High level design of inodebits interoperability

February 7, 2008

1 0.1 Introduction

The inodebits feature, documented in inodebits-lock-hld.lyx, is a performance enhancement to parallel directory accesses. In particular, it distinguishes a LOOKUP lock, which protects the cached dentry and owner/group/mode information, from an UPDATE lock, which protects the directory data itself. Bug 8001 is the tracking bug for inode bits interoperability feature.

Since inode bits requires an over-the-wire protocol change, it is necessary to ensure interoperability between an old client/server and a new client/server. Because inode bits locks are a superset of plain locks, it is possible to provide interoperability by converting between plain locks and inodebits locks.

The following table shows how interoperability is maintained. Note that lock coversion is only required when the version of the client and server are different. Naturally, conversion only occurs on the newer system.

	New Client	Old Client
New Server	Full inodebits	Server does lock conversion
Old Server	Client does lock conversion	Full plain locks

While it is possible for a server to automatically recognize an old client by the use of plain locks, it is more difficult for a new client to recognize an old server. Therefore, it is necessary to negotiate inodebits in the client-server capability negotiation phase. This way the client will know that a server reporting the inodebits capability can handle it, and a server NOT reporting it can not.

It should be noted that the lock conversion should happen as close to the messaging as possible. Internally locks are kept in the native format (inodebits or plain) and are only converted for requests coming and going over the wire.

2 0.2 Requirements

Inodebits interoperability should not show any performance degredation in the homogeneous (all inodebits or all plain locks clusters). Because a heterogeneous configuration is presumed to be a temporary or transitional state, there are no explicit performance requirements. However, performance should never degrade below that of a homogeneous plain-locks configuration.

This feature will satisfy today's interoperability requirement that a given version of Lustre must be backward compatible with the previous version, and the software must be able to be rolled back in the event an upgrade is unsuccessful or undesired. In other words, inodebits interoperability should only be necessary between the version it is released in (version N) and version N-1. Once a user is also using N+1, all nodes should be either version N or N+1 and interoperability is not required since all nodes will speak inodebits natively. This feature may be optionally phased out when versions N and N-1 are no longer supported.

3 0.3 Functional specification

3.1 External dependencies

The following existing functions will require modification to support inode bits compatability:

server:

lustre/mds/handler.c:mds_connect() - handles connection from client, must be updated to fetch client capabilities from obd_connect_data structure and storing it in the export exp_connect_flags for later use

lustre/ldlm/ldlm_lockd.c:ldlm_handle_enqueue() - server-side lock request handler needs to convert plain locks from clients that don't support inodebits into an inodebit lock.

ldlm_server_completion_ast() - needs to convert inodebits lock completion response into a plain lock completion for clients that don't support inodebits. Lock handles are generic between plain and inodebits locks, so there may be nothing to do here.

client:

lustre_common_fill_super() - sets INODEBITS capability to be passed to obd_connect() and sent to server in connect message. Server capabilities are returned and stored in the client import structure.

ldlm_cli_enqueue() - needs to convert inodebits lock to a plain lock in the request message if server does not support inodebits

3.2 New functions

The following new functions will be implemented to perform lock conversion as necessary.

ldlm_ibits_to_plain(enqueue_request_msg) - Called by the client to convert an inodebits lock request message into a plain lock request to be sent to an old server. Since the plain lock has a broader scope, this is an acceptable operation.

ldlm_plain_to_ibits(enqueue_request_msg) - Called by the server to convert a plain lock request message received from an old client into an equivalent inodebits lock with both LOOKUP and UPDATE set in the lock policy.

4 0.4 Use cases

4.1 Client use cases

Client connects to MDS, setting INODEBITS capability in the request, and discovers server does not support inodebits in the reply. Capability is kept in client import state.

Prior to sending out lock request, client checks server capability flag in import structure. If inodebits are not supported, client changes request message to a plain lock and throws away the policy data.

Server receives plain lock. When lock is granted and the completion AST is sent back with the lock handle that identifies the inodebits lock on the client.

Client continues as if inodebits lock was granted.

4.2 Server use cases

Server receives client capabilities in client connection request. Server sees that client does not support inodebits, so the server clears the INODEBITS capability in the reply to avoid compatability issues.

Client requests a plain lock. Server recognizes client does not support inodebits, and converts the plain lock into an inodebits lock with UPDATE | LOOKUP bits set. All further processing is done as if it were an inodebits request.

Completion callback from server sends lock handle that refers to the plain lock sent by the client.

5 Logic specification

Since lock conversion is essentially transparent to both client and server, the logic for lock acquisition does not change with this feature.

6 State management

Relevant state is held in a single bit in the imp_connect_data field of the import structure on the client and the exp_connect_data field of the export structure on the server. This bit is used exclusively to determine whether conversion is necessary or not. Once conversion occurs, the state is transparent to the rest of the lock mangement.

7 Protocol, API's, Disk format

The primary purpose of this feature is to ensure over-the-wire compatability for the inodebits feature. Since inodebits and plain locks use the same request/reply message, only a change in the contents of the MDS_ENQUEUE message is required. In particular, an inodebits lock uses a different lock type and policy information to specify that it is an inodebits lock and which bits to lock. This is the only wire-visible protocol change introduced by inodebits.

8 Scalability and performance

In a homogeneous environment (all plain locks or all inodebits), no conversion is required and there is no performance degredation. Using inodebits locks is known to improve performance in certain circumstances over traditional plain locks when no conversion is necessary.

In a heterogeneous environment (mixed plain/inodebits locks), scalability will be slightly more affected, but not as bad as with only plain locks. Since clients are always readers of metadata and can thus all hold compatible locks whether they are plain or inodebits, there is no change. When a write lock is required by the MDS for a directory update operation (create/rename/unlink), some of the "compatible" plain locks may have to be revoked that wouldn't have to be if they were proper inodebits locks. Again, this is no worse than if all clients were using plain locks, and is slightly better because clients using inodebits may not require revokation.

9 Recovery

Inodebits locks are absolutely identical to plain locks in terms of recovery. The same lock timeout rules apply, so if there is no reply for blocking callback within certain time, client node is evicted (and all of its locks are released).

On replay client presents all of its locks to server with special flag set and server just grants the locks again.

10 Alternatives

A possible alternative would be to manage the locks in the form they were requested instead of doing conversion. This idea was dismissed due to complexity and being prone to race conditions.