Center for Information Technology Integration

Statement of Work

Linux NFS requirements for cluster file systems and multi-protocol servers

Prepared for Google, Inc.

Introduction

CITI proposes to improve support in Linux for exporting NFS from cluster file systems and to ride herd on Linux issues related to multi-protocol file servers. Specific areas of concern include POSIX locks, the Linux lease mechanism, RPC reply caching, and ACL interoperability.

Exporting a cluster file system with NFS improves availability, load balancing, and I/O bandwidth by allowing simultaneous NFS exports of the same filesystem from multiple cluster nodes, and by providing for migration and failover between nodes.

There is marked interest in engineering NFS for cluster file systems among users and developers of out-of-tree cluster filesystems (such as GFPS and Lustre), but the cooperation and consensus of the kernel community (and acceptance of patches) depend on demonstrating working implementations with in-tree filesystems.

There are currently two in-tree cluster filesystems: OCFS2 and GFS2. OCFS2 lacks support for NFS exports and for POSIX locks, so we are focusing our attention on GFS2. However, GFS2 is also still young and these features are not yet stable. This complicates our work.

We have identified three areas requiring the attention of file system developers: POSIX file locks, leases, and RPC reply caching. In addition, some of the issues that arise when exporting a cluster filesystem occur when two file servers (nfsd and Samba) attempt to export the same filesystem.

POSIX file locks

Previous work engineered NFS to enforce file locks consistently across a cluster filesystems, but work remains for migration and failover.

During migration, locks must be dropped on the source node and reacquired on the target node. We are testing a prototype implementation of this, described at

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

However, this provides only for migration and failover in file systems exported by one node at a time. While this supports high availability and limited load balancing, it does not offer the performance advantages of a cluster file system that can replicate service on multiple nodes. To support migration and failover for file systems exported by multiple nodes, we must prevent other nodes from acquiring locks while they are migrating. We have some ideas on how to do this, which we will add to the wiki. Suffice it to say that it is a challenging problem.

Lock requests from NFSv2 and NFSv3 clients are serviced by the in-kernel lockd daemon. lockd is single-threaded, so it must not block while the filesystem services lock requests. To promote kernel acceptance of a VFS API extension for asynchronous lock requests, we need to instrument and benchmark the extension, and to maintain and possibly extend the interface. We should also consider allowing multiple lockd threads, which would require rewriting some of lockd's data structures and locking rules. The venerable lockd code has often been a source of much confusion for developers working on these problems, so such changes should be welcomed.

Summary of POSIX file lock needs:

- Migration and failover when each file system is exported by exactly one cluster node
- Migration and failover when file systems are exported by multiple cluster nodes
- lockd cleanup and multithreading

The first and third tasks can be substantially completed in CY08. We can make good progress on the second task, but it is likely that much work will remain. Our focus is on NFSv2, v3, and v4.0, but any design should also anticipate v4.1.

Leases

NFSv4 introduces file delegation, which supports aggressive metadata caching on clients. Delegation requires mutual exclusion of opens. We enforce this against local applications using the Linux lease mechanism. The same

mechanism can be used to enforce delegations across a cluster, but the filesystem must be involved in the decision to grant a lease.

Previous work added a lease() call to the VFS. However, more work is needed.

First, leases do not provide all of the semantics needed by NFSv4 file delegations. For example, they do not currently conflict with renames or unlinks of leased files. Thus the lease mechanism must be extended. Some race conditions also need to be fixed.

Second, (at least) one in-tree filesystem needs to provide a real lease implementation. This work can proceed in parallel. However, we expect further refinements of the lease API will be required before the filesystem implementation is complete.

Third, we currently support only read delegations. Write delegations grant exclusive read-write access to a given file, allowing further optimization. A correct cluster-coherent implementation will be challenging: nearly every operation on the file has to break the lease (or, in the case of stat(), cause a getattr callback to the client).

Summary of lease-related work:

- Extend and fix problems with existing lease implementation.
- Implement leases in a Linux cluster filesystem.
- Implement write delegations.
- Implement cluster-coherent write delegations.

The first two items can be completed in CY08, although the second item will require cooperation with GFS2 and/or OCFS2 developers. (We are working in cooperation with Red Hat on the GFS2 implementation.) The latter two tasks are not likely to be completed in CY08.

RPC reply caching

Consider the classic non-idempotent scenario:

- An application deletes a file.
- The client sends a "remove" RPC to the server.
- The server receives the request and removes the file.
- Something prevents the client from receiving the RPC response.
- The client times out and resends the request.
- The server returns an error.
- An error is returned to the application, even though the file was successfully deleted.

To prevent this error, Juszczak introduced the RPC reply cache. As of yet, though, the Linux NFSv4 implementation does not support RPC reply caching, relying on the reliability of TCP instead. (Indeed, RFC 3530 §3.1.1 prohibits RPC retransmission unless the connection is broken.) However, the RPC reply cache is necessary for correct operation across a reboot, and Linux NFS developers are investigating solutions that do not scuttle server performance, such as the use of nonvolatile RAM.

Cluster file systems introduce further requirements: in failover, the client might resend the request to a different server, so we need to investigate ways to distribute the RPC reply cache across the cluster. At the very least, we need to look into transferring the RPC reply cache on migration or failover.

A cluster-wide RPC reply cache also needs work to better adapt it to TCP, higher packet rates, and compound operations:

- The switch to TCP as the default transport along with stricter requirements on retransmissions suggest that there is likely to be a long delay — as much as a minute — between an initial RPC request and a retransmission. If a single reply cache is shared among all clients, there is a good chance that a needed entry will expire before it can be used.
- A client can cycle through the 32-bit XID space more quickly every year, introducing false collisions due to rollover. Other implementations are adding heuristics like matching on the first hundred bytes of the RPC request as well as matching the XID.
- NFSv4 compound operations complicate the decision as to which replies to cache.

The "sessions" feature, new to NFSv4.1, changes the operation of the RPC cache. In particular, sessions allow us to bound the size of the RPC cache, and hence (in theory) to address the expiration and rollover problems. NFSv4.1 also specifies persistent sessions, which survive reboots.

Summary of work related to the RPC reply cache:

- Implement an RPC reply cache that persists across reboots.
- Extend state migration to support RPC reply cache migration.
- Implement the NFSv4.1 RPC reply cache using sessions.
- Implement the NFSv4.1 RPC reply cache using persistent sessions.

These are all challenging tasks; it is unlikely that patches to the mainline kernel will be ready in CY08.

CIFS/NFS integration

For Samba's oplocks and nfsd's delegations to work together we depend on leases that provide semantics that both can depend on. They aren't quite there yet; witness, for example, http://bugzilla.kernel.org/show_bug.cgi?id=9454 as well as the failure of the VFS to break leases on rename or unlink of the leased file.

ACLs are another source of problems. NFSv4 and CIFS have nearly identical ACLs, but Linux filesystems support only POSIX ACLs. This requires complex mapping algorithms in both file servers. Andreas Gruenbacher (with design and code review from CITI) has written preliminary patches providing native Linux support for NFSv4/CIFS ACLs. We could pursue this work further, but persuading Linux developers to accept the (much more complex) NFSv4/CIFS ACL model is an uphill battle.

Summary of CIFS/NFS integration:

- Ensure the Linux lease implementation meets Samba's needs.
- Implement native NFSv4 ACL support.

Milestones

Within the scope of the time and resources of this proposal, it is realistic to expect to accomplish only a few of the tasks listed above. CITI staff will apply effort to the following tasks, with the expectation that substantial progress on them can be reported after six months and one year.

- Migration and failover when each file system is exported by exactly one cluster node.
- Migration and failover when file systems are exported by multiple cluster nodes.
- lockd cleanup and multithreading.
- Extend and fix problems with existing lease implementation.
- Implement write delegations.
- Ensure the Linux lease implementation meets Samba's needs.

CITI will also monitor and report on changes to the status of the following requirements:

- Implement leases in a Linux cluster filesystem.
- Implement cluster-coherent write delegations.
- Implement an RPC reply cache that persists across reboots.
- Extend state migration to support RPC reply cache migration.
- Implement the NFSv4.1 RPC reply cache using sessions.
- Implement the NFSv4.1 RPC reply cache using persistent sessions.
- Implement native NFSv4 ACL support.

Deliverables

Although the fundamental metric for success is integration of patches into the mainline Linux kernel, acceptance of patches is a process involving peer review by and consensus among Linux developers and maintainers, so the principal deliverables are open source Linux software submitted for peer review, an open discussion of technical issues that arise, and demonstrated effort at responding to technical criticism and commentary.