DLD for OPEN HANDLING in CMD

Huang Hua

Jul 5, 2006

Contents

1 Introduction

This DLD describes the details of open handling in CMD. Open recovery and other recovery will be described in other documents.

2 Functional Specification

2.1 Abstract

Files opened by clients will be tracked by MDS. This will give MDS better orphan handling and easy recovery. Every client has an export on MDS, and all opened files by this client will be linked into a list. So, if this client is evicted, all opened files by this client can be quickly closed.

2.2 Data structures

2.2.1 struct mdt_file_data

This data structure describes an opened file. All opened files on a MDS by one client is linked into a list by `mfd_list`.

```
/* file data for opened files on MDS */
struct mdt_file_data {
        struct portals_handle mfd_handle; /* must be first */
        struct list_head      mfd_list;   /* protected by med_open_lock */
        __u64                 mfd_xid;    /* the xid of the open request */
        int                   mfd_mode;   /* open mode from client*/
        struct mdt_object    *mfd_object; /* point to opened object */
};
```

2.2.2 struct mdt_export_data

This data structure describes an export on MDT corresponding to a client. All opened files by a client on this MDT is linked into `med_open_head`, protected by `med_open_lock`. Some other client data, such transaction number, last committed transaction number are stored in `struct mdt_client_data *med_mcd`. Every client has an index on this MDT, that is `med_lr_idx`.

```
struct mdt_export_data {
        /* list head of all opened file by this export. */
        struct list_head        med_open_head;
        /* lock med_open_head to to manipulate this list. */
        spinlock_t              med_open_lock;
        /* per client data to track transactions & request. */
        struct mdt_client_data *med_mcd;
        /* lock ibits known to this client. */
        loff_t                  med_lr_off;
        /* index allocated for this client by MDS */
        int                     med_lr_idx;
};
```

2.3 Interfaces

2.3.1 add reference

```
static void mdt_mfd_get(void *mfdp);
```

Parameters: `void *mfdp` - unused now.

Return Value: No return value.

Description: Just for callback. No refcount is kept now for mfd, because mfd and opened file has one-one relationship.

2.3.2 create a new mdt_file_data

```
static struct mdt_file_data *mdt_mfd_new(void);
```

Parameters: None.

Return Value: newly created mdt_file_data, to hold a newly opened file data.

Description: create a new mdt_file_data to hold the newly opened file, and insert this mfd into hast table. User should call `mdt_mfd_free()` to free it when file closed.

2.3.3 free a mdt_file_data

```
static void mdt_mfd_free(struct mdt_file_data *mfd);
```

Parameters: `struct mdt_file_data *mfd` - the mfd to free.

Return Value: No return value

Description: free the mfd.

2.3.4 find a mfd in the hash table

```
static struct mdt_file_data *
mdt_handle2mfd(const struct lustre_handle *handle);
```

Parameters: `const struct lustre_handle *handle` - the handle returned to client, and client gets back to MDT.

Return Value: valid mfd if found, NULL if not not found.

Description: find a mfd in the global hash table according to the handle provided by client. When this function returns a valid mfd, the reference count of this mfd has been increased by one as a callback. So `mdt_mfd_put()` should be called by the user.

2.3.5 link a mfd into the open list

```
void mdt_mfd_link(struct mdt_file_data *mfd,
                  struct mdt_export_data *med);
```

Parameters: `struct mdt_file_data * mfd` - the mfd to link into list;
`struct mdt_export_data *med - export data for this client.`

Return Value: None.

Description: link the mfd into med list; lock should be held by caller. Files opened by a client are all linked into its according med list.

2.3.6 delete a mfd from the open list

```
void mdt_mfd_unlink(struct mdt_file_data *mfd,
                    struct mdt_export_data *med);
```

Parameters: `struct mdt_file_data * mfd` - the mfd to link into list;

`struct mdt_export_data *med` - the export data;

Return Value: None.

Description: delete the mfd from med; lock should be held by caller.

2.3.7 helper for open a file

```
static int mdt_mfd_open(struct mdt_thread_info *info,
                        struct mdt_object *o,
                        int flags)
```

Parameters: `const struct lu_context *ctxt` - the execution context;

`struct mdt_object *o` - the object to open.

`int flags` - the open flag provided by client.

Return Value: zero indicates success. Other value means error.

Description: create mfd for this opened file, packing attr & xattr back to client; increase object reference count to keep it valid.

2.3.8 helper for close an opened file by mfd

```
int mdt_mfd_close(const struct lu_context *ctxt,
                  struct mdt_file_data *mfd);
```

Parameters: `const struct lu_context *ctxt` - the execution context;

`struct mdt_file_data * mfd` - the mfd to close.

Return Value: zero indicates success. Other value means error.

Description: close this opened file data @mfd. This is called upon client closing a file, also when disconnected from a client. This helper should handle orphan properly: if we are closing an orphan (which means that object has zero nlink), we should return lov ea to client. So client can destroy ost object.

2.3.9 MDS_OPEN request handler

```
int mdt_reint_open(struct mdt_thread_info *info);
```

Parameters: `struct mdt_thread_info *info` - mdt thread info.

Return Value: zero indicates success. Other value means error.

Description: This open is an reint or intent request.

Open file request from client. This maybe an reint request, or an intent request. File may be open by name, or only by fid. The file to open may exist on local MDT, or on remote MDT. Open file is to increase the reference count of an object.

2.3.10 MDS_CLOSE request handler

```
int mdt_close(struct mdt_thread_info *info);
```

Parameters: `struct mdt_thread_info *info` - mdt thread info.

Return Value: zero indicates success. Other value means error.

Description: close file request from client. MDT should decrease the reference count on the required object, and free the corresponding mdt_file_data. If the closed object has been unlinked, underlying layer will destroy it from disk after last `lu_object_put()` has release all reference count on it.

3 Use Cases

see the HLD for detailed information: ../../doc/HLD/cmd-open.lyx.

4 Logic Specification

4.1 MDS_OPEN request handler

```
int mdt_reint_open(struct mdt_thread_info *info)
{
        struct mdt_device      *mdt = info->mti_mdt;
        struct mdt_object      *parent;
        struct mdt_object      *child;
        struct mdt_lock_handle *lh;
        struct ldlm_reply      *ldlm_rep;
        struct ptlrpc_request  *req = mdt_info_req(info);
        struct lu_fid          *child_fid = &info->mti_tmp_fid1;
        int                     result;
        int                     created = 0;
        struct mdt_reint_record *rr = &info->mti_rr;
        ENTRY;
        if (strlen(rr->rr_name) == 0) {
                /* reint partial remote open */
                RETURN(mdt_open_by_fid(info, rr->rr_fid1,
                                       info->mti_attr.la_flags));
        }
        /* we now have no resent message, so it must be an intent */
        /*TODO: remove this and add MDS_CHECK_RESENT if resent enabled*/
        LASSERT(info->mti_pill.rc_fmt == &RQF_LDLM_INTENT_OPEN);
        ldlm_rep = req_capsule_server_get(&info->mti_pill, &RMF_DLM_REP);
        intent_set_disposition(ldlm_rep, DISP_LOOKUP_EXECD);
        lh = &info->mti_lh[MDT_LH_PARENT];
        lh->mlh_mode = LCK_PW;
        parent = mdt_object_find_lock(info, rr->rr_fid1, lh,
                                      MDS_INODELOCK_UPDATE);
        if (IS_ERR(parent)) {
                /* just simulate child not existing */
                intent_set_disposition(ldlm_rep, DISP_LOOKUP_NEG);
                GOTO(out, result = PTR_ERR(parent));
        }
        result = mdo_lookup(info->mti_ctxt, mdt_object_child(parent),
                            rr->rr_name, child_fid);
        if (result != 0 && result != -ENOENT) {
                GOTO(out_parent, result);
        }
        if (result == -ENOENT) {
                intent_set_disposition(ldlm_rep, DISP_LOOKUP_NEG);
                if (!(info->mti_attr.la_flags & MDS_OPEN_CREAT))
                        GOTO(out_parent, result);
                if (req->rq_export->exp_connect_flags & OBD_CONNECT_RDONLY)
```

```
                        GOTO(out_parent, result = -EROFS);
                *child_fid = *info->mti_rr.rr_fid2;
        } else {
                intent_set_disposition(ldlm_rep, DISP_LOOKUP_POS);
                if (info->mti_attr.la_flags & MDS_OPEN_EXCL &&
                    info->mti_attr.la_flags & MDS_OPEN_CREAT)
                        GOTO(out_parent, result = -EEXIST);
                /* child_fid is filled by mdo_lookup(). */
                LASSERT(lu_fid_eq(child_fid, info->mti_rr.rr_fid2));
        }
        child = mdt_object_find(info->mti_ctxt, mdt, child_fid);
        if (IS_ERR(child))
                GOTO(out_parent, result = PTR_ERR(child));
        if (result == -ENOENT) {
                /* not found and with MDS_OPEN_CREAT: let's create it */
                result = mdo_create(info->mti_ctxt,
                                    mdt_object_child(parent),
                                    rr->rr_name,
                                    mdt_object_child(child),
                                    rr->rr_tgt,
                                    &info->mti_attr);
                intent_set_disposition(ldlm_rep, DISP_OPEN_CREATE);
                if (result != 0)
                        GOTO(out_child, result);
                created = 1;
        }
        /* Open it now. */
        result = mdt_mfd_open(info, child, info->mti_attr.la_flags);
        intent_set_disposition(ldlm_rep, DISP_OPEN_OPEN);
        GOTO(finish_open, result);
finish_open:
        if (result != 0 && result != -EREMOTE && created) {
                mdo_unlink(info->mti_ctxt, mdt_object_child(parent),
                           mdt_object_child(child), rr->rr_name);
        }
out_child:
        mdt_object_put(info->mti_ctxt, child);
out_parent:
        mdt_object_unlock_put(info, parent, lh);
out:
        return result;
}
```

4.2 MDS_CLOSE request handler

```
int mdt_close(struct mdt_thread_info *info)
{
        struct mdt_export_data *med;
        struct mdt_file_data   *mfd;
        int rc;
        ENTRY;
        med = &mdt_info_req(info)->rq_export->exp_mdt_data;
        spin_lock(&med->med_open_lock);
        mfd = mdt_handle2mfd(&(info->mti_body->handle));
        if (mfd == NULL) {
                spin_unlock(&med->med_open_lock);
                CDEBUG(D_INODE, "no handle for file close: fid = "DFID3
                       ": cookie = "LPX64, PFID3(&info->mti_body->fid1),
                       info->mti_body->handle.cookie);
                rc = -ESTALE;
        } else {
                class_handle_unhash(&mfd->mfd_handle);
                list_del_init(&mfd->mfd_list);
                spin_unlock(&med->med_open_lock);

                rc = mdt_handle_last_unlink(info, mfd->mfd_object,
                                            &RQF_MDS_CLOSE_LAST);
                rc = mdt_mfd_close(info->mti_ctxt, mfd);
        }
        RETURN(rc);
}
```

4.3 mdt_open_by_fid()

This is a partial reint open request, issued by client for opening an object located on different MDS from its parent .

```
int mdt_open_by_fid(struct mdt_thread_info* info, const struct lu_fid *fid,
                    __u32 flags)
{
        struct mdt_object *o;
        int rc;
        ENTRY;
        o = mdt_object_find(info->mti_ctxt, info->mti_mdt, fid);
        if (!IS_ERR(o)) {
                if (mdt_object_exists(info->mti_ctxt, &o->mot_obj.mo_lu)) {
                        rc = mdt_mfd_open(info, o, flags);
                } else {
```

```
                        rc = -ENOENT;
                }
                mdt_object_put(info->mti_ctxt, o);
        } else
                rc = PTR_ERR(o);
        RETURN(rc);
}
```

4.4 mdt_mfd_open() helper

```
static int mdt_mfd_open(struct mdt_thread_info *info,
                        struct mdt_object *o,
                        int flags)
{
        struct mdt_export_data *med;
        struct mdt_file_data   *mfd;
        struct mdt_body        *repbody;
        struct lov_mds_md      *lmm = NULL;
        struct lu_attr         *attr = &info->mti_attr;
        struct ptlrpc_request  *req = mdt_info_req(info);
        int                     rc = 0;
        ENTRY;
        med = &req->rq_export->exp_mdt_data;
        repbody = req_capsule_server_get(&info->mti_pill, &RMF_MDT_BODY);
        if (req_capsule_has_field(&info->mti_pill, &RMF_MDT_MD))
                lmm = req_capsule_server_get(&info->mti_pill, &RMF_MDT_MD);
        rc = mo_attr_get(info->mti_ctxt, mdt_object_child(o), attr);
        if (rc == 0) {
                if (!S_ISREG(attr->la_mode) &&
                    !S_ISDIR(attr->la_mode) &&
                     (req->rq_export->exp_connect_flags & OBD_CONNECT_NODEVO
                        /* If client supports this, do not return open handl
                        *  for special device nodes */
                        RETURN(0);
                /* FIXME:maybe this can be done earlier? */
                if (S_ISDIR(attr->la_mode)) {
                        if (flags & (MDS_OPEN_CREAT | FMODE_WRITE)) {
                                /* we are trying to create or
                                 * write an existing dir. */
                                rc = -EISDIR;
                        }
                } else if (flags & MDS_OPEN_DIRECTORY)
                        rc = -ENOTDIR;
        }
```

```
        if (rc != 0) {
                if (rc == -EREMOTE) {
                        repbody->fid1 = *mdt_object_fid(o);
                        repbody->valid |= OBD_MD_FLID;
                }
                RETURN(rc);
        }
        mdt_pack_attr2body(repbody, attr, mdt_object_fid(o));
        if (lmm) {
                rc = mo_xattr_get(info->mti_ctxt, mdt_object_child(o),
                                  lmm, info->mti_mdt->mdt_max_mdsize,
                                  XATTR_NAME_LOV);
                if (rc < 0)
                        RETURN(-EINVAL);
                if (S_ISDIR(attr->la_mode))
                        repbody->valid |= OBD_MD_FLDIREA;
                else
                        repbody->valid |= OBD_MD_FLEASIZE;
                repbody->eadatasize = rc;
                rc = 0;
        }
        if (flags & FMODE_WRITE) {
                /*mds_get_write_access*/
        } else if (flags & MDS_FMODE_EXEC) {
                /*mds_deny_write_access*/
        }
        mfd = mdt_mfd_new();
        if (mfd != NULL) {
                /* keep a reference on this object for this open,
                * and is released by mdt_mfd_close() */
                mdt_object_get(info->mti_ctxt, o);
                mfd->mfd_mode = flags;
                mfd->mfd_object = o;
                mfd->mfd_xid = mdt_info_req(info)->rq_xid;
                spin_lock(&med->med_open_lock);
                list_add(&mfd->mfd_list, &med->med_open_head);
                spin_unlock(&med->med_open_lock);
                repbody->handle.cookie = mfd->mfd_handle.h_cookie;
        } else
                rc = -ENOMEM;
        RETURN(rc);
}
```

4.5 mdt_mfd_close() helper

```
int mdt_mfd_close(const struct lu_context *ctxt,
                  struct mdt_file_data *mfd)
{
        ENTRY;
        if (mfd->mfd_mode & FMODE_WRITE) {
                /*mdt_put_write_access*/
        } else if (mfd->mfd_mode & MDS_FMODE_EXEC) {
                /*mdt_allow_write_access*/
        }
        /* release reference on this object.
         * it will be destroyed by lower layer if necessary.
         */
        mdt_object_put(ctxt, mfd->mfd_object);
        mdt_mfd_free(mfd);
        RETURN(0);
}
```

4.6 mdt_mfd_new()

```
/* Create a new mdt_file_data struct, initialize it,
 * and insert it to global hash table */
static struct mdt_file_data *mdt_mfd_new(void)
{
        struct mdt_file_data *mfd;
        ENTRY;
        OBD_ALLOC_PTR(mfd);
        if (mfd != NULL) {
                INIT_LIST_HEAD(&mfd->mfd_handle.h_link);
                INIT_LIST_HEAD(&mfd->mfd_list);
                class_handle_hash(&mfd->mfd_handle, mdt_mfd_get);
        } else
                CERROR("mdt: out of memory\n");
        RETURN(mfd);
}
```

4.7 mdt_handle2mfd()

```
/* Find the mfd pointed to by handle in global hash table. */
static struct mdt_file_data *mdt_handle2mfd(const struct lustre_handle *hand
{
        ENTRY;
```

```
        LASSERT(handle != NULL);
        RETURN(class_handle2object(handle->cookie));
}
```

5 State Specification

5.1 State invariant

Because MDT mostly acts as an request receiver and sender, so some work should be done by other layer to serve open request:

1. Object distribution policy should be performed by CMM;
2. Remote object creation should be initiated by CMM. This will be received by other MDT, then be handled properly;
3. Partial open request or resent request handling will be recognized by MDT;
4. OSD should perform necessary permission checking.

6 Focus for Inspection

1. Return value for remote object? The same question exists in other places. What value should be return to client when MDT gets -EREMOTE from CMM? Client should be able to handle this situation properly!