UUID Hash DLD for lustre-b1_5(Bug11013)

Yu Zhang Yong

2007-01-22

1 Requirements

1. The exports are held in the linked list and there are a lot of clients connected, it takes long time to traverse the list for unique UUIDs & eviction requests.

2 Summary of the solution

2.1 New solution - use the hash table to instead of lists.

1. uuid hash management chart (e.g. uuid_hash).

3 Functional Specification

3.1 Data struct

3.1.1 hash class data struct describe :

1. define hash bucket struct.

```
struct lustre_hash_bucket { /* define the hash bucket*/
      struct hlist_head lhb_list;
      spinlock_t lhb_lock;
#ifdef LUSTRE_HASH_DEBUG
      int lhb_item_count; /* the number of hash item per bucket, it will help us to ana
#endif

}
```

3. define a common hash body struct:

```
/* hash operations method define */
struct lustre_hash_operations {
    __s32 (*lustre_hashfn) (struct lustre_hash_body *hash_body, void *key);
   int    (*lustre_hash_key_compare) (void *key, struct hlist_node *actual_hnode, st
```

```
      void * (*lustre_hash_object_refcount_get) (struct hlist_node *hash_item) /* add t
      void   (*lustre_hash_object_refcount_put) (struct hlist_node *hash_item) /* del t
};
   struct lustre_class_hash_body {
       char hashname[10];
       spinlock_t lchb_lock; /* body lock */
       struct lustre_hash_bucket * lchb_hash_tables;

       __s32 lchb_hash_max_size; /* define the hash tables size */
       struct lustre_hash_operations * lchb_hash_operations;  /* define the hash operat
};

/* define the uuid hash operations */
struct lustre_hash_operations uuid_hash_operations {
        .lustre_hashfn = uuid_hashfn;
        .lustre_hash_key_compare = uuid_hash_key_compare;
        .lustre_hash_object_refcount_get = uuid_hash_object_refcount_get;
        .lustre_hash_object_refcount_put = uuid_hash_object_refcount_put;
}
uuid hash data struct
5. actual hash object : (e.g. export)
   struct obd_export {
         ...
         struct obd_uuid exp_client_uuid;
+        struct hlist_node   exp_uuid_hashitem;
+        struct hlist_node   exp_nid_hashitem;
         struct list_head exp_obd_chain;
         ...
   }
```

6. for uuid-hash, need to be specific for each obd device

```
   struct obd_device {
         ...
         struct obd_uuid obd_uuid;
+        struct lustre_class_hash_body * uuid_export_hash;  /* it point the actual uuid
         int    obd_minor;
         ...
   }
```

3.2 new function:

3.2.1 Hash class Common function define :

1. int lustre_hash_init(struct lustre_class_hash_body *hash_body,char * hashname, __s32 hashsize,
struct lustre_hash_operations * hash_operations)

Parameter :

struct lustre_class_hash_body * lustre_hash_body : point to the actual hash body (e.g. uuid_hash, nid_hash)

char * hashname : hashname

__s32 hashsize : hashtables size

struct lustre_hash_operations *hash_operations : hash body operations method.

Return value :

return 0 if success, else return error code.

Description :

this function : alloc space for the hash_body and initialize the hash body with the given parameter.

2. void lustre_hash_exit(struct lustre_class_hash_body * hash_body)

Parameter :

struct lustre_class_hash_body * hash_body : point to the actual hash body;

Return value :

N/A

Description :

this function delete all hash item and delete the hash tables.

3. static struct hlist_node * lustre_hash_getitem_in_bucket_nolock(struct lustre_class_hash_body *hash_body, int hashent, void *key)

Parameter :

struct lustre_class_hash_body *hash_body : hash body;

int hashent : index in hash bucket.

void * key : search item with hash key;

Return value :

if find return the hash_node, else return NULL;

Description:

search item with given key in given hash bucket;

4. int lustre_hash_additem (struct lustre_class_hash_body *hash_body, void *key, struct hlist_node *actual_hnode)

Parameter :

struct lustre_class_hash_body *hash_body: point the actual hash body;

void * key : key value;

struct hlist_node *actual_hnode: actual object's hash_node pointer, example: export->exp_uuid_hash;

Return value :

return 0 if success, else return error code.

Description :

add a hash_item data to related hashtable.

5. int lustre_hash_delitem_nolock(struct lustre_hash_body *hash_body, int hashent, struct hlist_node * hash_item)

Parameter :

struct lustre_hash_body *hash_body :

int hashent :

struct hlist_node *hash_item :
Return Value :
return 0 if success, else return error code.
Description :
delete a hash_item of no lock verstion.
5. int lustre_hash_delitem_by_key(struct lustre_class_hash_body *hash_body, void *key)
Paramter :
struct lustre_class_hash_body *hash_body : hash body
void *key : hash item key
Return Value :
return 0 if success, else return error code.
Description :
delete a hash item with given key.
5. int lustre_hash_delitem(struct lustre_clas_hash_body *hash_body, void *key, struct hlist_node * hash_item)
Parameter :
struct lustre_class_hash_body * lustre_hash_body: the actual hash body (e.g. uuid_hash, nid_hash);
void *key : hash key value.
struct hlist_node *actual_hnode : the item will be deleted from the hash tables.
Return value :
return 0 if success, else return error code.
Description :
delete a hash_item data from given hashtables.
6. void * lustre_hash_get_object_by_key(struct lustre_class_hash_body *hash_body, void *key)
Parameter :
struct lustre_class_hash_body * hash_body: actual hash body (e.g. uuid_hash, nid_hash);
void *key : key of actual object, need to get the actual object with the given key.
Return value :
return related actual object address with the given key if success, else return NULL.
Description :
Get related actual object value with the given key.

3.2.2 lustre hash body function pointer descripte (these function is defined in lustre_class_hash_body struct):

7. __s32 uuid_hashfn(struct lustre_hash_body *hash_body, void * key)
Parameter :
void * key : hash key.
struct lustre_hash_body *hash_body : hash body.

Return value :
hash value if success, else return -1 when fails;
Description:
calculate the hash value with given hash key.
8. int uuid_hash_key_compare(void * key, struct hlist_node * compared_hnode);
Parameter :
void *key : given source hash key.
struct hlist_node *compared_hnode : will be compared hash item in hash bucket.
Return value:
if their hash keys are match, return 1. else return 0;
Description:
actual object's hash compare function.
9. void * uuid_hash_object_refcount_get(struct hlist_node *item_hnode)
Paremeter :
struct hlist_node *item_hnode * actual object's hash pointer feild.
Return Value :
actual object's address pointer, e.g. export.
Description :
add the object reference_count.
10. void uuid_hash_object_refcount_put(struct hlist_node *item_hnode)
Paremeter :
struct hlist_node *item_hnode * actual object's hash pointer feild.
Return Value :
N/A
Description :
dec the object reference_count.

4 Use Cases

4.1 uuid hash use

- when obd_device initialize, we have to create an "UUID_HASH" uuid_hash_body for this obd_device
- when create a new export, We need to add the hash_item information to uuid_hash_body.
- when delete (an | all) export, We have to delete (an | all) hash_item information from the uuid_hash_body.
- if known an uuid , We can find it's export from the UUID in the uuid_hash_body.

4.2 unit test

- run lustre system, and connect multi client.
- print the hash_body create system message / hash_item initailize system message / hash_item add system message / hash_item delete system message / hash_item exit system message.
- add an exist hash_item / delete an non-exist hash_item.

5 Logic Specificationtips

5.1 hash class function: (hash init, exit, additem, delitem, delitem_directly, get_object_by_key)

```
1. int lustre_hash_init(struct lustre_class_hash_body *hash_body,char * hashname, __s32
                        struct lustre_hash_operations * hash_operations)
{


    LASSERT(hash_body == NULL);
    LASSERT(hashsize > 0);
    LASSERT(hash_operations != NULL);


    OBD_ALLOC(hash_body, sizeof(*hash_body));  /* alloc space for hash_body */
    if (hash_body == NULL)
        GOTO(err_exp, err);
   strcpy(hash_body->lchb_hashname, hashname);
   hash_body->lchb_hash_max_size = hashsize;
   hash_body->lchb_hash_operations = hash_operations;

   /* alloc space for the hash tables */
   OBD_ALLOC(hash_body->lchb_hash_tables, sizeof(*hash_body->lchb_hash_tables) * hash_b

   If (hash_body->lchb_hash_tables == NULL)
   {
       print error message;
       return -ENOMEM;
   }

   spin_lock_init(hash_body->lchb_lock); /* initialize the body lock */

   for(i =0 ; i < hash_body->lchb_hash_max_size; i++)
   {
```

```
        /* initial the bucket lock and list_head */

        INIT_HLIST_HEAD( hash_body->lchb_hash_tables[i].lhb_list);
        spin_lock_init(hash_body->lchb_hash_tables[i].lhb_lock);
    }

     return 0;
}

2. void lustre_hash_exit(struct lustre_class_hash_body * hash_body)
{

       spin_lock(hash_body->lchb_lock); /* lock the hash tables */

       if (hash_body->lchb_hash_tables == NULL ){
           spin_unlock(hash_body->lchb_lock);
           return;
       }

       for( i = 0; i < hash_body->lchb_hash_max_size; i++ )
       {
         struct lustre_hash_bucket * bucket;
         struct hlist_node * actual_hnode, *pos;
         bucket = hash_body->lchb_hash_tables[i];

         spin_lock(hash_body->lchb_hash_tables[hashent]->lhb_lock); /* lock the bucket *
         list_for_each_safe_rcu(actual_hnode, pos, &bucket->lhb_list)) {

             lustre_hash_delitem_nolock(hash_body, hashent, actual_hnode);

         }
         spin_unlock(hash_body->lchb_hash_tables[hashent]->lhb_lock);
       }

       OBD_FREE(hash_body->hash_tables_tables); /* free the hash_tables's memory space *

       hash_body->hash_tables_tables = NULL;
       spin_unlock(hash_body->lchb_lock);

       OBD_FREE(hash_body); /* free the hash_body's memory space */
       return;
}

3. static struct hlist_node * lustre_hash_getitem_in_bucket_nolock(struct lustre_class_
{
     struct lustre_hash_bucket * bucket = hash_body->lchb_hash_tables[hashent];
```

```
     struct hlist_node  * hash_item_node, *pos ;
     int find = 0;

     list_for_each_safe_rcu(hash_item_node, pos, &bucket->lhb_list) {

       find = hash_body->lustre_hash_key_compare(key, hash_item_node);
       if (find == 1)
       {
              break;
       }
    }
    return find == 1 ? hash_item_node : NULL;
}
4. int lustre_hash_additem (struct lustre_class_hash_body *hash_body, void *key, struct
{
   int hashent;

   hashent = lustre_hash_body->lchb_hash_operations->lustre_class_hashfn(hash_body, key
   if ( hashent == -1 )
      return -EINVAL;

   /* first, lock the hashbucket */
   spin_lock(hash_body->lchb_hash_tables[hashent]->lchb_lock);
   if ( (lustre_hash_getitem_in_bucket_nolock(hash_body, hashent, key)) != NULL) {
       /* the added item exist in hashtables, So cannot add it */
       spin_unlock(hash_body->lchb_hash_tables[hashent]->lchb_lock);
       print error message;
       return -EALREADY;
   }

   hlist_add_head(actual_hnode, hash_body->lchb_hash_tables[hashent]);

#ifdef LUSTRE_HASH_DEBUG
      hash_body->lchb_hash_tables[hashent]->lhb_item_count++; /* hash distribute debug
#endif

    hash_body->lchb_hash_operations->lustre_hash_object_refcount_get(actual_hnode); /*
    spin_unlock(hash_body->lchb_hash_tables[hashent]->lchb_lock);

    return 0;
}

4. int lustre_hash_delitem_nolock(struct lustre_hash_body *hash_body, int hashent, stru
{

   if (hash_item == NULL) {
```

```
        return -ENOENT;
    }
    hlist_del_init(hash_item);
    hash_body->lchb_hash_operations->lustre_hash_object_refcount_put(hash_item); /* do t
#ifdef LUSTRE_HASH_DEBUG
       hash_body->lchb_hash_tables[hashent]->lhb_item_count--;
#endif

    return 0;
}
5. int lustre_hash_delitem_by_key(struct lustre_class_hash_body *hash_body, void *key)
{
    int hashent ;
    struct hlist_node * hash_item;
    int retval = 0;

    hashent = lustre_hash_body->lchb_hash_opeartions->lustre_class_hashfn(hash_body, key
    if ( hashent == -1 )
       return -EINVAL;

    /* first, lock the hashbucket */
    spin_lock(hash_body->lchb_hash_tables[hashent]->lchb_lock);

    /* get the hash_item from hash_bucket */
    hash_item = lustre_hash_getitem_in_bucket_nolock(hash_body, hashent, key);

    /* call a delitem_nolock() to delete the hash_item */
    retval = lustre_hash_delitem_nolock(hash_body, hashent, hash_item);

    spin_unlock(hash_body->lchb_hash_tables[hashent]->lchb_lock);
    return retval;
}

int lustre_hash_delitem(struct lustre_clas_hash_body *hash_body, void *key, struct hlis
{
    int hashent ;
    int retval = 0;

    hashent = hash_body->lchb_hash_operations->lustre_class_hashfn(hash_body, key);
    if ( hashent == -1 )
       return -EINVAL;

    /* first, lock the hashbucket */
    spin_lock(hash_body->lchb_hash_tables[hashent]->lchb_lock);
```

```
   /* call a delitem_nolock() to delete the hash_item */
   retval = lustre_hash_delitem_nolock(hash_body, hashent, hash_item);

   spin_unlock(hash_body->lchb_hash_tables[hashent]->lchb_lock);
   return retval;
}

6. void * lustre_hash_get_object_by_key( struct lustre_class_hash_body *hash_body, void
{
    int hashent ;
    struct hlist_node * hash_item_hnode = NULL;
    void * obj_value = NULL;

   /* get the hash value from the given item */
   hashent = hash_body->lustre_class_hashfn(lustre_body, key);
   if ( hashent == -1 )
      return -EINVAL;
    rcu_read_lock();
    hash_item_hnode = lustre_hash_getitem_in_bucket_nolock(hash_body, hashent, key, NUL

    if (hash_item_hnode == NULL) {
        rcu_read_unlock();
        print error message;
        return NULL;
    }
    obj_value = hash_body->lchb_hash_operations->lustre_hash_object_refcount_get(hash_i
    rcu_read_unlock();
    return obj_value;
}
```

5.2 UUID-HASH function define :

/* string hashing using djb2 hash algorithm */

```
7. __s32 uuid_hashfn(struct lustre_hash_body *hash_body,  void * key)
{
       __s32 hash = 5381;
       struct obd_uuid * uuid_key = NULL;
       int c;
       char *ptr = NULL;
       LASSERT(key != NULL);

       uuid_key = (struct obd_uuid*)key;

       ptr = uuid_key->uuid;
```

```
        while ((c = *ptr++)) {
             hash = hash * 33 + c;
        }

        hash %= hash_body->lhcb_max_hash_size;
        if ( hash < 0 || hash > hash_body->lhcb_max_hash_size )
            hash = -1;
        return hash;
}


8. int uuid_hash_key_compare(void *key, struct hlist_node * compared_hnode);
{

     struct obd_export *export = NULL;
     struct obd_uuid *uuid_key = NULL, *compared_uuid = NULL;

     LASSERT( key != NULL);
     uuid_key = (struct obd_uuid*)key;

     export = hlist_entry(compared_node, struct obd_export, exp_uuid_hash);
     compared_uuid = &export->exp_client_uuid;
     return obd_uuid_equals( uuid_key, compared_uuid);
}
9. void * lustre_hash_object_refcount_get(struct hlist_node * actual_hnode)
{
    struct obd_export *export = NULL;


    LASSERT(actual_hnode != NULL);
    export = hlist_entry( actual_hnode, struct obd_export, exp_uuid_hash);
    LASSERT((export != NULL) && (sizeof(*export) == sizeof(struct obd_export)));

    class_export_get(export);

    return export;
}

10. void lustre_hash_object_refcount_put(struct hlist_node * actual_hnode)
{
    struct obd_export *export = NULL;


    LASSERT(actual_hnode != NULL);
    export = hlist_entry( actual_hnode, struct obd_export, exp_uuid_hash);
    LASSERT((export != NULL) && (sizeof(*export) == sizeof(struct obd_export)));
```

```
        class_export_put(export);

    }
```

5.3 Actual object for example (uuid-export hash)

5.3.1 Create an uuid-hash-body :

```
int class_setup(struct obd_device *obd, struct lustre_cfg *lcfg)
    {
    int err = 0;
    struct obd_export *exp;
    ENTRY;
    LASSERT(obd != NULL);
    LASSERTF(obd == class_num2obd(obd->obd_minor), "obd %p != obd_devs[%d]
%p\n",
    obd, obd->obd_minor, class_num2obd(obd->obd_minor));
    LASSERTF(obd->obd_magic == OBD_DEVICE_MAGIC, "obd %p obd_magic
%08x != %08x\n",
    obd, obd->obd_magic, OBD_DEVICE_MAGIC);
    /* have we attached a type to this device? */
    if (!obd->obd_attached) {
    CERROR("Device %d not attached\n", obd->obd_minor);
    RETURN(-ENODEV);
    }
    if (obd->obd_set_up) {
    CERROR("Device %d already setup (type %s)\n",
    obd->obd_minor, obd->obd_type->typ_name);
    RETURN(-EEXIST);
    }
    /* is someone else setting us up right now? (attach inits spinlock) */
    spin_lock(&obd->obd_dev_lock);
    if (obd->obd_starting) {
    spin_unlock(&obd->obd_dev_lock);
    CERROR("Device %d setup in progress (type %s)\n",
    obd->obd_minor, obd->obd_type->typ_name);
    RETURN(-EEXIST);
    }
    /* just leave this on forever. I can't use obd_set_up here because
    other fns check that status, and we're not actually set up yet. */
    obd->obd_starting = 1;
    spin_unlock(&obd->obd_dev_lock);
    exp = class_new_export(obd, &obd->obd_uuid);
    if (IS_ERR(exp))
    RETURN(PTR_ERR(exp));
```

```
obd->obd_self_export = exp;
list_del_init(&exp->exp_obd_chain_timed);
class_export_put(exp);
err = obd_setup(obd, sizeof(*lcfg), lcfg);
if (err)
GOTO(err_exp, err);
+
+ lustre_hash_init(obd->uuid_hash_body, "UUID_HASH", 128, &uuid_hash_operations);
+
obd->obd_set_up = 1;
spin_lock(&obd->obd_dev_lock);
/* cleanup drops this */
class_incref(obd);
spin_unlock(&obd->obd_dev_lock);
CDEBUG(D_IOCTL, "finished setup of obd %s (uuid %s)\n",
obd->obd_name, obd->obd_uuid.uuid);
RETURN(0);
err_exp:
CERROR("setup %s failed (%d)\n", obd->obd_name, err);
class_unlink_export(obd->obd_self_export);
obd->obd_self_export = NULL;
obd->obd_starting = 0;
RETURN(err);
}
```

5.3.2 Delete an uuid-hash-body :

```
int class_cleanup(struct obd_device *obd, struct lustre_cfg *lcfg)
{
int err = 0;
char *flag;
ENTRY;
OBD_RACE(OBD_FAIL_LDLM_RECOV_CLIENTS);
if (!obd->obd_set_up) {
CERROR("Device %d not setup\n", obd->obd_minor);
RETURN(-ENODEV);
}
spin_lock(&obd->obd_dev_lock);
if (obd->obd_stopping) {
spin_unlock(&obd->obd_dev_lock);
CERROR("OBD %d already stopping\n", obd->obd_minor);
RETURN(-ENODEV);
}
/* Leave this on forever */
obd->obd_stopping = 1;
spin_unlock(&obd->obd_dev_lock);
```

```
    if (lcfg->lcfg_bufcount >= 2 && LUSTRE_CFG_BUFLEN(lcfg, 1) > 0)
{
    for (flag = lustre_cfg_string(lcfg, 1); *flag != 0; flag++)
    switch (*flag) {
    case 'F':
    obd->obd_force = 1;
    break;
    case 'A':
    LCONSOLE_WARN("Failing over %s\n",
    obd->obd_name);
    obd->obd_fail = 1;
    obd->obd_no_transno = 1;
    obd->obd_no_recov = 1;
    /* Set the obd readonly if we can */
    if (OBP(obd, iocontrol))
    obd_iocontrol(OBD_IOC_SET_READONLY,
    obd->obd_self_export,
    0, NULL, NULL);
    break;
    default:
    CERROR("unrecognised flag '%c'\n",
    *flag);
    }
    }
    /* The three references that should be remaining are the
    * obd_self_export and the attach and setup references. */
    if (atomic_read(&obd->obd_refcount) > 3) {
    if (!(obd->obd_fail || obd->obd_force)) {
    CERROR("OBD %s is still busy with %d references\n"
    "You should stop active file system users,"
    " or use the –force option to cleanup.\n",
    obd->obd_name, atomic_read(&obd->obd_refcount));
    dump_exports(obd);
    GOTO(out, err = -EBUSY);
    }
    /* refcounf - 3 might be the number of real exports
    (excluding self export). But class_incref is called
    by other things as well, so don't count on it. */
    CDEBUG(D_IOCTL, "%s: forcing exports to disconnect: %d\n",
    obd->obd_name, atomic_read(&obd->obd_refcount) - 3);
    dump_exports(obd);
    class_disconnect_exports(obd);
    }
    LASSERT(obd->obd_self_export);
    /* Precleanup stage 1, we must make sure all exports (other than the
    self-export) get destroyed. */
```

```
+
+ if (obd->uuid_hash_body != NULL) {
+ lustre_hash_exit(obd->uuid_hash_body);
+ }
err = obd_precleanup(obd, OBD_CLEANUP_EXPORTS);
if (err)
CERROR("Precleanup %s returned %d\n",
obd->obd_name, err);
class_decref(obd);
obd->obd_set_up = 0;
RETURN(0);
out:
/* Allow a failed cleanup to try again. */
obd->obd_stopping = 0;
RETURN(err);
}
```

5.3.3 Add an uuid-item to uuid-hash:

```
/* Creates a new export, adds it to the hash table, and returns a
 * pointer to it. The refcount is 2: one for the hash reference, and
 * one for the pointer returned by this function. */
struct obd_export *class_new_export(struct obd_device *obd,
struct obd_uuid *cluuid)
{
struct obd_export *export, *tmp;
OBD_ALLOC(export, sizeof(*export));
if (!export)
return ERR_PTR(-ENOMEM);
export->exp_conn_cnt = 0;
atomic_set(&export->exp_refcount, 2);
export->exp_obd = obd;
CFS_INIT_LIST_HEAD(&export->exp_outstanding_replies);
/* XXX this should be in LDLM init */
CFS_INIT_LIST_HEAD(&export->exp_ldlm_data.led_held_locks);
spin_lock_init(&export->exp_ldlm_data.led_lock);
CFS_INIT_LIST_HEAD(&export->exp_handle.h_link);
class_handle_hash(&export->exp_handle, export_handle_addref);
export->exp_last_request_time = CURRENT_SECONDS;
spin_lock_init(&export->exp_lock);
export->exp_client_uuid = *cluuid;
obd_init_export(export);
- spin_lock(&obd->obd_dev_lock);
if (!obd_uuid_equals(cluuid, &obd->obd_uuid)) {
- list_for_each_entry(tmp, &obd->obd_exports, exp_obd_chain) {
- if (obd_uuid_equals(cluuid, &tmp->exp_client_uuid)) {
```

+ if (lustre_hash_additem (obd->uuid_hash_body, cluuid, &export->exp_uuid_hash)
!= 0){
- spin_unlock(&obd->obd_dev_lock);
CWARN("%s: denying duplicate export for %s\n",
obd->obd_name, cluuid->uuid);
class_handle_unhash(&export->exp_handle);
OBD_FREE_PTR(export);
return ERR_PTR(-EALREADY);
}
}
+ spin_lock(&obd->obd_dev_lock);
LASSERT(!obd->obd_stopping); /* shouldn't happen, but might race */
class_incref(obd);
list_add(&export->exp_obd_chain, &export->exp_obd->obd_exports);
list_add_tail(&export->exp_obd_chain_timed,
&export->exp_obd->obd_exports_timed);
export->exp_obd->obd_num_exports++;
spin_unlock(&obd->obd_dev_lock);
return export;
}
EXPORT_SYMBOL(class_new_export);

5.3.4 Delete a uuid-item from uuid-hash:

void class_unlink_export(struct obd_export *exp)
{
class_handle_unhash(&exp->exp_handle);
spin_lock(&exp->exp_obd->obd_dev_lock);
+ if (!hlist_unhashed(&exp->exp_uuid_hash)) {
+ lustre_hash_delitem (exp->exp_obd->uuid_hash_body, &exp->exp_client_uuid,
&exp->exp_uuid_hash)
+ }
list_del_init(&exp->exp_obd_chain);
list_del_init(&exp->exp_obd_chain_timed);
exp->exp_obd->obd_num_exports–;
spin_unlock(&exp->exp_obd->obd_dev_lock);
class_export_put(exp);
}

5.3.5 Get export-object from uuid-hash with given obd_uuid key:

int target_handle_connect(struct ptlrpc_request *req, svc_handler_t han-
dler)
{
struct obd_device *target, *targref = NULL;
struct obd_export *export = NULL;

```
    struct obd_import *revimp;
    struct lustre_handle conn;
    struct obd_uuid tgtuuid;
    struct obd_uuid cluuid;
    struct obd_uuid remote_uuid;
    struct list_head *p;
    char *str, *tmp;
    int rc = 0, abort_recovery;
    struct obd_connect_data *data;
    int size[2] = { sizeof(struct ptlrpc_body), sizeof(*data) };
    ENTRY;
    OBD_RACE(OBD_FAIL_TGT_CONN_RACE);
    LASSERT_REQSWAB(req, REQ_REC_OFF);
    str = lustre_msg_string(req->rq_reqmsg, REQ_REC_OFF, sizeof(tgtuuid)-
1);
    if (str == NULL) {
    DEBUG_REQ(D_ERROR, req, "bad target UUID for connect");
    GOTO(out, rc = -EINVAL);
    ...
    if (lustre_msg_get_op_flags(req->rq_reqmsg) & MSG_CONNECT_LIBCLIENT)
{
    if (!data) {
    DEBUG_REQ(D_WARNING, req, "Refusing old (unversioned) "
    "libclient connection attempt\n");
    GOTO(out, rc = -EPROTO);
    } else if (data->ocd_version < LUSTRE_VERSION_CODE -
    LUSTRE_VERSION_ALLOWED_OFFSET) {
    DEBUG_REQ(D_WARNING, req, "Refusing old (%d.%d.%d.%d) "
    "libclient connection attempt\n",
    OBD_OCD_VERSION_MAJOR(data->ocd_version),
    OBD_OCD_VERSION_MINOR(data->ocd_version),
    OBD_OCD_VERSION_PATCH(data->ocd_version),
    OBD_OCD_VERSION_FIX(data->ocd_version));
    data = lustre_msg_buf(req->rq_repmsg, REPLY_REC_OFF,
    offsetof(typeof(*data),
    ocd_version) +
    sizeof(data->ocd_version));
    if (data) {
    data->ocd_connect_flags = OBD_CONNECT_VERSION;
    data->ocd_version = LUSTRE_VERSION_CODE;
    }
    GOTO(out, rc = -EPROTO);
    }
    }
    /* lctl gets a backstage, all-access pass. */
    if (obd_uuid_equals(&cluuid, &target->obd_uuid))
```

```
    goto dont_check_exports;
    spin_lock(&target->obd_dev_lock);
    - list_for_each(p, &target->obd_exports) {
    - export = list_entry(p, struct obd_export, exp_obd_chain);
    - if (obd_uuid_equals(&cluuid, &export->exp_client_uuid)) {
    + export = lustre_hash_get_object_by_key(target->uuid_hash_body, clu-
uid);
    if (export->exp_connecting) { /* bug 9635, et. al. */
    CWARN("%s: exp %p already connecting\n",
    export->exp_obd->obd_name, export);
    + class_export_put(export);
    export = NULL;
    rc = -EALREADY;
    - break;
    + } else {
    export->exp_connecting = 1;
    + class_export_put(export);
    - spin_unlock(&target->obd_dev_lock);
    LASSERT(export->exp_obd == target);
    rc = target_handle_reconnect(&conn, export, &cluuid);
    - break;
    }
    - export = NULL;
    - }
    /* If we found an export, we already unlocked. */
    if (!export) {
    spin_unlock(&target->obd_dev_lock);
    OBD_FAIL_TIMEOUT(OBD_FAIL_TGT_DELAY_CONNECT, 2 * obd_timeout);
    } else if (lustre_msg_get_conn_cnt(req->rq_reqmsg) == 1) {
    CERROR("%s: NID %s (%s) reconnected with 1 conn_cnt; "
    "cookies not random?\n", target->obd_name,
    libcfs_nid2str(req->rq_peer.nid), cluuid.uuid);
    GOTO(out, rc = -EALREADY);
    } else {
    OBD_FAIL_TIMEOUT(OBD_FAIL_TGT_DELAY_RECONNECT, 2 *
obd_timeout);
    }
    ...
    revimp = export->exp_imp_reverse = class_new_import(target);
    revimp->imp_connection = ptlrpc_connection_addref(export->exp_connection);
    revimp->imp_client = &export->exp_obd->obd_ldlm_client;
    revimp->imp_remote_handle = conn;
    revimp->imp_dlm_fake = 1;
    revimp->imp_state = LUSTRE_IMP_FULL;
    if (lustre_msg_get_op_flags(req->rq_reqmsg) & MSG_CONNECT_NEXT_VER)
{
```

revimp->imp_msg_magic = LUSTRE_MSG_MAGIC_V2;
lustre_msg_add_op_flags(req->rq_repmsg, MSG_CONNECT_NEXT_VER);
}
class_import_put(revimp);
out:
if (export)
export->exp_connecting = 0;
if (targref)
class_decref(targref);
if (rc)
req->rq_status = rc;
RETURN(rc);
}

5.3.6 list of all code that walks obd_exports (via exp_obd_chain) and should needs to be fixed with the hash class:

1.lustre/ldlm/ldlm_lib.c:
int target_handle_connect(struct ptlrpc_request *req, svc_handler_t handler)
export = list_entry(p, struct obd_export, exp_obd_chain);
2. lustre/obdclass/genops.c
int obd_export_evict_by_nid(struct obd_device *obd, char *nid)
3. lustre/obdclass/genops.c
int obd_export_evict_by_uuid(struct obd_device *obd, char *uuid)
4. lustre/obdclass/genops.c
struct obd_export *class_new_export(struct obd_device *obd,
struct obd_uuid *cluuid)
5. lustre/obdclass/genops.c
static void class_disconnect_export_list(struct list_head *list, int flags)
6. lustre/obdfilter/filter.c
static void filter_grant_sanity_check(struct obd_device *obd, const char *func)

5.3.7 code list of conn_list

ptlrpc/connection.c : struct ptlrpc_connection * ptlrpc_lookup_conn_locked (lnet_process_id_t peer)

6 State Specification

N/A