MDS on top OSD HLD

Wang Di

2006/01/17

1 Introduction

This HLD describes how the MDD works based on OSD and what kind OSD support is needed for MDD. This project is to be developed during Colibri-4 development cycle (2006.03–2006.09).

2 Requirements

According to the description of the this project, the requirement include

- Build a metadata API that can layer on a slightly extended OBD API. Make sure to be very clear about any state maintained inside the layers that are called, and make the functional specification as clear as possible. The metadata API should in principle be usable by llite. It should not contain any network elements.
- The metadata API should not contain per client data structures.
- Good portability, which means these mdd api will be easy to port to another platform, for example, userspace and database.

3 Functional Specification

According to the requirement, the metadata API will be built in MDD, and exported to MDT and llite, which will include all the functions the Lustre filesystem required. Note: readdir and transaction will be discussed in other HLD.

3.1 Data Structure

According to mds layer definition, in mds, there are two fids lookup, one is name–>fid, which is in MDD, another one is fid–>object, which is in OSD. Both of them will be implemented by IAM. The cache inside IAM will help these two lookup. Furthermore, to help these kinds of lookup, also for attaching

some layer specific things to each layer object, we defined some layer objects structure, These structures are:

```
struct lustre_obj {
   struct fid fid;  /*fid number*/
   int flags;  /*related flags*/
   ...../*layer-specific stuff*/
   void *mdd_obj;
}
struct mdd_obj {
   int flags;
   /*layer-specific stuff*/
   void *osd_obj;
}
struct osd_obj {
   atomic_t ref_count;
   int flags;
   /*layer-specific stuff*/
   struct object *obj;
}
```

When mdt got some request, it will first allocate a lustre_obj, then call lookup method to only get the fid according to the name, if the fid is local object, then it will call related method to fill mdd_obj and osd_obj accordingly. Then in the following process, the object can be gotten directly from local pointer to avoid unnecessary lookup in mdd/osd or some kind of cache. So actually these structures are transfered between the layers to identify the object. The reason that why it did not just choose fid and implement a fid/object cache, will be discussed in althernatives. Note: the object pointer in each layer-object should be opaque to the upper layer.

3.2 MDD API

3.2.1 Create Method

```
1)int mdd_object_create(struct obd_device *obd,
                        struct lustre_obj *obj,
                        struct context *uc_context)
```

1. Description:

   This method is used for creating an object.

2. Parameters:

   - obd: this obd in the metadata stack.
   - obj: the layer-object will be filled after create.

- uc_context: the context of this creation, which should also include create mode, type and fsuid/gid attrs for create_obj method. Maybe also include permission info, or some other related stuff.

3. Return value:

   Return 0 when success, other value for Error.

```
2)int mdd_index_insert(struct obd_device *obd,
                       struct lustre_obj *parent_obj,
                       const char *name,
                       struct lustre_obj *obj,
                       struct uc_context *uc_context)
```

1. Description:

   This is used for inserting fid/name into the namespace of the parent.

2. Parameters:

   - obd: this obd in the metadata stack.
   - parent_obj: the parent obj.
   - obj: the fid in it which will be inserted to the namespace of the parent.
   - name: the name of the being inserted.
   - uc_context: the context of insert.

3. Return value:

   Return 0 when success, other value for Error.

```
3)int mdd_create(struct obd_device *obd,
                 struct lustre_obj *parent_obj,
                 const char *name,
                 struct lustre_obj *obj,
                 struct uc_context uc_context)
```

1. Description:

   This method is used for creating an object according to fid in obj and inserting fid/name into the namespace of the parent_obj. Actually, mdd_create can be implemented by the two separate methods above, but if implementing it in that way, MDT must start the transaction itself, then transfer it to these two methods, which is not good for llite if it running on mdd. To keep the same interface between MDT and llite, and keep transaction handling inside MDD, we also export this method to MDT/llite.

2. Parameters:

   - obd: this obd in the metadata stack.

- parent_obj: the parent obj.
- name: the name of the object being inserted.
- obj: the create object.
- name: the name of the being inserted.

3. Return value:

   Return 0 when success, other value for Error.

3.2.2 unlink method

```
1)int mdd_object_delete(struct obd_device *obd,
                        struct lustre_obj *obj,
                        struct context *uc_context)
```

1. Description:

   This method is used for deleting an object.

2. Parameters:

   - obd: this obd in the metadata stack.
   - fid: the delete object.
   - uc_context: the context of this operation, maybe include some permission info, so on.

3. Return value:

   Return 0 when success, other value for Error.

```
2)int mdd_index_delete(struct obd_device *obd,
                       struct lustre_obj *parent_obj,
                       struct lustre_obj *obj,
                       const char *name,
                       struct context *uc_context)
```

1. Description:

   This method is used for removing fid/name(fid in obj) from the name namespace of the parent.

2. Parameters:

   - obd: this obd in the metadata stack.
   - parent_obj: the parent obj.
   - obj: the delete object.
   - name: the name of the deleted object.
   - uc_context: the context of unlink.

3. Return value:

   Return 0 when success, other value for Error.

```
3)int mdd_unlink(struct obd_device *obd,
                 struct lustre_obj *parent_obj,
                 struct lustre_obj *obj,
                 const char *name,
                 struct context *uc_context)
```

1. Description:

   This method is used for deleting an object and remove it from the namespace of the parent. This method is exported to MDT/llite for same reason as mdd_create.

2. Parameters:

   - obd: this obd in the metadata stack.
   - parent_obj: the parent obj.
   - obj: the delete object.
   - name: the name of the deleted object.
   - uc_context: the context of this operation.

3. Return value:

   Return 0 when success, other value for Error.

3.2.3 attr method

```
1)int mdd_attr_set(struct obd_device *obd,
                   struct lustre_obj *obj,
                   void         *buf,
                   int           buf_len,
                   const char    *name,
                   struct context   *uc_context)
```

1. Description:

   This method is used for setattr to a object.

2. Parameters:

   - obd: this obd in the metadata stack.
   - obj: the object which will be setted attributes.
   - buf: the attr buf which will be set.
   - buf_len: the length of the buffer.
   - name: the attr name.

- uc_context: the context of this operation.

3. Return value:

   Return 0 when success, other value for Error.

   2) `int mdd_attr_get(struct obd_device *obd, struct lustre_obj *obj, const char *name, void *attr_buf, int* buf_len)`

1. Description:

   This method is used to get both attr and EA from the obj according to the attr name. Note: this method will be used for both for attr and EA (also include nlink or some other kind of stuff). The returned value will be stored in attr_buf, *buf_len is the length of the buf. If the attr_buf is NULL, *buf_len will return the length of attr according to the name.

2. Parameters:

   - obd: this obd in the metadata stack.
   - obj: the object.
   - name: the name of the attr, which indicate what kind of attr, the method will return, for example LINK will return link.
   - attr_buf: the buf for return attr.
   - buf_len: the length of attr_buf.

3. Return value:

   Return 0 when success, other value for Error.

3.2.4 Link method

```
int mdd_link(struct obd_device *obd,
             struct lustre_obj *target_obj,
             struct lustre_obj *src_obj,
             const char *name,
             struct context *uc_context)
```

1. Description:

   This method is used for link one object to the target obj, and the object will be insert to the namespace of the target dir.

2. Parameters:

   - obd: this obd in the metadata stack.

- target_obj: the target obj.
- src_obj: the src target obj.
- name: the name of the being deleted object.
- uc_context: the context of this operation.

3. Return value:

   Return 0 when success, other value for Error.

3.2.5 Rename method

```
int mdd_rename(struct obd_device *obd,
               struct lustre_obj *src_parent_obj,
               struct lustre_obj *tgt_parent_obj,
               struct lustre_obj *src_obj,
               const char *src_name,
               struct lustre_obj *tgt_obj,
               const char *tgt_name,
               struct context *uc_context)
```

1. Description:

   This method is used for rename src object to the target object , and the src object will be removed from the namespace of the src dir, the target object will be removed from the namespace of the target dir.

2. Parameters:

   - obd: this obd in the metadata stack.
   - src_parent_obj: the source parent object.
   - tgt_parent_obj: the target parent object.
   - src_obj: the src obj.
   - src_name: the name of source.
   - tgt_obj: the target obj.
   - tgt_name: the name of target.
   - uc_context: the context of this operation.

3. Return value:

   Return 0 when success, other value for Error.

3.2.6 lookup method

```
1) int mdd_lookup(struct obd_device *obd,
                  struct lustre_obj *parent_obj,
                  const char *name,
                  struct lustre_obj *obj)
```

1. Description:

   This method is used to lookup the fid according to the name and parent object.

2. Parameters:

   - obd: this obd in the metadata stack.
   - parent_obj: the parent object of the source object.
   - name: the name of the object being lookup.
   - obj: the fid found will be set in obj.

3. Return value:

   Return 0 when success, other value for Error.

```
2) int mdd_object_lookup(struct obd_device *obd,
                         struct lustre_obj *obj)
```

1. Description:

   This method is used to lookup the object according to the fid in obj.

2. Parameters:

   - obd: this obd in the metatdata stack.
   - obj: the object.

3. Return value:

   Return 0 when success, other value for Error.

3.2.7 object method

```
1) void mdd_object_get(struct obd_device *obd,
                       struct lustre_obj *obj)
2) void mdd_object_put(struct obd_device *obd,
                       struct lustre_obj *obj)
```

1. Description:

   These method is used to increase/decrease the refcount of the object, something like iget/iput.

2. Parameters:

   - obd: this obd in the metatdata stack.
   - obj: the object.

4 Use cases

These API will be used by both llite and MDT to handle namespace operation.

4.1 create usecase

When mdt/llite create an object in the namespace,

- Check whether the object should be allocated locally.
- If not locally, call remote object_create method to create the object remotely.
- If locally, call object_create method to allocate object according to the fid.
- Call index_insert method to insert the fid/name into the parent namespace.

4.2 unlink usecase

When mdt/llite delete an object in the namespace,

- Call index_delete method to delete it from the namespace.
- Check whether the delete object is a local object or not.
- If it is not, call remote destory object to delete the object remotely.
- If it is, call mdd_object_destroy method to delete the object, which is actually called in mdd_object_put, will be discussed 0.5.2.1.

4.3 Attr usecase

When mdt/llite want set/get attributes to an object,

- Call mdd setattr/getattr method to do that according to the fid.
- For changing uid/gid of the object, the quota change log will be writen to quota log. If mdt setattr(uid/gid) successful, mdt will send change uid/gid request to ost to change the uid/gid of the correspondent obj. If this is failed, in recovery, mdt will call mdd_setattr again to restore the uid/gid of the obj.

4.4 Link usecase

When mdt/llite link an object to another target object,

- Check src fid is local or not.
- If it is local, call mdd_link to do link locally.
- If it is not, call remote method to increase the object nlink remotely, then call mdd_index_insert to insert the name/fid to the new target.

4.5 Rename usecase

When mdt/llite rename the src object to the target object,

- Get source and target fids.
- Check whether target parent is local object or not.
- If it is, call mdd_rename method to rename the object.
- If it is not, call remote rename method to rename the object remotely.

Note: Actually, rename in CMD is so complicate, this is just a simple description. The detailed description of Rename is in another HLD.

4.6 lookup intent usecase

When mdt/llite do lookup or revalidate

- Get the fid by name, if needed.

4.7 open usecase

When mdt do open

- If the request is a replay request, call mdd_lookup to try to find the fid according to the replay fid in the parent namespace and in the orphan namespace.
- If it could find, just call mdd_object_lookup to find the object, then call mdd_object_get to increase the refcount of the object, and do sth related with open, then return.
- If it could not, and check the flag in req and see whether it indicate CREATE the obj, if not, return -ENOENT, otherwise create it.
- Call mdd_object_create and mdd_index_insert to create the object and insert it into the namespace, then call mdd_object_get to increase the object refcount.
- Check the permission and other sanity check.
- Create open structure(mfd) to maintain the open state.

4.8 close usecase

When mdt do close

- find open structure(mfd).
- call mdd_object_put to put the object(stored in mfd).

5 Logic Specifications

Since MDD will not only run in linux filesystem, but also will run in userspace or other system, so the API should be completely in terms of fids, no dentry will be seen in these API.

5.1 MDD API implementation

5.1.1 object method

In put get/put object method

- increase/decrease the refcount of the object, similar as iget/iput.
- in put_object method, if the refcount is zero after put, the object will be freed. Note: here, the nlink of the object should be checked too, if it is zero, mdd_object_destory should be called instead to destory the object. The prototype of mdd_put_object should be:

```
int mdd_object_put(obd, lustre_obj)
{
    mdd_obj = lustre_obj->mdd_obj;

    /*put ref_count of object in osd*/
    osd_object_put(osd_obd, mdd_obj->osd_obj);
    if (osd_object_count(mdd_obj->osd_obj) == 0) {
         nlink = osd_obj_getattr(mdd_obj->osd_obj);
         if (nlink == 0)
              mdd_object_destory(obd, lustre_obj);
         else
              mdd_object_free(obd, lustre_obj); /*free the object*/
    }
}
```

5.1.2 create method

In create_object method:

- Start transaction.
- Allocate mdd_obj,

- Create the object.
- call mdd_get_obj to increase the refcount of mdd_obj.
- Stop transaction.

```
int mdd_create_object(obd, lustre_obj, uctxt, *handle)
{
      /*start trans, sanity check */
      mdd_obj = lustre_obj->mdd_obj;
      OBD_ALLOC(mdd_obj, sizeof(struct mdd_obj));
      osd_create_object(obd, &mdd_obj->osd_obj, uctxt, handle);
      osd_get_obj(mdd_obj->osd_obj);
      lustre_obj->mdd_obj = mdd_obj;
      /*stop trans*/
 }
```

In insert_index method:

- Lock the pobj and obj with write_lock mode.
- The fid/name item will be inserted into the namespace.

```
int mdd_insert_index(obd, pobj, name, obj, uctxt, *handle)
{
      /*start trans, sanity check */
      mdd_pobj = pobj->mdd_obj;
      mdd_obj = obj->mdd_obj;

      mdd_get_obj(mdd_pobj);
      mdd_lock(mdd_pobj, WRITE_LOCK);
      mdd_lock(mdd_obj, WRITE_LOCK);
      osd_insert_index(obd, mdd_pobj->osd_obj, name, obj->fid,
                       uctxt, handle);

      mdd_unlock(mdd_pobj, WRITE_LOCK);
      mdd_unlock(mdd_obj, WRITE_LOCK);
      mdd_put_obj(mdd_pobj);

      /*stop trans*/
}
```

In create method:

- Call mdd_create_object to create the object according to the obj.
- call mdd_insert_index to insert the name/fid to the namespace of parent.
- call mdd_put_object to decrease ref_count of the object.

5.1.3 unlink method

In mdd_destroy_object method:

- Lock obj with write_lock mode.
- Check the open_orphan count.
- If it is zero, the object will be destoryed and unlink log will be added for unlink orphan handling.
- If not, the object will be linked to the orphan dir, which will be destoryed when the object is closed.

The prototype should be

```
int mdd_destroy_object(obd, *handle, obj)
{
     /*Start trans, sanity check*/
     mdd_obj = obj->mdd_obj;
     mdd_lock(mdd_obj, WRITE_LOCK);
     if (open_orphan(mdd_obj)) {
           mdd_add_orphan(obd, mdd_obj, handle);
     } else {
           osd_object_destroy(osd_obd, mdd_obj->osd_obj, handle);
           mdd_add_unlink_log(obd, mdd_obj, handle);/*add unlink log*/
     }
     mdd_unlock(mdd_obj, WRITE_LOCK);
     /*stop trans*/
}
```

In this method, mdd_add_orphan will link the object to orphan list, osd_destroy_object will destory the object in osd, mdd_add_unlink_log will add unlink log in mdd for unlink orphan handling.

In mdd_delete_index method,

- Lock pobj and obj with write_lock mode.
- Delete fid/name from the namespace of the pobj.

The prototype should be

```
int mdd_delete_index(obd, pobj, name, obj, uctxt, *handle)
{
      ..................
      mdd_pobj = pobj->mdd_obj;
      mdd_obj = obj->mdd_obj;
      mdd_get_obj(mdd_pobj);
```

```
        mdd_lock(mdd_pobj, WRITE_LOCK);
        mdd_lock(mdd_obj, WRITE_LOCK);
        osd_delete_index(osd_obd, mdd_pobj->osd_obj, name, obj->fid,
                         uctxt, handle);
        mdd_unlock(mdd_pobj, WRITE_LOCK);
        mdd_unlock(mdd_obj, WRITE_LOCK);
        mdd_put_obj(mdd_pobj);

        ..................
}
```

In mdd_unlink method:

- Call mdd_set_attr to decrease nlink of the object.
- call mdd_delete_index to delete the object from the namespace of parent.

5.1.4 setattr method

In setattr method,

- Get the fid of the object.
- Call osd setattr method to do setattr, except attr and EA, nlink will be also set by this method.

5.1.5 link method

In link method,

- Lock src_obj and target obj with write_lock mode.
- Insert src_fid/name into the target obj.
- call osd_obj_setattr to increase the nlink of object.

And the prototype should be

```
int mdd_link(obd, src_obj, tgt_obj, name, uctxt, handle, flags)
{
        ..............
        mdd_src_obj = src_obj->mdd_obj;
        mdd_tgt_obj = tgt_obj->mdd_obj;
        mdd_get_obj(mdd_src_obj);
        mdd_get_obj(mdd_tgt_obj);
        mdd_lock(mdd_src_obj, WRITE_LOCK);
        mdd_lock(mdd_tgt_obj, WRITE_LOCK);
```

```
        osd_insert_index(osd_obj, tgt_obj, name, src_obj->fid,
                         uctxt, handle);
        nlink = osd_object_getattr();
        ++nlink;
        osd_object_setattr(nlink);
        mdd_unlock(mdd_src_obj, WRITE_LOCK);
        mdd_unlock(mdd_tgt_obj, WRITE_LOCK);
        mdd_put_obj(mdd_src_obj);
        mdd_put_obj(mdd_tgt_obj);
        ..............
}
```

5.1.6 rename method

In rename method,

- Lock the source, target and their parents obj according to rename lock policy.
- The source fid entry will be removed from namespace of the source parent.
- The target fid will be removed from target parent.
- The source fid entry will be inserted into the target dir.
- The target will be destroyed.

The proto type should be

```
int mdd_rename(obd, src_pobj, tgt_pobj, sobj, sname, tobj, tname, uctxt)
{
     /*start sanity sanity check ....*/

     /*get mdd_sobj, mdd_tobj, mdd_src_pobj, mdd_tgt_pobj */
     mdd_get_obj(mdd_sobj);
     mdd_get_obj(mdd_src_pobj);
     mdd_get_obj(mdd_tobj);
     mdd_get_obj(mdd_tgt_pobj);
     /* we should lock the fid here according to the RENAME locking HLD*/
     mdd_rename_lock(src_pobj, tgt_pobj, sobj, tobj);

     osd_index_delete(osd_obd, mdd_pobj->osd_obj, sobj->fid, sname, handle, uctxt);
     osd_index_delete(osd_obd, mdd_pobj->osd_obj, tobj->fid, tname, handle, uctxt);
     osd_index_insert(osd_obd, mdd_pobj->osd_obj, sobj->fid, tname, handle, uctxt);
     osd_obj_destroy(osd_obd, mdd_pobj->osd_obj, mdd_sobj->osd_obj,
                     tname, handle, uctxt);
```

```
        mdd_rename_unlock(src_pobj, tgt_pobj, sobj, tobj);
        mdd_put_obj(mdd_sobj);
        mdd_put_obj(mdd_src_pobj);
        mdd_put_obj(mdd_tobj);
        mdd_put_obj(mdd_tgt_pobj);
        ..............
}
```

5.1.7 lookup method

In lookup method,

- lookup the fid according to the name by osd IAM API.
- allocate lustre_obj, and put fid in the obj.

In lookup_object method,

- lookup the object according to the fid, and fill the mdd_object and osd_object in the lustre_object.
- call osd_get_object to increase the ref_count of the object.

5.2 MDD Lock

As for mdd lock, it will lock the object by fid with the support of osd lock API. Since the lock is local lock, the lock will be released immediately once the resource usage is finished. If some remote user want to lock the locked resource, it will wait until the locked the resource is released. So the callback will not be needed for this lock, and we can export filesystem lock through lvfs to implement this lock. It will be discussed in OSD API HLD.

5.3 Metadata OSD API

Since MDD will implement namespace metadata management base on the OSD. so the OSD object API should be extended with some metadata elements.

5.3.1 OSD IAM API

These following OSD API will be used by MDD to lookup/insert/delete fid.

```
int osd_index_insert(...);
int osd_index_lookup(...);
int osd_index_delete(...);
```

These methods will be implemented in IAM API.

5.3.2 OSD object lookup and create

Except IAM API, the osd api should export following API to mdd.

```
int osd_object_create(obd, pobj, cobj, handle)
```

- This API will create a object whose fid is cfid in pfid.

```
int osd_object_getattr(obj, attr, name);
int osd_object_setattr(obj, attr, name, handle);
```

- These 2 API are used to getattr/setattr of the fid.

```
int osd_object_destroy(obd, obj, handle);
```

- This API is used for destroying the object fid pointed.

```
int osd_object_lock(obd, obj, mode);
int osd_object_unlock(obd, obj, mode);
```

- These 2 API are used to lock/unlock the object.

```
struct handle * osd_start_trans(obd, handle);
int osd_stop_trans(obd, handle);
```

- These 2 API are used to start/stop transaction.

The implementation of these API will be discussed in OSD API HLD.

6 State Management

6.1 mdd object status of fid

The object status is similar as inode. Each time, when we did sth on object, mdd_get_obj will be called first, then the object refcount will increase accordingly, after finish the operation, mdd_put_obj will be called to decrease the refcount of object as well, if the refcount is 0, the object will be freed. If the nlink is 0, the object will be destoryed here.

7 Alternatives

7.1 fid/object cache

In stead of defining the layering object in metadata stack, we can also implement MDT/MDD/OSD totally in terms fids, and implement a name/fid and fid/object cache in mdd and osd. But management the cache will be much more complicated than the ways we proposed. Because any modification in namespace or the osd should also be done for the cache, and furthermore the cache lock maybe also involved to keep cache invalidate and revalidate. Further more, with these kinds of layering object, some kinds of layering-specific stuff can be attached easily, so we choose this layering-object way.

8 Focus of Inspection

8.1 mdd and osd layer

Is it clear and right for osd and mdd layer separation? OSD exports only flat file and index API, and namespace manipulations are implemented on top of these APIs by mdd, right?

8.2 mdd locking

MDD/OSD lock is done by the exported lock of the filesystem through lvfs. Do we really need call_back for this lock?

9 Inspection Summary