SPLIT DIR

Author: Wang Di

Date 2006/09/08

1 Functional Specification

According to HLD, split will be handled in these three layers. CMM will control the split process, check split possiablity and split the dir. MDC will transfer the split dir entries to slave MDS. MDT will recevive the split entries and write them to the slave objects.

int cml_try_to_split(const struct lu_context *ctx, struct md_object *mo);

Args:

ctx: the contex of this operation.

mo: the md object will be split.

Return:

0 indicate success, otherwise failed.

Description:

It will check whether the dir should be split, if it is, split this dir.

int mdc_send_page(const struct lu_context *ctx, struct md_object *mo, struct page *page, __u32 end);

Args:

ctx: the context of this operation.

mo: the md object being split.

page: the page include split dir entries.

end: the hash segment value, which means the end of this splitting hash segment.

Return:

0 indicate success, -E2BIG means we have reach the end of the split segment, other value means error.

Description:

It will be used to send split dir entries to slave MDS.

static int mdt_writepage(struct mdt_thread_info *info)

Args:

info: the thread_info of this API.

Return:

0 indicate success, otherwise failed.

Description:

This API will retrieve the dir entries from the received page and write them to the slave object.

2 Use Cases

2.1 Split dir

If there are too much entries (dir_size > 64K) for a dir, the dir will be split. (Note: the root will not be split)

- Touch many entries in one dir.
- The dir should be split, when its size > 64K. (cmm layer will call split API to split the dir)
- It can be checked by lfs getstripe, and those slave objects can be shown by this.

2.2 Access split dir

When accessing split dir in client, LMV should help to find the right MDS.

- Accessing a split dir. (LMV layer should recognize split dir and give the right MDS for the request)
- Lookup/Readdir should works fine for a split dir with the help of LMV layer. (Note: this is already implemented by LMV layer now, so it will not be discussed detailly in this DLD)

3 Logic Specification

3.1 CMM Split dir

When creating an object in a dir, CMM layer will check whether this dir will be split. There are serval cases, the dir should be split

- The Lustre is setup with multiple MDSs, not single one.
- The dir size is bigger than 64K.
- The dir was not be splitted.
- The dir is not root object.

```
static int cmm_expect_splitting(const struct lu_context *ctx, struct md_object *mo, str
{
                struct cmm_device *cmm = cmm_obj2dev(md2cmm_obj(mo));
                struct lu_fid *fid = NULL;
                int rc = CMM_EXPECT_SPLIT;
                ENTRY;
```

```
                if (cmm->cmm_tgt_count == 0)
                            GOTO(cleanup, rc = CMM_NO_SPLIT_EXPECTED);
                if (ma->ma_attr.la_size < SPLIT_SIZE)
                             GOTO(cleanup, rc = CMM_NO_SPLIT_EXPECTED);
                if (ma->ma_lmv_size)
                            GOTO(cleanup, rc = CMM_NO_SPLIT_EXPECTED);
                OBD_ALLOC_PTR(fid);
                rc = cmm_root_get(ctx, &cmm->cmm_md_dev, fid);
                if (rc)
                            GOTO(cleanup, rc);
                if (lu_fid_eq(fid, cmm2_fid(md2cmm_obj(mo))))
                            GOTO(cleanup, rc = CMM_NO_SPLIT_EXPECTED);
cleanup:
                if (fid)
                           OBD_FREE_PTR(fid);
                RETURN(rc);
}
```

When spliting dir, it will first create the slave objects in slave MDS, then scan the split dir and retrieve the split entries and distribute those entries to the slave objects.

```
int cml_try_to_split(const struct lu_context *ctx, struct md_object *mo)
{
          struct md_attr *ma;
          int rc = 0;
          ENTRY;

          LASSERT(S_ISDIR(lu_object_attr(&mo->mo_lu)));
          OBD_ALLOC_PTR(ma);
          if (ma == NULL)
                       RETURN(-ENOMEM);
          ma->ma_need = MA_INODE;
          rc = mo_attr_get(ctx, mo, ma);
          if (rc)
                  GOTO(cleanup, ma);
          /* step1: checking whether the dir need to be splitted */
          rc = cmm_expect_splitting(ctx, mo, ma);
          if (rc != CMM_EXPECT_SPLIT)
                       GOTO(cleanup, rc = 0);
          /* step2: create slave objects */
          rc = cmm_create_slave_objects(ctx, mo, ma);
          if (rc)
                      GOTO(cleanup, ma);
          /* step3: scan and split the object */
```

```
          rc = cmm_scan_and_split(ctx, mo, ma);
cleanup:
          if (ma->ma_lmv_size && ma->ma_lmv)
                    OBD_FREE(ma->ma_lmv, ma->ma_lmv_size);
         OBD_FREE_PTR(ma);
         RETURN(rc);
}
```

When creating slave objects, the new fid should be allocated because the objects will be transfered to other MDS. After those slave objects being created, the MEA(split dir EA info) will be set to the master dir object.

```
static int cmm_create_slave_objects(const struct lu_context *ctx,  struct md_object *mo
{
       struct cmm_device *cmm = cmm_obj2dev(md2cmm_obj(mo));
       struct lmv_stripe_md *lmv = NULL;
       int lmv_size, i, rc;
       struct lu_fid *lf = cmm2_fid(md2cmm_obj(mo));
       ENTRY;

      lmv_size = cmm_md_size(cmm->cmm_tgt_count + 1);
        /* This lmv will be free after finish splitting. */
     OBD_ALLOC(lmv, lmv_size);
     if (!lmv)
              RETURN(-ENOMEM);
     lmv->mea_master = -1;
     lmv->mea_magic = MEA_MAGIC_ALL_CHARS;
     lmv->mea_count = cmm->cmm_tgt_count;
     lmv->mea_ids[0] = *lf;
     rc = cmm_alloc_fid(ctx, cmm, &lmv->mea_ids[1], cmm->cmm_tgt_count);
     if (rc)
               GOTO(cleanup, rc);
     for (i = 1; i < cmm->cmm_tgt_count; i ++) {
              rc = cmm_creat_remote_obj(ctx, cmm, &lmv->mea_ids[i], ma);
              if (rc)
                       GOTO(cleanup, rc);
      }
      rc = mo_xattr_set(ctx, md_object_next(mo), lmv, lmv_size,  MDS_LMV_MD_NAME, 0);
      ma->ma_lmv_size = lmv_size;
      ma->ma_lmv = lmv;
cleanup:
      RETURN(rc);
}
```

When splitting dir, we will first compute the split segments for each MDS, then get entries according these segments and send them to other slave MDSs.

```
#define MAX_HASH_SIZE 0x3fffffff
#define SPLIT_PAGE_COUNT 1
static int cmm_scan_and_split(const struct lu_context *ctx,  struct md_object *mo, stru
{
        struct cmm_device *cmm = cmm_obj2dev(md2cmm_obj(mo));
        __u32 hash_segement;
        struct lu_rdpg   *rdpg = NULL;
        int rc = 0, i;

       OBD_ALLOC_PTR(rdpg);
       if (!rdpg)
                RETURN(-ENOMEM);
       rdpg->rp_npages = SPLIT_PAGE_COUNT;
       rdpg->rp_count  = CFS_PAGE_SIZE * rdpg->rp_npages;
       OBD_ALLOC(rdpg->rp_pages, rdpg->rp_npages * sizeof rdpg->rp_pages[0]);
       if (rdpg->rp_pages == NULL)
               GOTO(free_rdpg, rc = -ENOMEM);
       for (i = 0; i < rdpg->rp_npages; i++) {
                rdpg->rp_pages[i] = alloc_pages(GFP_KERNEL, 0);
                if (rdpg->rp_pages[i] == NULL)
                       GOTO(cleanup, rc = -ENOMEM);
      }
       hash_segement = MAX_HASH_SIZE / cmm->cmm_tgt_count;
       for (i = 1; i < cmm->cmm_tgt_count; i++) {
               struct lu_fid *lf = &ma->ma_lmv->mea_ids[i];
                __u32 hash_end;
                rdpg->rp_hash = i * hash_segement;
                hash_end = rdpg->rp_hash + hash_segement;
                rc = cmm_split_entries(ctx, mo, rdpg, lf, hash_end);   /* Here it will


                if (rc)
                        GOTO(cleanup, rc);
        }
cleanup:
        for (i = 0; i < rdpg->rp_npages; i++)
               if (rdpg->rp_pages[i] != NULL)
                      __free_pages(rdpg->rp_pages[i], 0);
      if (rdpg->rp_pages)
             OBD_FREE(rdpg->rp_pages, rdpg->rp_npages *  sizeof rdpg->rp_pages[0]);
free_rdpg:
      if (rdpg)
            OBD_FREE_PTR(rdpg);
      RETURN(rc);
}
```

3.2 Send dir entries to other MDSs

When sending split entries to slave MDSs, it will send one page each time with bulk RPC interface.

```
int mdc_send_page(const struct lu_context *ctx, struct md_object *mo,  struct page *pag
{
        struct mdc_device *mc = md2mdc_dev(md_obj2dev(mo));
        struct lu_dirpage *dp;
        struct lu_dirent  *ent;
        int rc, offset = 0, rc1 = 0;
        ENTRY;
        kmap(page);
        dp = page_address(page);
        for (ent = lu_dirent_start(dp); ent != NULL;  ent = lu_dirent_next(ent)) {
                if (ent->lde_hash < end) {
                     offset = (int)((__u32)ent - (__u32)dp);
                      rc1 = -E2BIG;
                      goto send_page;
                }
                 /* allocate new fid for each obj */
               rc = obd_fid_alloc(mc->mc_desc.cl_exp, &ent->lde_fid, NULL);
               if (rc) {
                        kunmap(page);
                       RETURN(rc);
                }
      }
       kunmap(page);
       offset = CFS_PAGE_SIZE;
send_page:
      if (offset > 0) {
                rc = mdc_sendpage(mc->mc_desc.cl_exp, lu_object_fid(&mo->mo_lu), page,
                CDEBUG(D_INFO, "send page %p  offset %d fid "DFID" rc %d \n",
                                  page, offset, PFID(lu_object_fid(&mo->mo_lu)), rc);
      }
       if (rc == 0)
                rc = rc1;
      RETURN(rc);
}
```

When MDS receive the splitting pages, it will retrieve the split entries and insert them to the slave objects.

```
static int mdt_write_dir_page(struct mdt_thread_info *info, struct page *page)
{
```

```
        ..................................
        for (ent = lu_dirent_start(dp); ent != NULL;   ent = lu_dirent_next(ent)) {
                struct lu_fid *lf = &ent->lde_fid;
                ..................... insert the name to the slave objects;
                rc = mdo_name_insert(info->mti_ctxt,  md_object_next(&object->mot_obj),
                if (rc) {
                        kunmap(page);
                        RETURN(rc);
                }
         }
         kunmap(page);
         RETURN(rc);
}
static int mdt_writepage (struct mdt_thread_info *info)
{
         ........................
         desc = ptlrpc_prep_bulk_exp (req, 1, BULK_GET_SINK, MDS_BULK_PORTAL);
          if (desc)
                RETURN(-ENOMEM);
         /* allocate the page for the desc */
        page = alloc_pages(GFP_KERNEL, 0);
        if (!page)
                GOTO(desc_cleanup, rc = -ENOMEM);
         ptlrpc_prep_bulk_page(desc, page, 0, CFS_PAGE_SIZE);

          OBD_ALLOC_PTR(lwi);
          if (!lwi)
                  GOTO(cleanup_page, rc = -ENOMEM);
          rc = ptlrpc_start_bulk_transfer (desc);
       if (rc == 0) {
          .....................waiting for page transfer finished
       }
       rc = mdt_write_dir_page(info, page);
cleanup_lwi:
       OBD_FREE_PTR(lwi);
cleanup_page:
       __free_pages(page, 0);
desc_cleanup:
         ptlrpc_free_bulk(desc);
         RETURN(rc);
}
```

3.3 readdir for split entries

For a split dir, when one of its hash segment is finished, the end offset of this hash segment should be reset for switch to the next segment to read. Here we

retrieve the start offset of next hash segment as the the end offset of this one. This was handled in LMV layer.

```
static int lmv_readpage ( ....)
{
.................
          rc = md_readpage(tgt_exp, &rid, offset, page, request);
          if (rc)
                    GOTO(cleanup, rc);
         if (obj && i < obj->lo_objcount - 1) {
                     struct lu_dirpage *dp;
                     __u32 end;

                      kmap(page);
                     dp = cfs_page_address(page);
                     end = le32_to_cpu(dp->ldp_hash_end);
                     if (end == ~0ul)
                     rc = lmv_reset_hash_seg_end(lmv, obj, fid, i + 1, dp);
                     kunmap(page);
          }
           ........................
}
static int lmv_reset_hash_seg_end (struct lmv_obd *lmv, struct lmv_obj *obj,  const str
                                                        struct lu_dirpage *dp)
{
          struct ptlrpc_request *tmp_req = NULL;
          struct page *page = NULL;
          struct lu_dirpage *next_dp;
          struct obd_export *tgt_exp;
          struct lu_fid rid = *fid;
          __u32 seg_end, max_hash = MAX_HASH_SIZE;
          int rc = 0;
           do_div(max_hash, obj->lo_objcount);
           seg_end = max_hash * index;
           /* Get start offset from next segment */
           rid = obj->lo_inodes[index].li_fid;
           tgt_exp = lmv_get_export(lmv, &rid);
           if (IS_ERR(tgt_exp))
                   GOTO(cleanup, PTR_ERR(tgt_exp));
           /* Alloc a page to get next segment hash,
            * FIXME: should we try to page from cache first */
            page = alloc_pages(GFP_KERNEL, 0);
            if (!page)
                    GOTO(cleanup, rc = -ENOMEM);
            rc = md_readpage(tgt_exp, &rid, seg_end, page, &tmp_req);
          if (rc) {
```

```
                        /* E2BIG means it already reached the end of the dir,
                         * no need reset the hash segment end */
                          if (rc == -E2BIG)
                                  GOTO(cleanup, rc = 0);
                        if (rc != -ERANGE)
                           GOTO(cleanup, rc);
                     if (rc == -ERANGE)
                           rc = 0;
                }
                 kmap(page);
                 next_dp = cfs_page_address(page);
                LASSERT(le32_to_cpu(next_dp->ldp_hash_start) >= seg_end);
                dp->ldp_hash_end = next_dp->ldp_hash_start;
                kunmap(page);
                 .........
        }
```

4 State Specification

Since split is only for verification testes, and there will be no split when do recovery, so recovery will not be considered in the split.

4.1 split lock

Split will only happen in creating objects in CMM layer. Since in create process the parent object, which might be splitted, is already locked by EX lock, so in the whole split process, the split object will be locked by this lock.

```
static int mdt_md_create(struct mdt_thread_info *info)
{
        ......................
        parent = mdt_object_find_lock(info, rr->rr_fid1, lh, MDS_INODELOCK_UPDATE);
        ......................
        /* Call CMM layer create API, where the parent might be split */
 }
```