Statement of Work

NFSv4 Extensions for Performance and Interoperability

Center for Information Technology Integration
March 1, 2009 – February 28, 2010

Summary

This is a statement of work to be performed by CITI in a project sponsored by EMC. The major goals of this project are to complete the implementation of pNFS block layout capability in the Linux NFSv4.1 client and to shepherd its integration into the mainline Linux kernel.

To complete the implementation of pNFS block layout capability in the Linux NFSv4.1 client, CITI will address outstanding protocol compliance issues and known performance bottlenecks, measure and improve block client parallel performance against CITI's Celerra server, participate in Connectathon and Bakeathon interoperability testing events. In addition, to aid implementation and testing by pNFS block layout client and server developers, CITI will continue to enrich the Python-based PyNFS client and server to support complex pNFS features.

To advance pNFS block layout integration into the mainline Linux kernel, CITI will continue to work closely with the Linux NFS client maintainer and principal developers and to participate actively 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. 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.

A pNFS server manages storage metadata and responds to client requests for storage layout information, which the client uses as a map for locating data in files. The map might show all the data on one server, in which case the layout represents a file redirection, or it might show the data spread across multiple servers, allowing the client to access the file in parallel.

While the generic 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 have been approved by the Internet Engineering Steering Group as an IETF Proposed Standard, and the final specification[2] is now in the hands of the RFC editors. That document also describes the pNFS file layout protocol. Mechanisms for access to object[3] and block[4] storage, described in separate documents, have also been approved by the IESG and are under review by the RFC editors.

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

- Very high application performance
- Massive scalability with sustained performance
- Interoperability across standards-compliant application platforms from multiple vendors
- Insulates storage architects from the risks of deploying proprietary technologies

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

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.

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

[2] Shepler, S., Eisler, M., and Noveck, D. (eds.). 2008. NFS Version 4 Minor Version 1. December 15, 2008. http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-minorversion1-29.txt

[3] Halevy, B., Welch, B., and Zelenka, J. 2008. Object-based pNFS Operations. December 15, 2008. http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-pnfs-obj-12.txt

[4] Black, D.L., Fridella, S., and Glasgow, J. 2008. pNFS Block/Volume Layout. December 22, 2008. http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-pnfs-block-12.txt

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 was tracking changes in the IETF specifications and the Linux kernel. In the past 12 months, the specification of pNFS advanced from draft 19 to draft 29 (10 drafts), the block layout specification went from draft 5 to draft 12 (seven drafts), and the Linux kernel moved from 2.6.24 to 2.6.28 (four major releases). The NFS specifications are now finalized, but the Linux kernel will continue to see major releases every three months or so.

Development plans are necessarily flexible, not only to accommodate the 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.

Status and goals

This document outlines CITI's goals in its partnership with EMC for the next 12 months.

A year ago, our goal was 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.

We have substantially accomplished those goals:

- The NFSv4.1 and block layout final drafts documents have been approved as proposed standards by the IESG. Official RFC numbers will be issued soon. This is an enormously significant development.
- The block layout client is fully functional and compliant with the final NFSv4.1 and block layout specifications, with only some protocol nuances remaining to be resolved.
- In response to Linux maintainer and developer feedback, the client now constructs complex topologies in user space using the logical volume manager.
- The implementation is built on Linux 2.6.28, the current stable release. We will rebase to 2.6.29 shortly.

Furthermore, we remain on schedule for the road map we projected a year ago:

- We are on track to help Linux maintainers shepherd code patches into the mainline kernel in 2009 and 2010.
- We expect to see NFSv4.1 and pNFS present in Linux distributions in 2010.

Over the next twelve months, CITI proposes three major themes:

- Address outstanding development issues.
- Construct comprehensive performance tests using standard measurement tools.
- Address interoperability, performance, and scalability issues that arise through testing.

Development tasks

The outstanding development issues center on block layout compatibility with generic Linux or NFSv4.1 code. Tasks are classified as High, Medium, or Low priority.

H1: Coalescing block layouts

The generic layout code does not coalesce adjacent or overlapping layout segments. For example, if a client gets layouts for ranges 0–10 and 11–20, then issues an I/O request for range 5–15, the generic layer fails to find a matching layout, so it requests a layout for the range 5–15, then reissues the request. Although the resulting I/O is correct, the mechanism is inefficient.

To avoid the extra request, the generic layer could coalesce the segments internally or break the I/O into ones that fit the existing layout. Coalescing layouts incurs minimal cost but requires a substantial amount of change to the generic layer.

Related to this issue is the tracking of extent ranges. The block layout code manages ranges in a simple linked list. We need to understand the scalability implications of this and identify a better data structure if necessary.

Overall, we need to experiment with block layouts under representative workloads, measure performance, address scalability issues, improve layout management and the generic layer, and make the case to Linux pNFS developers and maintainers to accept those changes.

H2: Draining I/O
There are many places in the generic client code where we should — but do not — wait for pending I/O to complete before proceeding. We need to provide the software scaffolding for this. This task has high priority because other tasks depend on it.

H3: Submit and shepherd patches
The fundamental metric for success is support for pNFS block layout in the mainline Linux kernel. Acceptance into the mainline kernel is a complex, time-consuming process involving peer review by and consensus among Linux developers and maintainers. Because the outcome of the process can not be guaranteed, the principal deliverable of this project overall is Linux software queued for peer review and acceptance. Accordingly, CITI developers will submit pNFS block client patches for peer review, will respond to inquiries and suggestions, and will provide general support for the software.

It is CITI's understanding that Red Hat Fedora 11/RHEL 6 will branch off the 2.6.31 kernel. The deadline for 2.6.31 is likely to be sometime in June. At this writing, the Linux kernel stands at version 2.6.29, with 2.6.30 about to open. We anticipate that 2.6.30 will include basic NFSv4.1 sessions, with significant work to be done in 2.6.30 and 2.6.31. If all goes well, 2.6.31 (and thus RHEL 6) will have the necessary structure for pNFS, and pNFS developers can begin to submit patches for 2.6.32, which should open in the fall.

H4: Test plan
CITI and EMC will jointly produce a test plan. CITI will execute the plan and provide results to EMC.

M1: Large blocks
The Linux kernel assumes that file blocks are no larger than memory pages, so support for large blocks makes it impossible to use large chunks of the kernel supplied block infrastructure. Furthermore, I/O to block devices uses a pointer on the page structure that the NFS code assumed was available for constructing RPCs, since there was no overlap between I/O to NFS and block file systems (until now).

To work around this problem, we are forced to make a private copy of a number of needed functions. This has maintainability implications and leaves our implementation somewhat painful and hackish. (In our private copy, the role of blocks and pages is reversed; we are left with an implementation that supports block size ≥ page size but no longer supports block size < page size.)

Tackling the block size ≤ page size assumption directly would simplify our code, with concomitant advantages in reliability and maintainability, but this would involve a huge, invasive change to much existing kernel code, which we deem too expensive for this project.[5] Instead, we will need to revisit the existing block layout code to clarify, document, and test it thoroughly.

M2: CB_NOTIFY_DEVICEID
The CB_NOTIFY_DEVICEID handlers are not yet implemented. These functions process CHANGE and DELETE callbacks from the metadata server specifying updates to the device topology. The DELETE function can be processed easily by reverting the metadata server NFS path, but asynchronous changes require the ability to drain I/O, discussed above, and then to make changes to the layout maps. This code must be written carefully and tested thoroughly.

M3: PyNFS maintenance
PyNFS, CITI's Python-based NFSv4.1 client and server, continue to prove extremely valuable for testing completeness and correctness of the Linux pNFS block layout client and the Celerra-based block layout metadata server. CITI will continue to maintain these tools, and will use them for configurable and repeatable performance testing.

[5] A complete rewrite of the buffer cache is not out of the question: since June 2007, Nick Piggin has persisted with his "fsblock" patch, which does away with the block size ≤ page size assumption. Piggin may ultimately succeed in seeing his patch accepted. http://lkml.org/lkml/2007/6/23/252,

M4: DM target
We have tested Haying Tang's user space implementation of support for complex topologies for correctness, but we have not yet tested performance. We will use IOzone to measure performance when I/O is contained within devices and when I/O crosses devices. Once we are satisfied that performance is satisfactory, we will submit patches to the Linux NFS mailing list and work with the NFS-utils maintainer to shepherd patches into the common release.

L1: LAYOUTRETURN
LAYOUTRETURN does not wait for pending I/O to complete, which has some implications to the coalesced layout issue described in H1.

L2: Partial success on I/O
A POSIX I/O operation is allowed to succeed for fewer bytes than requested. The generic NFS client handles partial success by reissuing the RPC after adjusting the offset and count values to reflect the successful I/O, but this model does not fit layout drivers that are not RPC-based, nor does it suggest a way to handle a parallel request that succeeds partially, where the successful I/O is not necessarily contiguous.

In these cases, the block layout driver is forced to fail the entire request and reissue the request using the NFS path through the metadata server. This behavior is correct but potentially slow. Intuition suggests that this kind of failure is rare, in which case we need not do anything to improve the implementation, but we do need to log and monitor this error when it occurs.

Milestones
Completion targets are aligned with the three major interoperability testing events, Bakeathon in June and September, and Connectathon in February.

Task	Description	Priority	Dependencies	Target dates
H1	Coalescing block layouts	High		June '09 Bakeathon
H2	Draining I/O	High		September '09 Bakeathon
H3	Submit and shepherd patches	High	Linux 2.6.32 kernel	Begin after September '09 Bakeathon
H4	Test plan	High		Produce plan: June '09 Bakeathon Initiate testing: September '09 Connectathon Complete testing: February '10 Connectathon
M1	Large blocks	Medium		Ongoing
M2	CB_NOTIFY_DEVICEID	Medium	H2	February '10 Connectathon
M3	Python maintenance	Medium		Ongoing
M4	DM target	Low		February '10 Connectathon
L1	LAYOUTRETURN	Low	H2	February '10 Connectathon
L2	Partial success on I/O	Low		Monitoring

Risks

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

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 to the Linux kernel 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 iterative, 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 and these collegial relations lead naturally to shared goals and understanding. However, it must be emphasized that in their roles as Linux maintainers, the maintainers exercise independent judgement in assessing the means to advance the Linux kernel, so that when all is said and done, the decision on whether patches 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 may resist including CITI's block layout driver. CITI obtained funding from IBM to develop a GFS2-based metadata server and produced a minimal implementation before funding ran out. CITI is assisting the brave users experimenting with the code and is trying to identify funding for further development.

As an alternative, NetApp has developed spNFS, a user-level pNFS server potentially acceptable to Linux developers and maintainers. Similarly, CITI is advocating its Python-based block layout server as a vehicle to promote acceptance of pNFS block layout client patches into the mainline Linux kernel.

The issue remains unresolved. 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.

Reporting

CITI will provide a written status report to EMC every other month.