Statement of Work

NFSv4 extensions for performance and interoperability

Center for Information Technology Integration
February 15, 2007 – February 14, 2008

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 implement pNFS block layout capability in the Linux NFSv4.1 client. CITI will implement a pNFS driver capable of operating on a single file in parallel using block I/O. The driver will implement storage class-independent pNFS extensions and layout management API following specifications from the emerging draft specifications of pNFS protocol operations and block based pNFS. CITI will demonstrate the pluggable block-layout driver prototype co-existing with other pluggable prototype layout drivers (file, pVFS2) in the Linux pNFS client.

To advance pNFS standardization in the NFSv4.1 extension draft, CITI will implement the sessions and directory delegation in the Linux client and server. CITI will implement pNFS features in the Python-based PyNFS server to aid prototyping and testing.

The fundamental 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 deliverable is Linux software queued for peer review and acceptance.

Task 1: pNFS

pNFS, a standard feature for minor extension NFSv4.1, is in the middle stages of definition by the IETF. pNFS shows great potential to extend NFS client performance by allowing a client direct access to storage servers. Furthermore, a pNFS client can simultaneously access a collection of storage servers in parallel, multiplying the potential for high-performance access, limited only by the cross-sectional bandwidth of the storage area network.

The draft specification for pNFS [1] specifies extensions to NFSv4 and direct I/O to file-based storage. The draft specification for pNFS block-based access [2] specifies pNFS operations for block and volume based storage. EMC's Celerra Highroad file system is similar in spirit to the pNFS block-based architecture and serves as a reference for the implementation of the pNFS block layout driver.

Development has begun on the Linux NFS client extensions required to provide storage-class independent pNFS functionality using pluggable storage-class specific extensions, called layout drivers. The present storage class-independent pNFS client supports file and pVFS2 layout drivers. A block-layout driver requires a more complicated layout driver interface.

---

1 S. Shepler, M. Eisler, and D. Noveck (Eds.), "NFSv4 Minor Version 1," draft-ietf-nfsv4-minorversion1-08.txt. (October 22, 2006; expires April 25, 2007)

2 D.L. Black, S. Fridella, and J. Glasgow, "pNFS Block/Volume Layout," draft-ietf-nfsv4-pnfs-block-02.txt (February 21, 2007, expires August 2007)

Task 1.1: I/O path

The I/O path in the current implementation goes directly to the block device cache, bypassing the NFS page cache. This introduces some size mismatch problems when writing partial blocks. CITI will change the I/O path to use the NFS page cache, which is already equipped to manage zero filling of partial block writes.

Task 1.2: I/O path

The NFSv4.1 specification requires that errors in direct I/O fall back to conventional RPC requests to the NFS server. Task 1.1 changes the I/O path to use the NFS page cache, which is already equipped to issue conventional RPC requests. In this task, CITI will eliminate the complex error handling introduced to support an I/O path that went directly to the block device cache.

Task 1.3: GETDEVICELIST, GETDEVICEINFO, and LAYOUTGET update

The recently revised block layout specification retrieves the device topology with OP_GETDEVICELIST instead of OP_LAYOUTGET. This requires changes to the existing block layout driver in CITI's client and EMC's server. In lieu of a server that understands the new XDR formats, CITI will use the PyNFS suite to test the modified driver.

To promote acceptance by Linux kernel maintainers, CITI will investigate the potential to reuse the topology management functionality in the existing Linux kernel multi-device driver (driver/md), which maps a file offset into a device and block number.

Task 1.4: Client layout cache

The generic (i.e., storage class independent) pNFS client implementation supports a per file whole file layout cache designed to serve file layouts. The block layout driver expands this simple cache to hold two layouts per file based on I/O mode. This is sufficient for simple I/O processing, but inadequate for complex block layouts. CITI will extend the layout cache to support complex topologies.

The object layout driver faces the same problem; CITI will investigate the potential to leverage our efforts by defining and implementing a common mechanism for caching complex layouts.

Task 1.5: Client reboot recovery

Client recovery from server reboot follows the pattern established for recovery of other client state: a client reclaims its layouts after the server grace period ends. However, if the client is holding dirty blocks for a newly allocated extent, then the client should write that data (using NFSv4) during the grace period. In the event that the data no longer resides in the client cache, then the client has only one option: issue a LAYOUTCOMMIT with the reclaim flag during the grace period and hope that the pNFS server can rebuild the block layout from that. To guarantee freedom from data loss in this case, it is necessary for the client to maintain data in its cache until a LAYOUTCOMMIT succeeds. CITI will extend the layout cache to respond to server reboot.

Task 2: PyNFS block layout server suite

This task extends the PyNFS NFSv4.1 server to include block layout specific functionality. Without integration with iSCSI, the PyNFS NFSv4.1 server will not be able to perform real I/O, but it can test some important features of the client block layout driver:

Task 2.1

CITI will develop a suite of PyNFS NFSv4.1 pNFS block layout volume topology tests using OP_GETDEVICELIST and OP_GETDEVICEINFO. These tests will provide semantically correct volume topology variations to test the block layout client's ability to parse and manage complex topologies. This will require a "switch" on the pNFS block layout client to skip disk signature verification.

Task 2.2

CITI will develop a suite of PyNFS NFSv4.1 pNFS block layout volume tests using OP_LAYOUTGET, CB_LAYOUTRECALL, and OP_LAYOUTRETURN to exercise the layout cache management in the pNFS block layout client.

Task 2.3

CITI will develop a suite of PyNFS NFSv4.1 metadata server reboot recovery tests for the client block layout driver using OP_LAYRETURN with the reclaim flag set.

Task 3: Support latest draft

Full support for pNFS requires that CITI implement these portions of the latest NFSv4.1 minor version:

- Support sessions operations EXCHANGE_ID, CREATE_SESSION, OP_SEQUENCE.
- Replace NFSv4.0 clientid and lock sequencing with sessionid and OP_SEQUENCE sequencing.
- Use OP_SEQUENCE sequencing (to prevent LAYOUTGET, LAYOUTRETURN races).
- Support the sessions callback infrastructure, and use it for CB_LAYOUTRECALL.
- Extend the pNFS client GETATTR processing to a 96-bit attribute bit mask and include all minor version pNFS attributes.
- Rebase the current block pNFS implementation (client and server) to the latest Linux kernel.

Task 4: Directory delegations

NFSv4 clients cache directory contents in the following ways:

- READDIR uses a directory entry cache
- LOOKUP uses the name cache
- ACCESS and GETATTR use a directory metadata cache.

To limit the use of stale cached information, RFC 3530 suggests the NFSv3 time-bounded consistency model, which forces a client to revalidate cached directory information periodically. This revalidation of directory information is wasteful — CITI's analysis of network traces reveals that a surprising amount of NFSv3 traffic is due to GETATTRs triggered by client directory cache revalidation. The NFSv3 model also opens a window during which a client might use stale cached directory information.

To improve performance and reliability, NFSv4.1 introduces read-only directory delegations, a protocol extension that allows consistent caching of directory contents. CITI has begun implementing directory delegations, as described in Section 11 of NFSv4.1 Internet Draft. A modest effort will move that work to completion. Beyond the performance advantages, completing the implementation directory delegations will help move the NFSv4.1 minor extension standardization forward.

Risks

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

Celerra simulator

CITI has struggled to configure the Celerra simulator with volume topologies other than a single VOLUME_CONCAT with a single VOLUME_SIMPLE. Providing CITI with a Celerra system would allow testing the complex volume topologies needed for the pNFS block layout driver. A Celerra system would also allow realistic performance measurement.

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

Linus Torvalds and a select group of kernel subsystem developers and integrators maintain the Linux kernel. Proposed modifications to the kernel are first discussed in the Linux kernel mailing list and other focused Linux mailing lists. After achieving consensus, kernel subsystem maintainers forward patches to Linus, who, after a final review, applies them to the mainline kernel. It is thus vital to have the attention and support of the designated kernel maintainers.

CITI is fortunate to have a relationship with Trond Myklebust — the NFS client maintainer — that is nonpareil: Trond's office is located at CITI and he is a peer to CITI developers and researchers. At the same time, Trond is independent; he is not funded by CITI and is not a developer on CITI's sponsored projects, so CITI has no particular privilege in determining the outcome of the Trond's review. Yet, the collegial relationship leads naturally to shared goals and shared understanding. However, it must be emphasized that at the end of the day, the decision on whether developments are incorporated into the mainline kernel is not CITI's.

Linux kernel requirements

An important milestone that must be achieved before a multifaceted subsystem is allowed into the mainline kernel and utilities is an open source solution for all facets of the system. For example, a client/server system must include an open source client and an open source server.

CITI ran up against this restriction in its multi-year, multi-sponsor effort to enhance the NFS server for cluster file systems: CITI's kernel enhancements 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 requirement for a complete solution affects this project: until the Linux kernel includes a block pNFS server, kernel maintainers and developers will resist including CITI's block layout driver. On the other hand, because CITI's block layout driver is an open source reference implementation of an IETF standard, it may be considered exceptional and allowed for inclusion.

To moderate this risk, CITI will encourage discussion and review of its block layout driver implementation and make a best effort to achieve consensus. If the consumer requirement delays mainline inclusion, the software can be packaged and distributed as a loadable kernel module in the interim.

Milestones and deliverables

Milestone 1, April 30, 2007

- **Milestone M1.1.** T1.1: New I/O path implemented and tested.
- **Milestone M1.2.** T2.1: PyNFS block layout volume topology (needed for T1.3/M3.1) implemented and tested.

Milestone 2, June 30, 2007

- **Milestone M2.1.** T1.2: New I/O path error handling implemented and tested.
- **Milestone M2.2.** T2.2: PyNFS block layout volume (needed for T1.4/M4) implemented and tested.
- **Milestone M2.3.** T3: Sessions callback processing, GETATTR, and rebasing implemented and tested.
- **Milestone M2.4.** T4: Directory delegations implemented and tested.

Milestone 3, August 31, 2007

- **Milestone M3.1.** T1.3: Complex volume topology and file offset mapping to device and block implemented and tested against PyNFS server.
- **Milestone M3.2.** T2.3: PyNFS metadata server reboot recovery (needed for M5).

- **Deliverable D1.** Sessions, callback processing, GETATTR, and rebasing queued for Linux kernel peer review.
- **Deliverable D2.** Directory delegations implementation queued for Linux kernel peer review.

Milestone 4, October 31, 2007.

- **Milestone M4.** T1.4: Test client layout cache against PyNFS and Celerra servers.

Milestone 5, December 31, 2007

- **Milestone M5.1.** T1.5: Test reboot recovery against PyNFS and Celerra servers.
- **Milestone M5.2.** Linux block client implemented and tested.

Milestone 6, February 14, 2008

- **Deliverable D3.** Linux pNFS block layout implementation queued for Linux kernel peer review.