Unique Client Connections DLD (Bug 9635)

Andreas Dilger

Jan 26, 2006

1 Functional specification

When clients establish a Lustre connection to a server, the server maintains persistent state for that client in the LAST_RCVD file. This includes the client connection identifier (UUID), and information about the last operation performed by this client (*last_xid*, *last_transno*, *last_result*). The server also maintains transient data about the connection in an OBD_EXPORT (connection handle, epoch, connection flags, open file handles, etc).

For any connection the client may be making a new connection, or reconnecting after a server failure. The server must determine if it has received a connection from this client previously, and in that case re-establish the client connection with the same persistent and transient state as it previously had (reconnect), instead of starting a new connection. If the server has no record of this client connection and the client is trying to reconnect then the client must be notified of this and invalidate any saved state that it has (eviction).

The server must ensure that only a single instance of this client is allowed to connect at one time, even though the same client may be resending one or more connect requests after server unresponsiveness. Due to queuing of requests at the server, it is possible that one or more (re)connection requests are processed at the same time or out of order on the server. In all cases, only a single client connection should be allowed to succeed.

2 Use cases

2.0.1 Normal connection

When a client connects the first time, the server needs to assign a free slot in the LAST_RCVD file and mark it as belonging to this client by the client-supplied UUID, and allocate a new obd_export. The obd_export must be added to the list of exports for this target after ensuring that no other export from the same client exists. The client will be sent a connection handle that will allow future requests to be associated with this obd_export.

2.1 Connection after network partition

When a client is attempting to reconnect after a network partition (e.g. dropped client request causes a client timeout) the server must search the current obd_export list to see if this client has connected previously (i.e. an export with the same connection handle and client UUID exists), and in that case re-establish the client's connection to that export. In the network partition case, the server already knows the client connection handle and will refuse connection attempts from the same client UUID if it doesn't have the same connection handle.

Each connection request holds an epoch number (*req->rq_reqmsg->conn_cnt*) that distinguishes one connection from a later one on the client. The connection epoch ensures that the server will not process requests which may have been delayed, but are resent after the client has reconnected.

2.2 Connection after server failure

When a client is attempting to reconnect after a server failure (e.g. reboot and/or failover) the server must first create a list of previously-connected clients by walking the LAST_RCVD file and allocating an obd_export for each valid entry. Similar to the network partition case, when the clients reconnect the server will search the obd_export list to see if this client has connected previously (client UUID exists), and in that case re-establish the client's connection to that export. In the server failure case, the server does not know the client connection handle (which is not persisitent on disk) and instead accepts the client's handle and stores this in the obd_export that was created from the LAST_RCVD file.

2.3 Connection requests processed simultaneously

When a client trying to establish a connection has not heard from the server for some period of time, it will resend its connection request until it receives a reply. In the majority of cases the unreplied connection requests are simply lost in the network because of network failure or because the server was not running. In some rare cases the two (or more) connection requests will be processed concurrently. This can happen if the server is not processing requests promptly (e.g. blocked on a long-held lock or RPC), and the two connection requests are waiting in the request queue and two separate service threads proceed to process both requests at the same time.

In this case, the server must ensure that only a single (re)connection is processed at a time. This is done by locking the obd_export list (*obd_dev_lock*) on the target when searching for an existing connection (as in 2.1 or 2.2) and setting the export *exp_connecting* flag if a matching export has been found. If another connection attempt arrives for this export after the *obd_dev_lock* has been dropped that request will be returned with -EALREADY. Once the first (re)connection is processed, the *exp_connecting* flag is cleared.

2.4 Connection requests processed out of order

In the out-of-order connection request case, it may be that the requests are processed concurrently (as in 2.3), or it may be that the later connection is processed after the first connection is completed. In the latter case, the connection epoch (*conn_cnt*) is used to distinguish between the new connection and the old one. The old connection request is simply replied with -EALREADY as soon as it is seen to hold an older connection epoch.

3 Logic specification

There are two major components that require change in order to ensure unique connections from a client. In target_handle_connect() when searching for an existing export with the same UUID and connection handle we check whether *exp_connecting* is set, and deny the new connection from also establishing a new connection. If this is the first connection for the existing export, we set *exp_connecting* to prevent other connections after dropping *obd_dev_lock*. When the connection is completed, we clear exp_connecting.

```
int target_handle_connect()
{
        spin_lock(&target->obd_dev_lock);
        list_for_each(p, &target->obd_exports) {
                export = list_entry(p, struct obd_export, exp_obd_chain);
                if (obd_uuid_equals(&cluuid, &export->exp_client_uuid)) {
+                       if (export->exp_connecting) { /* bug 9635, et. al. */
+                               CWARN("%s: exp %p already connecting\n",
+                                     export->exp_obd->obd_name, export);
+                               export = NULL;
+                               rc = -EALREADY;
+                               break;
+                       }
+                       export->exp_connecting = 1;
                        spin_unlock(&target->obd_dev_lock);
        :
        :
out:
+       if (export)
+               export->exp_connecting = 0;
        if (rc)
                req->rq_status = rc;
        RETURN(rc);
}
```

The second component changed is class_new_export(), which now takes the client UUID as a parameter. Before adding a new client export, the target export list is searched under *obd_dev_lock* and if the client UUID is found in the list the connection is refused. If no matching export is found it adds the new export (with initialized UUID) to the target export list to ensure no new connections from that client can be processed.

```
-struct obd_export *class_new_export(struct obd_device *obd)
+struct obd_export *class_new_export(struct obd_device *obd,
+                                     struct obd_uuid *cluuid)
{
        :
        :
+       export->exp_client_uuid = *cluuid;
+       obd_init_export(export);
+
        spin_lock(&obd->obd_dev_lock);
+       if (!obd_uuid_equals(cluuid, &obd->obd_uuid)) {
+               list_for_each_entry(tmp, &obd->obd_exports, exp_obd_chain) {
+                       if (obd_uuid_equals(cluuid, &tmp->exp_client_uuid)) {
+                               spin_unlock(&obd->obd_dev_lock);
+                               CWARN("%s: denying duplicate export for %s\n",
+                                     obd->obd_name, cluuid->uuid);
+                               class_handle_unhash(&export->exp_handle);
+                               OBD_FREE_PTR(export);
+                               return ERR_PTR(-EALREADY);
+                       }
+               }
+       }
```

4 State specification

4.1 Resources involved and their state

The client obd_export gets a new state bit *exp_connecting* to indicate that the export is already in the process of (re)connecting, in case there is another connection request. The existing request *conn_cnt* is used in the connection process to determine whether a reconnect request is stale, by comparing it to the current *export->exp_conn_cnt*.

The target *obd_exports* list is now guaranteed to only have a single instance of a given client UUID, which was previously not guaranteed in the face of multiple concurrent first-connect requests.

This change introduces new ways for connections to have an error returned to the client. The existing failure cases (e.g. *rq_reqmsg->conn_cnt* < *export->exp_conn_cnt*) are to be expected because the old connection request is the one getting an error returned. It is now possible to have the new connection request get an error returned. This should not cause any problems on the client because the client will just retry with a higher connection count.

4.2 Locking

The existing *obd_dev_lock* is used to protect the *obd_exports* list during searching. Because this spinlock cannot be held over the entire connection process, the new *exp_connecting* flag causes further (re)connect requests to fail until after the first (re)connect is finished and has cleared the *exp_connecting* flag.

4.3 Recovery

The intent of this change is specifically to address issues with recovery of Lustre connections. Under some circumstances repeated client (re)connection requests have exposed flaws in the server-side connection state machine, including inconsistent saved state (last_rcvd with duplicate UUIDs) and crashes as structures are being modified concurrently by multiple threads. It does not, however, change any of the recovery protocol or state machine.

5 Environment

5.1 Network Protocol Compatibility - New clients, Old Servers

No compatibility issues exist, changes are all internal to the server.

5.2 Network Protocol Compatibility - Old clients, New Servers

No compatibility issues exist, changes are all internal to the server.

5.3 Disk Format Changes

None.

5.4 Documentation Changes

None required. The change is purely internal and does not impact the user interface at all.

6 Alternatives

Several alternatives exist in the implementation of this code.

1. Create the new export with UUID at the time target_handle_connect() has searched the export list and not found a matching export. This was deemed unsuitable because it violates code layering and would mean that callers of obd_connect() need to supply an export instead of that code returning the connection handle for a new export.

2. Create a hash table for existing obd_exports on the target obd to give O(1) lookup of UUIDs in the export list. This is still a possibility if the performance/scale requires it, but since the export list is already being searched for each connect without ill effect this wasn't considered a priority to implement, and would cause increased complexity in already complex code.

3. Holding the *obd_dev_lock* over the entire connect process, instead of using *exp_connecting* to signal an already-connecting export and having to re-search the target export list was also considered impractical. There are potentially many blocking operations in the connection path (allocations, writes to last_rcvd, other locks) that may block the connect. Changing *obd_dev_lock* to be a semaphore instead of a spinlock would be possible, but the longer hold times of this lock would serialize and slow down connection more than holding a spinlock for a short period of time and doing the list walking. This may also negatively impact locking in other parts of the code.

4. Reordering the operations to initially assume a new connection by calling obd_connect(client_uuid) first and then performing a reconnect if -EALREADY is returned from class_new_export() was also considered. The tradeoff is that new connections go faster but reconnections need to walk the list twice. The current implementation has reconnections only searching the list once and new connections have to walk the list twice. This was considered an acceptable tradeoff because reconnection is probably more time-critical during recovery than new connections are during normal operations.