State Abstraction Layer (SAL)

Design Document

Authors : Philippe Deniel (philippe.deniel@cea.fr)
Adam C. Emerson (aemerson@linuxbox.com)
Matthew W. Benjamin (matt@linuxbox.com)
Peter Honeyman (honey@citi.umich.edu)

Version 0.5

Abstract : This document describes the current work in progress on creating the SAL, a new abstraction layer in Ganesha dedicated to providing a generic (and possibly distributed) state management.

1 The Big Picture behind SAL

1.1 Why a new Layer?

NFS-GANESHA started as an NFSv3 daemon and so was a stateless. As it grew to support protocols like NFSv4, NFSv4.1, and NLMv4 (an ancillary protocol designed to managed locks with NFSv3), it became necessary to seriously consider a design for state management. At present, state management is spread across several modules: the FSAL has lock-dedicated calls, Cache_Inode manages state (locks and share reservations), with NFSv4 and NLMv4 state management layered under them.

The trouble is that, in addition to being uncoordinated, state management is incomplete and linked to a specific local implementation. This is unsuitable for coordinating multiple MDSs or the state propagation between the MDS and DS as required for proper operation of pNFS.

1.2 Aspects to be addressed by SAL

The SAL should be called and used each time states are to be managed. This covers the following cases:

- NFSv3 locks, covered by NLMv4 implementation
- NFSv4 share reservations
- NFSv4/NFSv4.1 locks
- NFSv4/NFSv4.1 file delegations
- NFSv4.1 directory delegations
- NFSv4.1/pNFS layouts

The SAL should be responsible for state recovery described in the NFSv4.x RFCs. This recovery API will be called by upper modules of nfs-ganesha when the daemon reboots.

1.3 Who will require the SAL's service ?

The primary client of the State Abstraction Layer will be cache_inode. Calls to cache_inode that mutate state (open, close, getlayout, etc.) will perform an FSAL operation and change state in one operation (that is, an open state should never be added if the FSAL operation fails.) Proper operation of LAYOUTGET and LAYOUTRETURN require that the FSAL be provided with limited access to state management (possibly through callbacks.) The FSAL must also interact with state granting and recall operations (deciding when a delegation should be recalled, potentially determining when a delegation or directory delegation should be granted, etc.) Top-level protocol operations should also be able to perform 'pure state' operations, such as querying the validity of or freeing a stateid.

2 Special Concerns

2.1 NLMv4/NFS4 Interaction

The NLMv4 protocol adds stateful locking to the stateless NFSv3. The current method to provide proper semantics in interaction between these two is to synthesize open and lock states (using the oh element in the NLM lock structure as a state owner) and insert them as if they were NFS4 states into the table. Normal NFSv3 read and write operations will check for conflicting reservations and increment/decrement a semaphore to ensure proper operation with share deny semantics.

2.2 Locks

The SAL will not manage individual locks, instead associating some FSAL specific (or perhaps separate lock-manager specific) data structure with a given stateid (that is, a different lock_owner/filehandle/clientid triple). This avoids the penalty of a double-test for on every lock (once from the state manager and once from the FSAL) and allows the filesystem to implement lock dependence and other semantics.

2.3 Lease Expiration

The SAL as currently designed does not handle lease expiration. Proper handling of lease expiration requires calls into Cache_Inode or the FSAL itself to free all resources associated with a given state. A process, integrated into some other system (maybe a generalised background housekeeping thread) should iterate the states of expired clients and free them.

2.4 Wants

A wants table is related to state, however when to record a want, when to grant a want, and which want to grant are the concern of higher level systems in Ganesha. The design of a wants table has been postponed until these issues are addressed and some notion of interaction and interface can be defined.

3 Functionality

The SAL provides a mechanism, but mostly avoids policy. To the extent possible, it will refuse to grant conflicting states or violate RFC requirements, but enforces no other constraints. It supports the granting, upgrading, downgrading, querying, and destroying of share states (as well as some convienience functions for fast checks and NFS2/3 reads/writes.)

It supports creation, query, and destruction of delegations and directory delegations.

It supports the association of filesystem specific lock data with NFS protocol state identifiers.

It supports the granting, changing, and destruction of layouts.

It also supports iteration, retrieval, and locking operations. (The current design assumes that one must serialize access to a state, but multiple NFS4 clients possessing open state may write simultaneously unless one holds a lock.)