High Level Design for OST I/O locks.

Drokin Oleg <green@clusterfs.com>, Danilov Nikita <nikita@clusterfs.com>

February 11, 2008

1 Functionality specification

Liblustre client cannot participate in the normal DLM locking protocol, because liblustre node can be monopolized for uninterruptible computation for an arbitrary amount of time, and, hence, lock callbacks cannot be executed within any given timeout.

As a result, liblustre client has to operate in the lock-less mode, in which it doesn't cache DLM locks requiring asynchronous AST to be cancelled. This HLD/DLD limits itself to the data (extent) locks only. Elimination of meta-data locks from liblustre client is covered elsewhere.

Required functionality:

- client should be able to request OST to perform LDLM locking on client's behalf for certain READ/WRITE requests if both OST and client support this functionality. That functionality is called „OST-side locking" hereinafter;
- fix OST/LDLM so that other extent locks are never granted to the liblustre client.

2 Logic specification

Upon receiving R/W request flagged for OST-side locking, OST should obtain appropriate lock before actually performing operation and release it after completion.

Glimpse handling code should be changed not to grant extent lock to the liblustre client even when the lock is uncontended.

3 State management

Since data from such requests is not protected by any locks now, client must not put such data into any sort of a cache. Such conditions are met for DIRECTIO read/write requests in llite and for any file i/o in liblustre.

KMS on the client is „protected" by DLM locks in the sense that it is only with the help of client-held DLM lock that KMS can be used to detect short reads (see 8 on page 3).

libsysio provides asynchronous IO APIs (i{,p}{read,write}{,v}) that launch IO and return without waiting for its completion. libsysio client can poll the

status of asynchronous IO by iodone(), and wait for IO completion by iowait(). These APIs do not interfere with OST-side locking, because data-in-transit are not cached: they are stored in buffers private to the calling thread and are not visible to other users.

4 Protocol, APIs, disk format

New flag should be added to OST capabilities mask to show that such a feature is supported. A flag to indicate lock-less i/o request should be added as well. Disk format is unchanged. API is unchanged.

5 Scalability and performance

Lock-less i/o should be a bit faster because there is no need to send RPC asking for a lock, and then there is no need to cancel those locks.

6 Recovery

No changes to the recovery code are necessary. Recovery on liblustre clients will be simpler for since there should not be OST locks granted to such clients at all.

7 Alternatives

- Synchronous lock cancellation.

  As stated in the functional specification the real problem lies in asynchronous ASTs rather than in DLM locks themselves. Instead of delegating locking to the OST, liblustre client can grab DLM locks as usual and synchronously cancel them at the end of the operation. This can be achived by either

  - explicitly calling obd_cancel_unused() at the end of llu_file_rxw() and llu_glimpse_size(), or
  - setting LDLM_FL_CBPENDING bit in ldlm_cli_enqueue() for all extent locks granted to the liblustre client, or
  - setting LDLM_FL_CBPENDING bit on the server (in ldlm_lock_enqueue() or filter_intent_policy()) for all extent locks granted to liblustre client.

  Advantages of this approach:

  - much less intrusive;
  - changes localized to liblustre;
  - possible without server side modifications;
  - no new server capability bit and related interoperability problems.
  - no KMS problems (see 8 on the next page).

Disadvantages:

- OST-side locking might be interesting by itself, because of potential performance improvements;
- not compatible with iread/iwrite APIs (can be worked around by forcing IO completion wait at the end of llu_file_rwx()).

8 Concerns

- Lock-less O_APPEND support may be tricky, because in-page offset of real EOF may not match that of a liblustre idea of file EOF at the time of write since OST receives data and performs I/O in pages. Possible solutions:
  - mark O_APPEND write RPCs and do necessary re-alignment (data-copying) on the server;
  - O_APPEND is rare case, its efficiency is not critical. Appending writes can be handled by synchronous lock cancellation mode described at 7 on the preceding page.
- KMS usage. Lustre read(2) implementation uses KMS to efficiently detect *short reads* (i.e., situation when [offset, offset + count) segment being read is partially outside of the file). This code relies on the DLM lock over [offset, offset + count). There are two problems with that related to the OST-side locking:
  - short-reads can be mis-detected: liblustre client can return zeroes instead of data just written by another client, or it can return data that were just truncated. Race window in this case is actually pretty small, and whole thing looks like „legal“ race, possible on a local file system;
  - traditionally incorrect KMS maintenance resulted in data corruptions, when client C1 didn't see data just written by client C2 (reading zeroes instead of them), and later used these cached zeroes during ->prepare_write() stage of non-page-aligned write, thus overwriting C2 data with zeroes. This cannot happen on liblustre, because there is no caching.