon, and at the end of this project in February 2009.

Connectathon milestones

The following tasks will be complete by the May 2008 Connectathon. CITI will report on project status.

S1 Session recovery.
S2 Callback channel implementation.
B1 Block layout draft 6.
B2 Server block sizes greater then 4 KB.
B3 Revisit block layout client implementation based on architectural review.

Bakeathon milestones

The following tasks will be complete by the fall 2008 Bakeathon. CITI will report on project status.

S3 Back channel security using machine credentials
C1 LAYOUTGET, LAYOUTRETURN, and CB_LAYOUTRECALL.
C2 CB_RECALL_ANY, RECLAIM_COMPLETE, and CB_RECALLABLE_OBJ_AVAIL.
P1 PyNFS block client and server support latest drafts
P2 PyNFS block server passes full Connectathon tests.

The following tasks will be under way by the Fall Bakeathon. CITI will report on their progress at that time.

C3 Integration of block layout requirements into the generic client.
C4 Draft 19–21 pNFS features and behavior.
B4 Complex volume topologies.
B5 Copy-on-write.

Project end milestones

All tasks will be complete. CITI will report on the remaining tasks and overall project accomplishments.

S4 NFSv4.1 back channel security using secret state verifiers
C3 Integration of block layout requirements into the generic client.
C4 Draft 19–21 pNFS features and behavior.
C5 Reboot recovery.
B4 Complex volume topologies.
B5 Copy-on-write.
B6 CB_NOTIFY_DEVICEID
B7 Timed lease I/O fencing mechanism.

Risks

Circumstances beyond CITI's control may affect progress on tasks, milestones, and deliverables.

Server rebasing

EMC must keep pace with CITI's client implementation by rebasing their pNFS server to the latest IETF NFSv4 minor version draft and completing the implementation of NFSv4.1 operations.

Linux kernel review

Linux kernel modification is accomplished through a complex socio-technical process that involves Linux developers worldwide, specialized kernel subsystem maintainers, and ultimately Linus Torvalds. A change is proposed by submitting patches and verbal justification to the Linux kernel mailing list or other focused Linux mailing lists for peer review. Consensus, when achieved, arises through discussion on the lists and at specialized workshops. Often the process is an iterative one, requiring architectural or other changes to meet the standards of the relevant kernel maintainers. Once a patch is deemed acceptable by the kernel maintainers, it is forwarded to Linus for final review.

It is vital to have the attention and support of the designated kernel maintainers. CITI is fortunate to have close ties with Linux NFS client and server maintainers, Trond Myklebust and Bruce Fields. Trond moved to CITI in 2003, while still a doctoral student. He later accepted employment from NetApp, but retains his CITI office and is a peer to CITI developers and researchers. Bruce, who holds a faculty appointment at the University of Michigan, has been a member of staff at CITI since 2001.

CITI's close relationships with Trond and Bruce are expected to continue throughout this project. However, it must be emphasized that in their roles as Linux maintainers, Trond and Bruce exercise independent judgement in assessing the means to advance the Linux kernel. In particular, CITI has no special privilege in determining the outcome of the their reviews. So, while collegial relations lead naturally to shared goals and understanding, at the end of the day, the decision on whether developments are incorporated into the mainline kernel is not under CITI's control.

Linux kernel consumer requirements

An open source solution for all facets of a system is an important milestone when patches for a multifaceted system are submitted for inclusion into the mainline kernel. In particular, a client/server system typically includes an open source client and an open source server. CITI ran up against this restriction in its effort to enhance the NFS server for cluster file systems: CITI's kernel patches were not accepted until the kernel also included a consumer of the enhancements, i.e., a cluster file system of its own. Once GFS2 and OCFS2 were accepted into the kernel, CITI's work met the consumer requirement and patches began to be accepted.

The need for a complete solution affects this project: unless the Linux kernel includes a block pNFS server, kernel maintainers and developers will resist including CITI's block layout driver. CITI is responding by advocating its Python-based block layout server as an implementation acceptable to Linux developers and maintainers: at the Linux Storage and Filesystem Workshop, CITI and EMC explored with Linux developers and maintainers whether a PyNFS block server that passes Connectathon tests meets the requirement for a "complete" block server solution, i.e., client and server. This would promote acceptance of pNFS block layout client patches into the mainline Linux kernel.

The issue remains unresolved, but should this avenue not succeed, CITI will propose that because the block layout driver is an open source reference implementation of an IETF standard, it should be considered exceptional and reviewed for inclusion. In addition, CITI is engaged in discussions with other potential partners to staff and fund a native Linux-based, open source implementation of the block layout pNFS server. If the consumer requirement delays mainline inclusion, the software can be packaged and distributed as a loadable kernel module in the interim. In this case, CITI and EMC can work with the Linux distributions (Red Hat, SUSE, etc.) for inclusion in the kernels they build and distribute.

Appendix I

This table is a partial list of organizations and engineers who have contributed to NFSv4.1 and pNFS.

Organization	People	Role
CITI	Andy Adamson	Generic pNFS client File and block layout client Generic pNFS server Long haul WAN performance for NFSv4.1 and pNFS file layout
	Bruce Fields	Code review
	Peter Honeyman	Advisory
	Fred Isaman	pNFS block layout client PyNFS NFSv4.1 test suite PyNFS pNFS block client and server
	Jim Rees	Long-haul WAN performance for NFSv4.1 and pNFS file layout
	David Richter	Directory delegation
DESY	Tigran Mkrtchyan	pNFS file layout server for dCache in Java pNFS wireshark module
EMC	Richard Chandler	pNFS block layout architecture pNFS Celerra implementation
	Daniele Gardere	NFSv4.1
	Jean-Loiuis Rochette	pNFS and delegation implementation
	Jason Glasgow	Code review
	Mario Wurzl	Advisory
IBM	Marc Eshel Dean Hildebrand	pNFS file layout client Generic pNFS client and server
Network Appliance	Ricardo Labiaga Mike Sager	NFSv4.1 sessions client and server
	Dan Muntz	pNFS file layout server on linux
	Trond Myklebust	Code review
	Tom Talpey	Advisory
Panasas	Benny Halevy	pNFS generic client and server pNFS OSD layout client and server

Appendix II
Parallel NFS Block Layout Module for Linux

William A. Adamson, University of Michigan
Frederic Isaman, University of Michigan
Jason Glasgow, EMC

Introduction

This position statement presents CITI's Linux prototype of NFSv4.1 pNFS client block layout module and reviews our implementation approach. CITI's prototype implements the IETF draft specification draft-ietf-nfsv4-pnfs-block and is one of three layout modules being developed along with the Linux pNFS generic client, which implements the draft-ietf-nfsv4-minorversion1 specification. The block layout module provides for an I/O data path over iSCSI directly to client SCSI devices identified by the pNFS block server.

CITI has also developed a Python-based NFSv4.1 test environment – an LVM-based pNFS block layout server that supports SCSI disks emulated in RAM and the iSCSI protocol – to test direct block I/O and complex volume topologies along with the pNFS and other NFSv4.1 operations.

We refer to draft-ietf-nfsv4-minorversion1-17.txt Section 12 for a detailed description of NFSv4.1 Parallel NFS.

http://www1.ietf.org/internet-drafts/draft-ietf-nfsv4-minorversion1-17.txt

Here are snippets of the introduction to set the stage.

```
12.1.  Introduction

   pNFS is a set of optional features within NFSv4.1; the pNFS feature
   set allows direct client access to the storage devices containing
   file data.  When file data for a single NFSv4 server is stored on
   multiple and/or higher throughput storage devices (by comparison to
   the server's throughput capability), the result can be significantly
   better file access performance.

   pNFS takes the form of OPTIONAL operations that manage protocol
   objects called 'layouts' which contain data location information.

   The NFSv4.1 pNFS feature has been structured to allow for a variety
   of storage protocols to be defined and used.

   The NFSv4.1 protocol directly defines one storage protocol, the NFSv4.1
   storage type, and its use.

   Examples of other storage protocols that could be used with NFSv4.1's
   pNFS are:

   o  Block/volume protocols such as iSCSI ([35]), and FCP ([36]).  The
      block/volume protocol support can be independent of the addressing
      structure of the block/volume protocol used, allowing more than
      one protocol to access the same file data and enabling
      extensibility to other block/volume protocols.

   o  Object protocols such as OSD over iSCSI or Fibre Channel [37].

   o  Other storage protocols, including PVFS and other file systems
      that are in use in HPC environments.
```

The pNFS block layout module is specified in draft-ietf-nfsv4-pnfs-block-05.txt

http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-pnfs-block-05.txt

The pNFS operations carry an opaque payload, which conforms to a storage protocol layout type, between the pNFS client and the pNFS server. The Linux pNFS client and server prototype implementations reflect this design.

The Linux pNFS generic client performs tasks common to all layout types. Per layout type payloads are opaque to this generic code. The generic code passes an opaque payload to the appropriate registered layout module via a layout module API, which runs code specific to the layout type.

The Linux NFSv4.1 pNFS server exports pNFS capable file systems. The pNFS server design is similar to the pNFS client. The pNFS server performs tasks common to all layout types. Opaque payloads are passed to and from the exported pNFS capable file system via a new set of operations in struct export_operations.

A team of engineers from CITI, Network Appliance, Panasas, and IBM are implementing the Linux pNFS client and server prototypes. The Linux block layout module is based on the EMC MPFS file system client, which shares many design points with pNFS.

The Linux pNFS client and server prototypes have been tested at four interoperability events where the Linux pNFS client demonstrated support for simultaneous multiple layout modules, and the Linux pNFS server was used to export object (Panasas) and NFSv4.1 (IBM and Network Appliance) based file systems.

Protocol Requirements
Here is a list of the requirements placed on the pNFS block layout client by the protocol. We are considering only the iSCSI storage protocol transport at this time.

1) Identify storage volumes by content
2) Support arbitrarily complex volume topologies per file system id
3) Break down and reset logical disk/volume topology
4) I/O requires mapping file offset ⇒ extent ⇒ volume logical offset ⇒ physical disk + offset
5) Block I/O to SCSI disk
6) 0 ≤ write size ≤ server file system block size
7) Fail over to NFSv4.1 server
8) Specify the maximum I/O time of the I/O stack
9) Copy-on-write support

Here are the kernel interfaces that our block layout module prototype uses to implement the requirements.

Prototype Kernel Interfaces
The first job is to identify which of the SCSI disks the client can see belong to the pNFS server.

1) Identify storage volumes by content.

Our prototype does the following.

> EXPORT drivers/scsi/hosts.c: shost_class symbol lists all SCSI devices. We walk the list, and identify all SCSI disks that can be open_by_dev() and bd_claim().
> We retrieve a list of device IDs from the pNFS block server and corresponding content at an offset. For each device ID, the prototype reads each SCSI disk comparing content at the offset. When we find a match, the pNFS server device ID is associated with the disk. We bd_release() and close all unassociated disks.

INTERFACE:
a) shost_class
b) open_by_dev
c) bd_claim
d) bios read.

The next three requirements inspired our use of the LVM interface.

2) Support arbitrarily complex volume topologies per file system id
3) Break down and reset logical disk/volume topology
4) I/O requires mapping file offset ⇒ extent ⇒ volume logical offset ⇒ physical disk + offset

These requirements seemed to be a good match for the user land LVM2 software. When our prototype gets a volume topology with disks identified via content, we call the LVM2 services in the kernel.

> Create a dm device (a la user land LVM ioctl interface) to represent volume topology.
> For I/O, the dm device handles the logical offset to physical disk + offset mapping for us.

INTERFACE: create a dm device

This does most of what we want. We could write our own dm target. We use the ioctl interface in the kernel. Splitting dm functionality to service both the existing ioctl interface and the desired kernel interface remains an issue.

The dm device gives us a device for performing I/O. Next we are faced with the need to choose the best kernel interface for the following requirements.

5) Block I/O to SCSI disk
6) 0 ≤ write size ≤ server file system block size
7) Fail over to NFSv4.1 server

INTERFACE: We have implemented two paths:

Use the block_read/write_full_page() interface.

- We need our own callback routine for cleanup and error recovery.
- Calls bios, and does it right!
- Large server file system block size is an issue because we might have to deal with fs block sizes larger than a page.

Code our own bios routines.

- Write case is complex
- It is easy to code our own callback routine

Issues

We have yet to address these remaining requirements:

8) Specify the maximum I/O time of the I/O stack
9) Copy-on-write support

Knowing when to give up on I/O to a disk is an issue. We have not yet addressed the copy-on-write requirement.

We hope to discuss this and other pNFS position statements at the workshop, which will help us move pNFS into the Linux kernel over the next 1–2 years.

Appendix III
Report from LFS presentation

From: William A. (Andy) Adamson
Subject: LFS pNFS block client presentation report
Date: February 26, 2008 5:05:47 PM GMT-05:00

I presented after Benny Halevy's pNFS object presentation which was good 'cause we share a lot of functionality. BTW the object pNFS driver does not use DM - they wrote their own.....

[DM is the Linux "device manager" which is used to coalesce volumes for the Linux Volume Manager. It is not well-loved, in part because it is accessed through ioctls, and in part because it has only one consumer, LVM.]

There was a bit of resistance to the idea of block pNFS. In particular, the fail over to NFS was noted to be quite complicated WRT the two buffering and paging schemes (NFS and Block) and error handling. Christoph Hellwig was quite vocal calling the idea 'crap' ! Typical Christoph. He also said the same about NFSv4 in all it's forms so it's not surprising that he really disliked pNFS.

[Actually, the fail-over criticism has some substance: it points out the lack of an error path from block client to block MDS. And it can be a very real problem, with the prevalence of SAN multi-pathing.]

DM in general was questioned - how well the interfaces are constructed, how well it performs, and the error reporting to upper layers. All of these are exasperated by our use. We need new interfaces, the complex topologies we anticipate, and the distributed nature of our error reporting which needs to get back to the MDS. On the other hand, a new file system, BTRFS, mentioned that it considered using DM, but wrote it's own similar functionality instead which got some negative comments. So, I think writing a proper kernel interface to DM (get rid of using the ioctl) and using/measuring a recursive DM architecture for stripes within stripes is probably a good way to go.

The object pNFS talk mentioned the desire to have an iSCSI daemon to enable waiting to mount disks until they are actually used on the client instead of enumerating and identifying disks as we do now. Interesting.

There was no discussion of my reporting on trying to use nobh_writepages and friends - the NFS fall back problem (page->private being used by NFS and nobh_writepages on failure) was not popular. We could add a bios callback and an get_private function to nobh_writepages and see the response.

There was no suggestion on how to handle the server block size vs page size. I guess we are on our own. There was a good deal of discussion on how to handle disk errors - disks going off line and then coming back on line - kind of addressing our problem of when to give up on a disk.

All in all, we can continue with our base design, and sending email to fsdevel now that we have presented.