MDD LOV DLD

Author:Wangdi

Date 2006/06/29

1 Functional Specification

In the new MDS stack, it will talk to LOV anyway. This DLD will describe how to make previous lov implementation work in current metadata stack and implement it with less modification.

There are include serval API MDD will use them to connect with LOV.

2 Use Cases

2.1 mdd setup

In mdd stack initialization, the name and uuid of lov(in config log) will be used by mdd to setup the lov obd.

- MDT gets config log from MGS, and metadata stack layer is initialized.
- MDD gets the config log (name/uuid of lov), and call its api to setup data stack in MDS.
- When new ost added, lov in mdd layer is notified and then it notifies MDD to sync the lov connection (reset llogs, update lov description, clear orphans).

2.2 create object

In mdd create API,

- It will call mdd lov api to create data object, (note: this should be done before the create transaction start, since no remote rpc is permitted inside transaction) and lov md info will be returned.
- Then it will create the metadata object.
- After that the lov md info will be set.

2.3 Unlink object

In mdd unlink API, since unlink rpc will be sent by llite, so for mdd layer, we only need care about the unlink log in case of lov obd, which will be discussed by other DLD.

2.4 mdd lov recovery

After MDS recovery, there will be post recovery methods, in this method, each layer's post recovery method will be called.

- MDD will set next id of each ost. (originally done by mds_lov_set_nextid).
- MDD will cleanup pending (which will be discussed in other DLD, unlink-orphan objects).
- MDD will sync the lov connection.

These usecase tests will be verified by normal sanity and replay tests.

3 Logic Specification

Currently lov is still an obd device, and it is still mainly accessed by obd export in those mds api. Since our focus is data stack in this cycle, so a new tmp MDS obd will be created in mdd layer to access lov, which will be put to the lu_obd of the lu_device in MDD layer. MDD will first access this obd, then get lov_export through this tmp MDS obd. So those mds lov sync api can be reused in the new layer.

3.1 MDD setup/cleanup

In MDD setup, MDD will get lov uuid/name from the config log. Then these info will be packed into the lcfg parameters and do amount of process to create that MDS OBD. In this process,

the lov device will be found by uuid/name and attach to this mds obd. After this mds obd is created, it will be put to the lu_obd of the lu_device in mdd layer.

```
int mdd_init_obd(const struct lu_context *ctxt, struct mdd_device *mdd,  char *dev)
{
        struct lustre_cfg_bufs bufs;
        struct lustre_cfg      *lcfg;
        struct obd_device      *obd;
        struct dt_object *obj_id;
        int rc;
        lustre_cfg_bufs_reset(&bufs, MDD_OBD_NAME);
```

```
            ...........
            /*Pack the  lov info here to info the new created mds*/
            rc = class_attach(lcfg);   /*attach the class*/
            if (rc)
                        GOTO(lcfg_cleanup, rc);
             obd = class_name2obd(MDD_OBD_NAME);
             if (!obd) {
                              CERROR("can not find obd %s \n", MDD_OBD_NAME);
                              LBUG();,
            }

            /*Some init process, open obj_id and so on*/
            .....................
            rc = class_setup(obd, lcfg);
            if (rc)
                   GOTO(class_detach, rc);
             /*add the new mds to the mdd layer*/
             mdd->mdd_md_dev.md_lu_dev.ld_obd = obd;
    }
```

For Cleanup, that tmp MDS obd should be cleanup correspondently.

```
          int mdd_cleanup_obd(struct mdd_device *mdd)
          {
                  struct lustre_cfg_bufs bufs;
                  struct lustre_cfg      *lcfg;
                  struct obd_device      *obd;
                  int rc;

                 obd = mdd->mdd_md_dev.md_lu_dev.ld_obd;


                 lustre_cfg_bufs_reset(&bufs, MDD_OBD_NAME);
                 lcfg = lustre_cfg_new(LCFG_ATTACH, &bufs);
                 if (!lcfg)
                           RETURN(-ENOMEM);
                 rc = class_cleanup(obd, lcfg);
                 if (rc)
                                 GOTO(lcfg_cleanup, rc);
                 rc = class_detach(obd, lcfg);
                 if (rc)
                                GOTO(lcfg_cleanup, rc);
                 mdd->mdd_md_dev.md_lu_dev.ld_obd = NULL;
    lcfg_cleanup:
                  lustre_cfg_free(lcfg);
                  RETURN(rc);
         }
```

3.2 Lov notify in MDD

When there is new ost added, lov will notify its observer to update the descripion of it. Since MDD layer will use mds obd to acess the lov, and vice versa. So lov will only need notify that tmp mds obd, and the original lov/mds notify mechanism will be untouched. But MDT layer also need to know the some stuff from lov, when ost add and delete. So we need another mechinasm to notify from the lower layer (MDD) to upper layer(MDT). Here, an new md method upcall will be added to handle this.

```
struct md_upcall {
                struct md_device *mu_upcall_dev;
                int (*mu_upcall)(const struct lu_context *ctxt, struct md_device *md, e
    };
    struct md_device {
                .................
                struct md_upcall md_upcall;
    };
    static int mdt_upcall(const struct lu_context *ctx, struct md_device *md, enum md_u
    {
             struct mdt_device *m = mdt_dev(&md->md_lu_dev);
             struct md_device  *next  = m->mdt_child;
             int rc = 0;
             switch (ev) {
             case MD_LOV_SYNC:
                    rc = next->md_ops->mdo_get_maxsize(ctx, next, &m->mdt_max_mdsize,
                                                                                &m->
                    break;
                    .......
             }
}
```

In this function, MDT will call mdo_get_maxsize to retrieve max_cookiesize and max_mdsize from the lower layer.

3.3 MDD LOV API

3.3.1 LOV create object

In MDD create object, we need create data objects on OST, and the logic is almost the same as the original mds_create_objects,

```
int mdd_lov_create(const struct lu_context *ctxt, struct mdd_device *mdd,
                                  struct mdd_object *parent, struct mdd_object *child,
                                  struct lov_mds_md **lmm, int *lmm_size,
                                  const struct md_create_spec *spec, struct lu_attr *la
```

```
{
                struct obd_device       *obd = mdd2obd_dev(mdd);
                struct obd_export       *lov_exp = obd->u.mds.mds_osc_exp;
                struct obdo             *oa;
                struct lov_stripe_md    *lsm = NULL;
                const void              *eadata = spec->u.sp_ea.eadata;
                __u32                    create_flags = spec->sp_cr_flags;
                int                      rc = 0;

                 if (create_flags & MDS_OPEN_DELAY_CREATE ||
                      !(create_flags & FMODE_WRITE))
                               RETURN(0);
                 oa = obdo_alloc();
                  .............prepare oa struct
                 if (!(create_flags & MDS_OPEN_HAS_OBJS)) {
                         if (create_flags & MDS_OPEN_HAS_EA) {
                                 rc = obd_iocontrol(OBD_IOC_LOV_SETSTRIPE, lov_exp, 0
                                 if (rc)
                                         GOTO(out_oa, rc);
                          } else {
                                          /* get lov ea from parent and set to lov */
                                          ......................
                           }
                            rc = obd_create(lov_exp, oa, &lsm, NULL);
                  } else {
                             LASSERT(eadata != NULL);
                             rc = obd_iocontrol(OBD_IOC_LOV_SETEA, lov_exp, 0, &lsm,
                             if (rc)
                                        GOTO(out_oa, rc);
                             lsm->lsm_object_id = oa->o_id;
                  }
                  .......................
                  /*pack lmm back to mdd*/
                  rc = obd_packmd(lov_exp, lmm, lsm);
                  if (rc < 0) {
                       CERROR("cannot pack lsm, err = %d\n", rc);
                       GOTO(out_oa, rc);
                  }
 }
```

3.3.2 LOV setattr

Sometimes, when setting attr, mdd need contact OST for updating these attr, for example uid/gid.

```
int mdd_lov_setattr_async(const struct lu_context *ctxt, struct mdd_object *obj, struct
{
           struct mdd_device *mdd = mdo2mdd(&obj->mod_obj);
           struct obd_device *obd = mdd2_obd(mdd);
           struct lu_attr *tmp_la = &mdd_ctx_info(ctxt)->mti_la;
           struct dt_object *next = mdd_object_child(obj);
           __u32 seq = lu_object_fid(mdd2lu_obj(obj))->f_seq;
           __u32 oid = lu_object_fid(mdd2lu_obj(obj))->f_oid;
           int rc = 0;
           ENTRY;
           rc = next->do_ops->do_attr_get(ctxt, next, tmp_la);
           if (rc)
                       RETURN(rc);
            /*set these attr to ost*/
            rc = mds_osc_setattr_async(obd, tmp_la->la_uid, tmp_la->la_gid, lmm, lmm_si
            RETURN(rc);
}
```

4 State Specification

4.1 Post recovery

For new mds stack recovery, it would be discussed in other DLD. For MDD layer , it should sync its lov info with remote ost server just like the previous MDS recovery implementaion. Since we still use that tmp MDS obd to access the lov, so all the recovery related mds lov method can be reused.