NFSv4 extensions for performance and interoperability

Center for Information Technology Integration
January 8, 2007

This is a report of work performed by CITI in a project sponsored by EMC. The tasks detailed in the Statement of Work are in gray text. CITI reporting is in black text. Updates to the October 27, 2006 report are shaded in yellow.

Task 1: NFSv4.1 pNFS extension

Implement a prototype block layout driver capable of operating on a single file in parallel using block I/O. The prototype 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.

We can break down Task 1 into the following milestones.

T1 Month 1: Background

Work begins by studying the EMC File Mapping protocol, Celerra HighRoad implementation, and Linux 2.6.15+ SCSI driver interface.

- Install EMC pNFS block-based file service.
- Test NFSv4.0 functionality via NFSv4.0 client to EMC pNFS MDS.

CITI and EMC engineers determined that in order to run EMC's pNFS block-based service, the Celerra simulator on loan to CITI required a hardware upgrade. CITI returned the simulator to EMC on March 27 and received new hardware on April 5.

CITI and EMC engineers then installed and configured the pNFS block service. CITI successfully tested NFSv4.0 functionality with a Linux 2.6.17 client connected to the EMC pNFS MDS.

T1 Month 3: GETDEVICELIST and GETDEVICEINFO

A device ID may need to carry a lot of information, so GETDEVICELIST returns a list of device ID's, and GETDEVICEINFO needs to be called on each ID.

- "Ping" each device to confirm connectivity.
- Respond to changing devices.

We implemented the block-layout OP_GETDEVICELIST XDR and successfully ran against the pNFS EMC simulator.

We ran iSCSI on the Linux pNFS client connected to the pNFS simulator. We developed code for the block layout driver that queries each SCSI disk device present on the client, looking for the disk signature returned in the OP_GETDEVICELIST, but encountered errors. (SCSI reported that we were requesting blocks past the end of the disk.)

We discovered that the simulator was exporting LUNS as a file system, not as a raw disk. After configuring the simulator to export LUNS as a raw disk, the simulator aborted abnormally. Without iSCSI connectivity, we were unable to implement and test OP_GETDEVICELIST.

In mid-July, CITI and EMC engineers discovered that the simulator lacked the iSCSI pass-through feature—Blackbird virtual disks don't emulate enough of the SCSI commands to make this work—so we shipped the simulator back to EMC.

The upgraded simulator arrived in early August. Later that month, we were able to use iSCSI to talk to Blackbird exporting LUNS as a raw disk. By mid-September, our block layout driver was able to issue able to issue OP_GETDEVICELIST XDRs, recognize disk signatures, and ping devices to confirm connectivity.

T1 Month 5: LAYOUTGET, basic read/write

We anticipate some complications involving GETLAYOUT request and retry semantics and multiple layouts (read and read/write layouts) that cover the same range.

- Perform layout processing. Check pNFS layout iomode extent data type, extent coverage of min/desired size.
- Fetch and store data through the read/write interface.

We implemented the OP_LAYOUTGET XDR. This XDR is somewhat complicated because the logical volume topology is expressed with a recursive XDR declaration.

We installed the latest pNFS code from Pete Bixley, which implements LAYOUTGET, but calling LAYOUTGET caused the simulator to abort abnormally.

At the September 13 Bakeathon, Stephen Fridella fixed a configuration problem that prevented LAYOUTGET from working: the FMP service was not running, so FMP memory pools were not initialized. Subsequently, we were able to get GETDEVICELIST and LAYOUTGET working. The Linux pNFS block client then tried to perform direct I/O, which we have not yet implemented.

Stephen also showed us how to configure the simulator to export the two SCSI disks with various volume topologies. We were then able to configure a concatenated volume using the two disks.

In the latter half of October, we revisited the block layout driver to finish task T2, but once again found that iSCSI was causing the simulator to abort abnormally. After several attempts at debugging the iSCSI problem, we found that when we configured the simulator file system to consist of one SCSI disk, we could successfully run iSCSI. This limits us to the simplest volume topology (one VOLUME_CONCAT and one VOLUME_SIMPLE).

In late November, we proceeded with coding the READ/WRITE path for block I/O in the generic pNFS client, using our pnfs_block_get_block routine.

There are three code paths for pNFS I/O in the generic pNFS client.

1. Bypass the NFS page cache.

   For block I/O, we added pNFS-specific struct address_space_operations to the generic pNFS client. The block layout driver instantiates these operations. With this implementation and release 5 of the simulator, we can read and write files directly to the pNFS exported SCSI disks with `cat`, but editing with `vi` aborts. With release 6 of the simulator, `vi` no longer aborts.

2. Use the NFS page cache.

   This path instantiates the existing generic pNFS client operations in the block layout driver. With this implementation, we are able to read files. We are still developing write/commit for this path.

3. Direct I/O.

   We have not coded this path.

T1 Month 7: LAYOUTCOMMIT

- Deal with “holes” in layouts. The four types of layout data — RWdata, Rdata, Invalid-data, and none-data — complicate LAYOUTCOMMIT processing and treatment of holes.
- Investigate pNFS client participation in copy-on-write.

We implemented OP_LAYOUTCOMMIT. We have begun development of the client layout and extent cache to replace the single whole file layout. So far, we have a simple layout cache that divides layouts by `iomode`, creating a layout cache for READ and another for READ_WRITE data. Each cache has a list of extents ordered by file offset and extent state. A successful

OP_LAYOUTCOMMIT results in changing the extent state from INVALID_DATA to READ_WRITE data for the specified byte-range.

OP_LAYOUTCOMMIT tests succeed for small files; for large files, we are using the Connectathon basic tests to debug the code that manages the layout cache extent list.

T1 Month 9: CB_LAYOUTRECALL and LAYOUTRETURN
There is no layout StateID. NFSv4.1 sessions may be necessary here. • Avoid races in LAYOUTGET/LAYOUTRETURN

We have begun implementing the OP_LAYOUTRETURN XDR. We will use the layout cache extent list management code written for OP_LAYOUTCOMMIT to remove returned extents.

T1 Month 10: CB_SIZECHANGED
• EOF processing.

NFSv4 minor version 1 eliminated the CB_SIZECHANGED operation.

T1 Month 12: Reboot recovery and testing

No progress to report.

T1 Demonstration
At the end of the project, CITI will demonstrate the pluggable block-layout driver prototype co-existing with other pluggable prototype layout drivers (file, pVFS2) in the Linux pNFS client.

No progress to report.

Task 2: NFSv4 ACL
Build consensus around solutions to NFSv4 ACL interoperability problems, build a Linux client, demonstrate and document its interoperability with a windows CIFS client and a multi-protocol server.

ACL mapping

Solaris and Linux have implemented NFSv4 ACLs using an Internet Draft that describes how to map between POSIX draft ACLs and NFSv4 ACLs; an early version is kept at

`http://www.citi.umich.edu/projects/nfsv4/rfc/draft-ietf-nfsv4-acl-mapping-02.txt`

On the client side, the mapping is used to support legacy applications and user interfaces. On the server side, it is used to export file systems (such as Linux ext3 and Solaris UFS) that use POSIX draft ACLs.

Interoperability problems observed at testing events were due to minor deviations from the exacting requirements of this Internet Draft. After addressing these problems, most servers now comply with those requirements when mapping mode bits to NFSv4 ACLs. As a result, for most servers, if a file is created by a UNIX client (which always involves setting a mode), querying the ACL yields one that clients using the Internet Draft can map to a POSIX ACL.

Nonetheless, we prefer to invest our efforts in *removing* the need to comply with the Draft rather than investing in bringing server implementations into compliance. For this, we are pursuing two approaches.

First, we are trying to do away with the translation altogether by moving the Linux implementation towards a pure-NFSv4 ACL model. Solaris and others are also doing this—it seems to be the NFSv4 community consensus solution to the translation problems.

On the client side, we implemented usable prototypes of command line and GUI tools that directly manipulate NFSv4 ACLs instead of ACLs translated to POSIX. These tools are available from:

http://www.citi.umich.edu/projects/nfsv4/linux/

On the server side, we have contributed to Andreas Gruenbacher's efforts to support NFSv4 ACLs in the file systems that we export. He has a design document and a series of patches that implement NFSv4 ACL support for ext3. See

http://www.suse.de/~agruen/nfs4acl/

for links to patches and design documentation. We collaborate with Andreas on algorithm design and review his work.

Second, we are modifying our approach to the NFSv4↔POSIX ACL mapping to improve interoperability. While we hope to make further NFSv4↔POSIX ACL mapping unnecessary, this may be difficult, for two reasons:

- We anticipate some resistance from the kernel community, skeptical of the value of NFSv4 ACL support in native file systems. This may retard progress on the Linux server.
- We expect users with large file systems that rely on POSIX ACLs to resist upgrading for some time.

Consequently, our server may also continue to rely on NFSv4↔POSIX ACL mapping for some time.

The strictness of the original NFSv4↔POSIX mapping draft makes it unlikely that an unaided user will generate an NFSv4 ACL that is acceptable to the mapping. For our initial server implementation, we were forced to recommend client tools that perform the NFSv4↔POSIX mapping. This further slows the adoption of the pure NFSv4 ACL tools.

Taking a pragmatic view, we are therefore investing effort in improving the NFSv4→POSIX mapping.

Early versions of the Draft effectively "tunneled" POSIX ACLs over NFSv4 ACLs, mandating the precise form of a POSIX-mapped NFSv4—ACE-by-ACE and bit-by-bit—creating in essence a new protocol that had to be understood by both sides. The slightest deviation from that precise form led to interoperability failure.

More recently, we have discovered ways to map most NFSv4 ACLs to POSIX ACLs while still preserving some semantic guarantees. New versions of the draft therefore recommend accepting most NFSv4 ACLs and sketch algorithms that make this possible; see

http://www.citi.umich.edu/projects/nfsv4/rfc/
draft-ietf-nfsv4-acl-mapping-05.txt

We implemented these recommendations in the Linux server, and they are now included in the mainline Linux kernel. Consequently, the POSIX-ACL-centric client tools we built are no longer necessary to interact with our own server.

As far as we know, the only other implementation performing NFSv4↔POSIX mapping is Solaris. They have shown some interest in adopting the recommendations of the new draft.

Windows

Several steps have been made to improve Windows interoperability.

First, relaxation (or elimination) of the requirements imposed by NFSv4→POSIX mapping improves the likelihood that ACLs set by Windows clients will be understood by UNIX-like clients.

Second, we are modifying the recommended POSIX→NFSv4 mapping to no longer require unnecessary DENYs, which caused problems for the Windows ACL editor. We hope that together with the previous change this will also allow servers to move to a simpler “mode→NFSv4 ACL” mapping that avoids unnecessary DENYs.

Third, we proposed modifications to NFSv4.1 that add Windows automatic inheritance features to the protocol.

Finally, we participate in the effort led by Sam Falkner and Lisa Weeks of Sun to clarify the description of the existing protocol. Improvements include detailed explanations of ACL mask permissions bits and explanations of some common cases where the client is required to enforce permissions. These improvements should aid future implementers.

Despite long discussions, the working group's effort to detail the interaction between mode bits and ACLs has failed to reach consensus. Acknowledging the benefit of presenting uniform behavior across all servers, the working group recognizes that strict uniform behavior is not required for interoperability and has (so far) decided that total consensus may not be necessary.

These discussions did shed light on some problems with our implementation of NFSv4 ACLs, and hence contributed materially to its quality.

Task 2 breaks down into the following steps and milestones.

T2 Month 2: White paper
• Prepare a white paper that details NFSv4 ACL interoperability problems candidate and proposes solutions; circulate to EMC and the NFSv4 community for commentary.

We gathered information on NFSv4 ACL interoperability issues from the NFSv4 and Samba communities at Connectathon 2006 and through the relevant mailing lists. Some of the results of this discussion are summarized in

http://wiki.linux-nfs.org/index.php/ACLs

T2 Month 4: Internet draft
• Publish or contribute to an Internet draft that details NFSv4 ACL interoperability problems and proposes consensus solutions. The solution may involve ACL translation on the multi-protocol server. • Prototype the Linux NFSv4 client according to the proposed specification.

In addition to soliciting feedback through mailing lists and the wiki, we revised the NFSv4↔POSIX ACL mapping Draft. Mapping on the server is no longer required, in part because the Draft recommends a DEFAULT-DENY policy for NFSv4 ACLs, removing one of the most troubling differences between NFSv4 and Windows ACLs.

The NFSv4 minor version 1 draft now separates the ACL attribute into new SACL and DACL attributes. This provides automatic inheritance support, allowing NFSv4 to support new ACL features similar to those used in Windows since Windows 2000.

We contributed various minor edits to the 4.1 ACL section to attempt to encourage improved interoperability, e.g., we discourage "tunneling" other ACL protocols over the NFSv4 ACL protocol of the sort that was tried between Solaris and Linux for POSIX ACLs.

We circulated a draft proposal for automatic inheritance.

Our new NFSv4 client ACL tools, mentioned below, were also enabled in part by the improved mapping changes on our own server.

T2 Month 6: Build, test, iterate
• Refine the NFSv4 ACL interoperability Internet. • Prototype the Linux NFSv4 client according to the proposed specification. • Test setting and retrieving ACLs using the Linux NFSv4 client prototype and a windows CIFS client against the multi-protocol server.

Our ACL test bed is simple: a Windows 2003 client, a Linux NFSv4 client, and the EMC Celerra Simulator. Both clients can mount the simulator and see the file space. The simulator proves to be very unstable: it aborts abnormally with almost every ACL test, even when reading an ACL from one client and then from the other.

Nevertheless, testing the new client ACL tools against multi-protocol servers exposed problems in the Linux client ACL cache and in the mode bits cached with attributes. These bugs are not yet repaired, but with caching disabled, we tested our native NFSv4 ACL tools against all servers (including EMC and other multi-protocol servers) present at the September 2006 Bakeathon and found no problems.

We have not yet performed the suggested test involving Windows CIFS and Linux NFSv4 clients. Since the new Linux NFSv4 client tools deal with the native NFSv4 ACL protocol, we expect simple tests in such an environment to succeed. Testing interoperability of a Linux client using the POSIX-mapping tools and a Windows client might help understand whether improved ACL mapping can make them usable in that context.

We implemented the newer POSIX↔NFSv4 mapping in the NFSv4→POSIX direction on the server. We rewrote the server's POSIX→NFSv4 mapping to no longer use unnecessary DENIES. Server-generated ACLs should also be friendlier to Windows. We are in the testing phase, not yet in the mainline kernel.

We have not attempted to prototype automatic inheritance support in either our client or server.

Maintenance continues on the NFSv4 client ACL tools, which we expect to see in Red Hat development releases soon. The POSIX ACL tools now use the new NFSv4→POSIX mapping and accept any NFSv4 ACL from a server without complaint.

T2 Month 8: Demonstrate ACL interoperability

We are not yet able to demonstrate ACL interoperability.

Task 3: Global name space

This task involves basic NFSv4 name space construction and interoperability with Windows DFS. Build consensus around a common LDAP/DFS AD schema for fs_locations information, demonstrate and document Linux client. CITI will work with Linux kernel developers and maintainers to facilitate acceptance of this effort into the mainstream Linux kernel.

Task 3 breaks down into the following milestones.

T3 Month 1: OpenLDAP schema

- Complete the Linux NFSv4 client and server support for fs_locations.
- Prepare a white paper that describes an OpenLDAP directory service schema for fs_locations information; circulate to EMC and the NFSv4 community for commentary.

Linux client and server support for fs_locations is complete. We have a candidate LDAP schema for fs_locations. We are preparing a white paper.

T3 Month 2: Investigate AD

- Investigate the use of DFS Active Directory junction schema for fs_locations information.

Using Windows 2003 Active Directory, we configured a DFS junction namespace populated with a Windows 2003 file server and the EMC Celerra simulator. We discovered that the Active Directory LDAP junction schema stores referral information in an unstructured “binary blob” of type OCTET_STRING, and is therefore not suitable for querying fs_locations information.

We are able search the unstructured blob in an ad hoc way and find the file server + share name list for a junction, but Active Directory does not store the share name to pathname translation

needed for fs_locations. We will be forced to query the CIFS server for the path name associated with a share.

Linux Samba code indicates that there is a CIFS call to Active Directory that returns the desired file server + share name list of a DFS junction. In order to pull the fs_locations information from an Active Directory DFS back-end, we will need to switch from making Active Directory LDAP queries to Active Directory CIFS queries followed by a CIFS query to each server to resolve the share pathname.

It does not appear that Active Directory and Open LDAP can share junction and fs_locations schema without adding to the Active Directory schema.

T3 Month 3: Demonstrate client interoperability
• Demonstrate interoperability of the Linux NFSv4 client with the Linux NFSv4 server and EMC server using the same OpenLDAP schema. (This may involve adding to Active Directory junction schema.)

No progress to report.

T3 Month 6: Demonstrate client and server interoperability
• Demonstrate a Windows DFS client and a Linux NFSv4 client viewing the same file namespace with an EMC server and an Active Directory back-end.

No progress to report.