Center for Information Technology Integration
Report to PolyServe, Inc.

NFSv4 for Parallel File Systems

September 2004

Task 1 Deliverable 1

Analyze NFSv4 server state and determine those state elements that require support from the underlying file system.

Background

NFSv4 servers maintain state elements – ClientIDs, StateIDs, sequence numbers – and structures containing these elements. An NFSv4 server exporting a shared file system must coordinate its view of these elements with other servers. If the shared file system already has the means to coordinate shared state, an enticing option is to use that mechanism to manage NFSv4's state sharing requirements as well. Furthermore, by adding this capability at the level of file operations in existing VFS subsystems, we also coordinate local and multi-protocol access.

Assumptions

We assume that cooperating NFSv4 servers export no more than one parallel file system, and that the parallel file system is data and lock coherent across multiple servers.

We also assume that for any given file system, a client mounts and uses one server for all accesses to that file system, unless forcibly migrated. A client that migrates from one server to another voluntarily is not guaranteed consistent access; a client that migrates at the behest of a server is guaranteed a consistent view.

Linux NFSv4 server state overview

The following state elements reside in memory on an NFSv4 server and may require support from the underlying file system.

nfs4_client structure

SETCLIENTID and SETCLIENTID_CONFIRM operations compel the NFSv4 server to create a client structure if one does not yet exist. The client structure holds ClientID and information describing the delegation callback channel. All NFSv4 server state related to a client can be reached through the associated nfs4_client structure.

Relevant lists: nfs4_stateowners, nfs4_delegations.

nfs4_file structure

The first time an OPEN on a file is confirmed, an nfs4_file structure is allocated to represent that open instance in the Linux kernel. This provides for correct bookkeeping of a file opened by one server thread and closed by another.

Relevant lists: open files (nfs4_stateid), nfs4_delegations.

nfs4_stateowner structure

An nfs4_stateowner structure, created by the OPEN and OPEN_CONFIRM operations or by a LOCK operation, holds a protocol sequence number to ensure non-idempotent behavior for operations that manipulate StateID.

Relevant lists: open files (nfs4_stateid).

nfs4_stateid structure

An nfs4_stateid structure is created when an OPEN or LOCK operation is confirmed. This data structure holds the StateID used to represent an open file or a byte-range lock. An OPEN nfs4_stateid holds a list of LOCK nfs4_stateowners that in turn hold a list of LOCK nfs4_stateids representing byte range locks held on the file.

nfs4_delegation structure

An nfs4_delegation structure, created by OPEN, holds the StateID and file handle of a delegated file. It is destroyed by DELEGRETURN or by lease expiration and is "slaved" to a strict file lock of type LEASE in the VFS lease subsystem.

Processing state for multiple servers: Analysis and next steps

nfs4_client structure

Restricting a client to a single server for a file system – our second assumption – allows the nfs4_client structure representing a client to be stored on a single server; change to ClientID processing or callback channel code is unnecessary. The RENEW operation also requires no changes.

nfs4_stateowner structure

The restriction on sharing of the nfs4_client structure between servers also allows the nfs4_stateowner structure created at OPEN to be stored on one server, so no change is needed for OPEN sequence id checking and OPEN replay cache management.

Because the nfs4_stateowner structure is not shared across servers, several other portions of nfs4_stateid can also be stored on a single server. In fact, only the st_access_bmap and st_deny_bmap nfs4_stateid fields, used by the NFSv4 server to determine share conflicts at OPEN and to test for share compliance for READ/WRITE/SETATTR (truncate), need to be coordinated among servers.

nfs4_stateid structure (OPEN)

When a file is OPENed, the NFSv4 server finds (or creates) its nfs4_file structure, and walks its nfs4_stateid list to check for share conflicts. We augment this with a new call to the exported file system, to which responsibility is given for distributing access and deny bits in the st_access_bmap and st_deny_bmap.

Next step: Ask file system for share/deny access conflicts at open

nfs4_stateid structure (LOCK)

As for OPEN StateID, the byte range lock nfs4_stateowner structure created by LOCK can be stored upon the single server mounted by the client. LOCK sequence number checking and replay cache code remains unchanged. The NFSv4 server uses the POSIX portion of the lock subsystem provided by the Linux VFS. For the multiple server case, the underlying parallel file system must assume the management of POSIX locks. The existing file_operations lock() call should be used.

Next step: Ask file system to provide POSIX locking. Investigate current use of struct file_operations lock call.

nfs4_delegation structure

When a client requests an OPEN, the NFSv4 server may optionally offer a delegation. A conflicting open, which could come from local access, NFS access, Samba access, etc., requires that the delegation be recalled. Servers waiting for the completion of recalled delegations stall clients with NFSERR_DELAY. The Linux NFSv4 server delegation implementation uses the LEASE portion of the lock subsystem provided by the Linux VFS but this must now be revised to query the underlying parallel file system for the delegation status of the file at other servers.

Next step: Ask the underlying file system to check for a delegation recall in progress prior to granting an OPEN or delegation, or initiating a recall.

If the requested OPEN access forces a delegation recall, the NFSv4 server initiates a CB_RECALL on all conflicting delegations. This is currently implemented using the VFS layer break_lease call, which notifies LEASE holders of a conflicting OPEN. The VFS layer makes this determination without consulting the underlying file system.

Next step: Ask file system to notify the NFSv4 server to perform a CB_RECALL upon a conflicting OPEN.

Finally, the NFSv4 server determines whether it can hand out a delegation on the file for the requested OPEN. The VFS LEASE subsystem does this by examining in-memory inode fields to determine if there are any writers (to grant a READ delegation) or any readers or writers (to grant a WRITE delegation). The underlying file system now needs to be consulted to make this determination.

Next step: Ask the file system for information needed for granting a delegation.

If the NFSv4 server decides to grant a delegation, it needs to tell the underlying file system so that the file system can notify the NFSv4 server to recall the delegation later.

Next step: Tell file system that a delegation has been granted.

Task 5 Deliverable 1

Complete the server-side implementation of named attributes, as specified in RFC 3530.

Status

Named attribute support for the Linux NFSv4 server requires translating between two interfaces.

We use the Linux **xattr** API to communicate with the underlying exported file system:

int **setxattr** (const char ***path**, const char ***name**, const void ***value**, size_t **size**, int **flags**)
ssize_t **getxattr** (const char ***path**, const char ***name**, void ***value**, size_t **size**)
ssize_t **listxattr** (const char ***path**, char ***list**, size_t **size**)

These functions set, get, and examine an extended attribute **name** associated with file **path** by passing data for the extended attribute as an unstructured buffer **value** with length **size**.

In contrast, NFSv4 treats named attributes as first-class file system objects. OPENATTR returns the file handle for a given object's named-attribute directory. LOOKUP then returns a file handle to READ the value of a given attribute name. Other NFSv4 operations (READDIR, WRITE, etc.) work as expected in the named attribute directory.

The richness of the NFSv4 model exposes some limitations in the Linux API. For example, **getxattr** doesn't take an offset, so to READ the middle or end of a stored value requires reading and discarding the initial bytes, thus NFSv4 inherits the scalability limits of the Linux extended attribute model. Linux kernel mailing list participants are discussing ways to extend the **xattr** API; proposals under consideration would address many of the compatibility issues.

Trond Myklebust, the Linux NFS client maintainer, faces a similar mismatch on the client side and is working with CITI developers to design the new **xattr** interface and build a prototype.

Meanwhile, we are using the existing **xattr** API to implement server-side support, with the following steps:

1. Generate file handles for extended attributes.
2. Implement OPENATTR.
3. Implement LOOKUP in named attribute directories.
4. Implement file operations (READ, WRITE, OPEN, CLOSE, GETATTR, etc.) on named attribute file handles.
5. Implement directory operations (READDIR and UNLINK, at a minimum) on named attribute directories.

The latest kernel patches available from our website* accomplish steps 1—3, and enough of 4 and 5 (READ and READDIR) to provide read-only access to extended attributes. Work on other operations continues.

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