Project Status

NFSv4 Extensions for Performance and Interoperability

Center for Information Technology Integration

This is a report on the status of CITI's EMC-funded pNFS development project as of January 29, 2009. Items marked in blue reflect change from the November 21, 2008 report.

Sessions in the generic Linux pNFS client

The Linux Sessions implementation is a broad community effort, to which CITI contributes. Sessions code is currently being submitted to the Linux NFS maintainers, Bruce Fields and Trond Myklebust, consuming much of the attention leading Sessions implementers, Andy Adamson and Benny Halevy. As this proceeds, much of the remaining community effort is stalled while we wait for consensus. In particular, this affects our S1–4 and C1–5 tasks.

Task	Description	Status
S1	Session recovery.	This task was complete, but will be revisited when the Sessions integration is complete.
S2	Callback channel.	This task was complete, but will be revisited when the Sessions integration is complete.
S3	NFSv4.1 back channel security using machine credentials.	To provide for back channel security, we added support for machine credentials in the SETCLIENTID call. This makes it possible for the callback client to establish a secure channel to the corresponding principal on the callback server. Patches were committed to Linux 2.6.26-rc1. The 4.1 GSS security framework on the backchannel is being rewritten by Ricardo Labiaga (NetApp). We are working on extending the RPC upcall mechanism so that the callback client can acquire appropriate credentials from gssd. Patches were posted to the linux-nfs mailing list and are under discussion.
S4	NFSv4.1 security using secret state verifiers.	We now have a working Python implementation to test against. See Appendix II for details on SSV progress.

Other generic pNFS client issues

Task	Description	Status
C1	LAYOUTGET, LAYOUTRETURN, and CB_LAYOUTRECALL.	LAYOUTGET and LAYOUTRETURN are complete. We need to address a layering issue: the generic layer is unable to merge adjacent or overlapping layouts, so it sends more LAYOUTGET requests than it needs to. The block layer handles this under the covers, but it would be more efficient to merge them in the generic layer. We have a general framework and a draft implementation of CB_LAYOUTRECALL. Testing with the LSI server exposed some issues related to draining pending I/O in the generic session code; addressing the issue is deferred while the code stabilizes. We are currently preparing to test with the new EMC image that uses CB_LAYOUTRECALL.
C2	CB_RECALL_ANY, RECLAIM_COMPLETE, and CB_RECALLABLE_OBJ_AVAIL.	No progress to report. (So far, the NFSv4.1 development community is deferring work on these non-critical elements.)

Task	Description	Status
C3	Integration of block layout requirements into generic client.	This task is under way and ongoing. The main pNFS branch now includes appropriate hooks for the block driver in the write path.
C4	Implement new NFSv4.1 draft 19–21 pNFS features and behavior.	Layout stateid is under active development in the NFSv4.1 development community, with Andy Adamson (NetApp) leading the way. Basic stateid functionality exists, but further work is deferred pending Sessions integration. Device notification is under active development in the pNFS development community, with Marc Eshel (IBM) leading the development activity. Draft rewrites have simplified this task considerably by eliminating the ADD operation. XDR formats have been worked out and we have an initial implementation of the generic client and server processing code. Generic code exists to handle DELETE notify, but not CHANGE notify.
C5	Reboot recovery.	This task was nearly complete, until the NFS maintainers simplified state management in Linux 2.6.29, so the reboot recovery interfaces must now be revisited.

Block layout module

Task	Description	Status
B1	Rebase the implementation from block draft 3 to block draft 6.	We are at draft 12, which was approved by the IESG as a proposed standard, so this task is complete.
B2	Extend the block layout implementation to support large server block sizes.	This task is complete.
B3	Block layout client implementation based on architectural review.	We are conducting final review and integration of Tang Haiying's user-space disk scanning code, which replaces CITI's kernel-based prototype. Haiying's code recursively constructs a logical volume in user space and passes that information to the kernel.
B4	Support for complex volume topologies using the Linux device mapper (dm) needs to be reviewed to meet performance and quality requirements.	We are ready to begin comprehensive testing of Haiying's code.
B5	Extend the layout cache implementation to support at least two devices.	We have a working implementation that needs further testing.
B6	Extend the device mapper to support the asynchronous CB_NOTIFY_DEVICEID callback operation.	Block-specific device notification depends on generic device notification, which now exists (Task C4). We are ready to begin work on this task.
B7	The block layout client must implement a timed lease I/O fencing mechanism to insulate against network partition.	We are reviewing Haiying's code for this task.

PyNFS

Task	Description	Status
P1	Update PyNFS client and server to support new protocol features in the latest drafts.	The PyNFS client and server now support the latest drafts (minorversion1 draft 29 and pnfs-block draft 12).
P2	Enhance the block server implementation to pass full Connectathon tests.	The PyNFS server passes all Connectathon NFSv4 and non-pNFS NFSv4.1 tests except for the large file test. We now have a prototype implementation of a "real" file system that supports read, write, and file creation.

Milestone summary

The following tasks were projected to be complete by the May 2008 Connectathon.

Task	Description	Status
S1	Session recovery	Complete, revisiting
S2	Callback channel implementation	Complete, revisiting
B1	Block layout draft 6	Complete
B2	Server block sizes greater then 4 KB	Complete
B3	Revisit block layout client implementation based on architectural review	Nearly complete

The following tasks are projected to be complete by the Fall 2008 Bakeathon.

Task	Description	Status
S3	Back channel security using machine credentials	Under way
C1	LAYOUTGET, LAYOUTRETURN, and CB_LAYOUTRECALL	Nearly complete
C2	CB_RECALL_ANY, RECLAIM_COMPLETE, CB_RECALLABLE_OBJ_AVAIL	No progress
P1	PyNFS block client and server support latest drafts	Complete
P2	PyNFS block server passes full Connectathon tests, prototype file system.	Nearly complete

The following tasks are projected to be under way by the Fall 2008 Bakeathon.

Task	Description	Status
C3	Integration of block layout requirements into the generic client	Ongoing
C4	Draft 19–21 pNFS features and behavior. See Appendix for status.	Under way
B4	Complex volume topologies	Nearly complete
B5	Copy-on-write	Nearly complete

The remaining tasks are projected to be complete by the end of the project.

Task	Description	Status
S4	NFSv4.1 security using secret state verifiers	Under way
C5	Reboot recovery	Complete, revisiting
B6	CB_NOTIFY_DEVICEID	Under way
B7	Timed lease I/O fencing mechanism	Nearly complete

Appendix I

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

Organization	People	Role
CITI	Bruce Fields	Code review
	Peter Honeyman	Advisory
	Fred Isaman	pNFS block layout client PyNFS NFSv4.1 test suite PyNFS pNFS block client and server
	Olga Kornievskaia	SSV GSSAPI Long-haul WAN performance for NFSv4.1
	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
	Haiying Tang	pNFS block layout client
	Mario Wurzl	Advisory
IBM	Marc Eshel Dean Hildebrand	pNFS file layout client Generic pNFS client and server
NetApp	Andy Adamson	Generic pNFS client and server
	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: Notes on SSV

NFSv4.1 clients and servers need a secure way to manage state so that a malicious or errant client can not interfere with another client's state management operations, e.g., Alice should not be able to close Bob's open files. One way to provide this level of security is with "machine credentials" such as a Kerberos keytab or a Public Key Certificate, however these credentials must be manufactured in advance, which is not always convenient. The SSV GSSAPI[1] mechanism supports secure state management without prearrangement. Furthermore, the SSV mechanism provides a finer granularity for protection by preventing clients on the same host from interfering with one another.

An SSV (or Secret State Verifier) is a unique secret key shared by an NFSv4.1 client and server that serves as the secret key for the SSV GSS mechanism. As with any GSSAPI mechanism, the SSV GSSAPI implementation includes methods for key agreement, for integrity protection, and for privacy protection. SSV is unusual in that it does not have a context establishment method; instead security context is established with the NFSv4.1 EXCHANGE_ID and SET_SSV operations.

CITI has developed a Python implementation of SSV that we use to test the ongoing Linux SSV implementation. At this writing, no other SSV implementations exist, to our knowledge.

In CITI's Linux SSV implementation, we have a working SET_SSV operation on the client and have implemented GSS_GetMIC and GSS_VerifyMIC functions for Message Integrity Checking. We are currently working on the GSS_Wrap and GSS_UnWrap functions. Some work remains:

- Server side Set_SSV operation
- Multiple versions of SSV
- Multi-page data input to the SSV GSS functions
- Invoking Set_SSV for every new principal

[1] Linn, J. 2000. Generic Security Service Application Program Interface Version 2, Update 1. RFC 2078. The Internet Society.