HLD for MDT BASE in CMD

2006-04-13

1 Introduction

In CMD3, we will change many aspects of Lustre, and rewrite quite a bit of the Lustre code. For MDT, it will focus on handling network requests, dealing with locks, etc. It mainly forwards requests to lower layer, and relies on the lower layer to do the real work.

The design work is done by Nikita Danilov <nikita@clusterfs.com>.

2 Requirements

- adapting to the new layer model: device & device operations, object & object operations;
- handling all the network operations, such as connect, disconnect, and others;
- handling all the DLM locking requests; (It will be addressed in another HLD.)
- handling all the fid locking requests;
- forwarding all the meta-data operations to lower layer, and then replying to clients;

3 Functional specification

3.1 Device & device operations

3.1.1 mdt_device

mdt_device presents the metadata target device in Lustre with CMD. It is a kind of metadata device (md_device) and also a generic lustre device (lu_device), and can be

converted from/to metadata device. It contains RPC services, lock and FID related stuff. It also has a pointer to the underlying lower layer device.

mdt device stays at the top of the MDS server stack, receiving requests from the network (metadata requests and LDLM requests), passing these requests to the lower layer or handling them by itself, and then return the reply back to clients and/or other servers. Beside these, mdt is responsible for building the whole server stack: setup the lower layers one by one using their names & configurations; mdt will also do some other global initialization, such as sequence manager.

3.1.2 mdt device type & mdt device operations

mdt_device should provide some generic device operations: such as device type allocation, init & fini operations, object allocation & free. These operations are registered as luster device type operations and lustre device operations.

Though mdt device is a kind of metadata device, mdt does not provide metadata device operations because it is the top device and no other device relies on it.

3.2 Object & object operations

3.2.1 mdt_object

mdt_object represents a Lustre object in mdt layer. It is derived from metadata object (md_object, therefore is also derived from lustre object).

3.2.2 mdt object operations

MDT should register some generic lustre object operations to mdt_object, such as object init & fini. These operations will be called when mdt object is initialized and finalized.

3.3 Network operations

- connect

```
static int mdt_connect(struct mdt_thread_info *info,
                       struct ptlrpc_request *req, int offset);
```

- disconnect

```
static int mdt_disconnect(struct mdt_thread_info *info,
                          struct ptlrpc_request *req, int offset);
```

3.4 DLM locking & intent operations

These interfaces are discussed in another HLD.

- enqueue

```
static int mdt_enqueue(struct mdt_thread_info *info,
                       struct ptlrpc_request *req, int offset);
```

- convert

```
static int mdt_convert(struct mdt_thread_info *info,
                       struct ptlrpc_request *req, int offset);
```

- bl callback

```
static int mdt_bl_callback(struct mdt_thread_info *info,
                           struct ptlrpc_request *req, int offset);
```

- cp callback

```
static int mdt_cp_callback(struct mdt_thread_info *info,
                           struct ptlrpc_request *req, int offset);
```

- intent lock policy handler

```
int mdt_intent_policy(struct ldlm_namespace *ns,
                      struct ldlm_lock **lockp, void *req_cookie,
                      ldlm_mode_t mode, int flags, void *data)
```

3.5 Incoming operations

- Get root object

```
static int mdt_getstatus(struct mdt_thread_info *info,
                         struct ptlrpc_request *req, int offset);
```

- Get object's attributes

```
static int mdt_getattr(struct mdt_thread_info *info,
                       struct ptlrpc_request *req, int offset);
```

- Get object's attribute by name

```
static int mdt_getattr_name(struct mdt_thread_info *info,
                            struct ptlrpc_request *req, int offset);
```

- Set object's extended attributes

```
static int mdt_setxattr(struct mdt_thread_info *info,
                        struct ptlrpc_request *req, int offset);
```

- Get object's extended attributes

```
static int mdt_getxattr(struct mdt_thread_info *info,
                        struct ptlrpc_request *req, int offset);
```

- Get statfs

```
static int mdt_statfs(struct mdt_thread_info *info,
                      struct ptlrpc_request *req, int offset);
```

- Read a page

```
static int mdt_readpage(struct mdt_thread_info *info,
                        struct ptlrpc_request *req, int offset);
```

- Handle re-integrate requests, such as create, link, rename, unlink, etc.

```
static int mdt_reint(struct mdt_thread_info *info,
                     struct ptlrpc_request *req, int offset);
```

- Close a file

```
static int mdt_close(struct mdt_thread_info *info,
                     struct ptlrpc_request *req, int offset);
```

- Client finished writing

```
static int mdt_done_writing(struct mdt_thread_info *info,
                            struct ptlrpc_request *req, int offset);
```

- Pin an object for clients

```
static int mdt_pin(struct mdt_thread_info *info,
                   struct ptlrpc_request *req, int offset);
```

- Sync an object to disk

```
static int mdt_sync(struct mdt_thread_info *info,
                    struct ptlrpc_request *req, int offset);
```

- Generic set info request

```
static int mdt_set_info(struct mdt_thread_info *info,
                        struct ptlrpc_request *req, int offset);
```

- Check quota

```
static int mdt_handle_quotacheck(struct mdt_thread_info *info,
                                 struct ptlrpc_request *req, int offset);
```

- Quota control

```
static int mdt_handle_quotactl(struct mdt_thread_info *info,
                               struct ptlrpc_request *req, int offset);
```

3.6 Lustre llog operations

- Create a llog

```
static int mdt_llog_create(struct mdt_thread_info *info,
                           struct ptlrpc_request *req, int offset);
```

- Destroy a llog

```
static int mdt_llog_destroy(struct mdt_thread_info *info,
                            struct ptlrpc_request *req, int offset);
```

- Read next block

```
static int mdt_llog_next_block(struct mdt_thread_info *info,
                               struct ptlrpc_request *req, int offset);
```

- Read previous block

```
static int mdt_llog_prev_block(struct mdt_thread_info *info,
                               struct ptlrpc_request *req, int offset);
```

- Read header

```
static int mdt_llog_read_head(struct mdt_thread_info *info,
                              struct ptlrpc_request *req, int offset);
```

- Close a llog

```
static int mdt_llog_close(struct mdt_thread_info *info,
                          struct ptlrpc_request *req, int offset);
```

- cat info

```
static int mdt_llog_catinfo(struct mdt_thread_info *info,
                            struct ptlrpc_request *req, int offset);
```

3.7 MDT internal object operations

These helper operations are mainly used by MDT during handling metadata requests. The object lock method is implemented by using local LDLM lock with fids as resource names.

- Find an object by fid (also pump the reference count if success)

```
struct mdt_object *mdt_object_find(struct lu_context *ctxt,
                                   struct mdt_device *d,
                                   struct lu_fid *f);
```

- Put reference on an object

```
void mdt_object_put(struct lu_context *ctxt, struct mdt_object *o);
```

- Lock an object

```
int mdt_object_lock(struct ldlm_namespace *ns, struct mdt_object *o,
                    struct mdt_lock_handle *lh, __u64 ibits);
```

- Unlock an object

```
void mdt_object_unlock(struct ldlm_namespace *ns, struct mdt_object *o,
                       struct mdt_lock_handle *lh);
```

- Find and lock an object

```
struct mdt_object *mdt_object_find_lock(struct lu_context *ctxt,
                                        struct mdt_device *d,
                                        struct lu_fid *f,
                                        struct mdt_lock_handle *lh,
                                        __u64 ibits);
```

4 Use cases

The main responsibility of MDT is to handle requests from clients and other MDTes.

- All metadata requests are unpacked and forwarded by MDT to lower layer CMM. Then replies are returned from lower layer, and then are returned back to callers by MDT.
- All LDLM requests are handled by MDT locally.
- All objects locking requests (internal or external) should follow the correct order.

5 Logic specification

5.1 mdt configuration and startup

MDS registers itself to MGS during its first mount, and MGS constructs config llog from it. This config llog contains startup configuration scripts for mdt, cmm, mdd and

osd. The MGC on the MDS will retrieve config llog from MGS and process these llogs. So mdt device is initialized according to the config llog.

Because MDT sits on top of the MDS server stack, it has some extra responsibility: init some data structure (such as lu_site), constructs the server stack, registers and starts up the mdt service, init sequence manager.

When the client connects to mdt, mdt should return a newly allocated sequence number to it. Please refer to the cmd-fld-dld.lyx.

5.2 Internal handler interfaces

Most of the internal interfaces are listed in 3.3, 3.4, 3.5 and 3.6. These interfaces are easily understandable. All these mdt handlers have the same prototype. For example, getattr handler has the following prototype:

```
static int mdt_getattr(struct mdt_thread_info *info,
                       struct ptlrpc_request *req,
                       int offset);
```

1. Description

   This internal handler gets attributes for a Lustre object, returns the attributes to clients.

2. Parameters

   struct mdt_thread_info *info: contains information for handling this request, such as lu_context, mdt_device, mdt_object, and lock handles.;

   struct ptlrpc_request *req: the request itself;

   int offset: string buffer index.

3. Return value

   zero indicates success, other values indicate error.

5.3 mdt service and request handling

mdt registers mdt service as a ptlrpc service, and start several mdt service threads. mdt registers “mdt_handle” to handle all the mdt requests. Before these service threads call this mdt_handle, they have done some pre-processing. After the mdt_handle returns to these service threads, they will do some post-processing, and reply to the clients. mdt_handle has the following pseudo code:

```
int mdt_handle(struct ptlrpc_request *req)
{
        1. prepare execution context: such as init thread-specific data structure;
        2. do some sanity checking against this request: such as version compatibility;
        3. do recovery issue if necessary;
        4. find proper internal handler according to the request op code;
        5. do some extra work accord to different request type: such as unpacking, find
        6. call the corresponding internal handler if found:such as mdt_getattr;
        7. pack reply message to clients;
        8. cleanup execution context;
}
```

Descriptions:

- We will allocate necessary temprary memory from kernel memory when the service thread startup. These memory is allocated once, and is valid in the "mdt_handle" and its callees. This can reduce kernel stack comsuption and dynamic memory allocation.
- All of the internal handlers are stored in a vector (or called an array), grouped by their categories, with different flags. So the step 4 in the above is to scan the vector and find out the proper internal handler according to opcode of the request.
- according to different category, do some common operations on the request: such as unpacking, lookup and lock the mdt_object on which the request operates. By do these issues in the common wrapper, the internal request handle can focus on the actual work it has to do.

5.4 intent handling

Intent is a kind of lock which will pose some extra action on a specific Lustre object, such as getattr, lookup, or create. mdt registers intent lock call back to the LDLM service, and intent lock is almost served by the LDLM service, except for those extra actions on object. The registered call back should handle these extra action properly, return results to clients.

5.5 open and orphan handle

I have no idea about these issues. It will discussed and filled sometime later.

6 State management

6.1 State invariants

- All network stuff (import and export) on server are now in MDT. No other layer will see network stuff any more.
- All LDLM stuff are handled in MDT layer. All LDLM lock resource are composed from FID sequence and number.

6.2 Locking changes

- All LDLM stuff are handled in MDT layer. All LDLM lock resource are composed from FID sequence and number.

6.3 Wire format changes

Wire format are changed due to FIDs. Refer to the FID HLD/DLD.

6.4 Protocol changes

Protocol is slightly changed due to FIDs. Refer to the FID HLD/DLD.

6.5 API changes

API is slightly changed due to FIDs. Refer to the FID HLD/DLD.

API is also changed due to new lu_device/lu_device data structure and new layer model. Please refer to md-api-dld.lyx

7 Focus for inspections

7.1 Recovery issues

How about recovery issues due to network failure, mds failure, oss failure?

7.2 Status in persistent storage

What information should mdt keep in persistent storage? How is it updated?