Detailed Level Design for bug 10707(based upon b_release_1_4_6)

Tian Zhiyong

2006.8.4—2006.8.9

1 Functional Specification

1. Because of qd_count in struct `qunit_data` is 32bit, it may be overflowed in some conditions. So we will change it to 64bit.
2. After changing 32bit to 64bit, we will keep compatible.
3. Merging qd_type and qd_isblk into qd_flags of structre `qunit_data` which will use binary operation.

1.1 Data structure

- Changing struct `qunit_data` from

```
struct qunit_data {
   __u32 qd_id; /* ID appiles to (uid, gid) */
   __u32 qd_type; /* Quota type (USRQUOTA, GRPQUOTA) */
   __u32 qd_count; /* acquire/release count (bytes for block quota) */
__u32 qd_isblk; /* Block quota or file quota */
};
into
struct qunit_data {
   __u32 qd_id; /* ID appiles to (uid, gid) */
   __u32 qd_flags; /* Quota type (USRQUOTA, GRPQUOTA)occupy one bit; Block
   __u64 qd_count; /* acquire/release count (bytes for block quota) */
};
Adding structre qunit_data_old:
struct qunit_data_old {
   __u32 qd_id; /* ID appiles to (uid, gid) */
   __u32 qd_type; /* Quota type (USRQUOTA, GRPQUOTA) */
   __u32 qd_count; /* acquire/release count (bytes for block quota) */
   __u32 qd_isblk; /* Block quota or file quota */
};
```

```
Adding and changing some "#define"
In the file lustre_idl.h, adding macro
#define OBD_CONNECT_QUOTA64 0x80000ULL /*64bit for qd_count*/
#define MAX_QUOTA_COUNT32 (0xffffffffULL >> QUOTABLOCK_BITS <<QUOTABLOCK_BITS)
In the file lustre_idl.h, adding macro like this
#define QUOTA_IS_GRP   0x1UL  /* 0 is user, 1 is group*/
#define QUOTA_IS_BLOCK 0x2UL  /* 0 is inode, 1 is block*/
and change OST_CONNECT_SUPPORTED into
-   #define OST_CONNECT_SUPPORTED (OBD_CONNECT_SRVLOCK | OBD_CONNECT_GRANT | OBD_CONNEC
+   #define OST_CONNECT_SUPPORTED (OBD_CONNECT_SRVLOCK | OBDCONNECT_GRANT | OBD_CONNECT
in the file obd_support.h
+  #define OBD_FAIL_QUOTA_QD_COUNT_32BIT 0xA00
```

1.2 Main functions

This only includes the new added functions.

1.2.1 should_translate_quota

int should_translate_quota (struct obd_import *imp)

Parameters:

struct obd_import *imp: the imp which we will find the corresponding export.

Return value:

return 1 if need to split, 0 if don't need to split.

Description:

When a slave sends a quota request to a master, this function decides whether need tranlate it to structure qunit_data_old.

1.2.2 lustre_swab_qdata

void lustre_swab_qdata (struct qunit_data *d)

Parameters:

struct qunit_data *d: the qunit_data need to be swapped.

Return value:

none

Description:

swap the content of qunit_data in order to deal with it.

1.2.3 lustre_swab_qdata_old

void lustre_swab_qdata_old (struct qunit_data_old *d)

Parameters:

struct qunit_data_old *d: the qunit_data_old need to be swapped.

Return value:

none

Description:

swap the content of qunit_data_old in order to deal with it.

1.2.4 quota_old_to_new

struct qunit_data *quota_old_to_new(struct qunit_data_old *d)
Parameters:
struct qunit_data_old *d: the qunit_data_old need to change the format.
Return value:
return a pointer to struct qunit_data if successful, otherwise return NULL.
Description:
change the format of struct qunit_data_old into the format of struct qunit_data.

1.2.5 quota_new_to_old

struct qunit_data_old *quota_new_to_old(struct qunit_data *d)
Parameters:
struct qunit_data *d: the qunit_data need to change the format.
Return value:
return a pointer to struct qunit_data_old if successful, otherwise return NULL.
Description:
change the format of struct qunit_data into the format of struct qunit_data_old.

1.3 Trifling functions:

Because I change the qd_type and qd_isblk of struct qunit_data, I will do some trivial fix on functions which have used them. I just list them, fixing is simple and I won't explain one by one.

1. dqacq_handler in file quota_master.c
2. dqacq_completion in file quota_context.c
3. find_qunit in file quota_context.c
4. qunit_hashfn in file quota_context.c
5. chech_cur_qunit in file quota_context.c
6. macro QDATA_DEBUG
7. qctxt_adjust_qunit in file quota_context.c
8. qctxt_wait_pending_dqacq in file quota_context.c
9. qslave_recovery_main in file quota_context.c
10. check_qunit_data in file wirecheck.c
11. add the item in struct obd_connect_names of file lprocfs_status.c.

12. change some LASSERTF in file pack_generic.c.
13. change some LASSERTF in file lustre/utils/wiretest.c.

2 Use Cases

2.1 Initial stage

1. At the beginning, a mds(master) will connect to an ost(slave). It will call mds_lov_connect, which will add OBD_CONNECT_QUOTA64 to data->ocd_conncect_flags. this data will be transfered to the ost(slave).
2. The ost receives the data(function filter_connect_internal). If it finds the data sets the OBD_CONNECT_QUOTA64 bit, it will set its corresponding OBD_CONNECT_QUOTA64 bit of export, and send it back.
3. The mds receives the reply(function ptlrpc_connect_interpret). if it finds the OBD_CONNECT_QUOTA64 bit of data has been set, it will set the OBD_CONNECT_QUOTA64 bit of relative import and export of this connection.

2.2 Acquire/release quota

1. When a slave(ost) acquires/releases the quota, it will create a qunit_data to the master(mds' function schedule_dqacq). It will check the corresponding import(function should_translate_quota). If import->imp_connect_data.ocd_connect_flags is zero, it will assign it to the corresponding export's exp_connect_flags. Then if ocd_connect_flags contains OBD_CONNECT_QUOTA64, the slave will send directly.
2. The master receives the qunit_data and handle it(function target_handle_dqacq_callback). Then send it back to the slave.
3. The slave receives the quota_data reply(function dqacq_interpret). it will decreases/increases the quota limitation.

2.3 Release >4G quota

1. A client creates multi huge files (>8G)
2. The client then delete them.
3. At this time, a ost will release much more than 4G quota. after this fix, it will be fine; before this, the lustre's ost will be in an endless loop because of an endless loop.

2.4 Test method

In the functions of should_translate_quota, target_handle_dqacq_callback, dqacq_interpret, adding OBD_FAIL_CHECK macro so that master and slave can use 32bit's qd_count when the test needs. For example, when needing the test running 32bit of qd_count, we can do "sysctl -w lustre.fail_loc=2304"(2304 = 0x900)

3 Logic Specification

3.1 lustre_swab_qdata(changed)

```
void lustre_swab_qdata(struct qunit_data *d) {
    __swab32s (&d->qd_id);
    __swab32s (&d->qd_flags);
    __swab64s (&d->qd_count);
}
```

3.2 lustre_swab_qdata_old(a new function)

```
void lustre_swab_qdata_old(struct qunit_data_old *d) {
    __swab32s (&d->qd_id);
    __swab32s (&d->qd_type);
    __swab32s (&d->qd_count);
__swab32s (&d->qd_isblk);
}
```

3.3 quota_old_to_new(a new function)

```
struct qunit_data *quota_old_to_new(struct qunit_data_old *d){
    struct qunit_data_old tmp;
    struct qunit_data *ret;

    if (!d)
         return d;
    tmp = *d;
    ret = (struct qunit_data *)d;
    ret->qd_id = tmp.qd_id;
    ret->qd_flags = (tmp.qd_type ? QUOTA_IS_GRP : 0) | (tmp.qd_isblk ? QUOTA_IS_BLOCK :
    ret->qd_count = tmp.qd_count;
    return ret;
}
```

3.4 quota_new_to_old(a new function)

```
struct qunit_data_old *quota_new_to_old(struct qunit_data *d){
    struct qunit_data tmp;
    struct qunit_data_old *ret;

    if (!d)
         return d;
    LASSERT(d->qd_count <= MAX_QUOTA_COUNT32);
    tmp = *d;
    ret = (struct qunit_data_old *)d;
    ret->qd_id = tmp.qd_id;
    ret->qd_type = ((tmp.qd_flags & QUOTA_IS_GRP) ? GRPQUOTA : USRQUOTA);
    ret->qd_count = (__u32)tmp.qd_count;
    ret->qd_isblk = ((tmp.qd_flags & QUOTA_IS_BLOCK) ? 1 : 0);
    return ret;
}
```

3.5 should_translate_quota(a new function)

```
int should_translate_quota (struct obd_import *imp){
      struct obd_device *obd;
      struct obd_export *tmp;
      int ret = 0;

      LASSERT(imp);
      if (imp->imp_connect_data.ocd_connect_flags)
            if (imp->imp_connect_data.ocd_connect_flags & OBD_CONNECT_QUOTA64)
                  return 0;
            else
                  return 1;
      obd = imp->imp_obd;
      spin_lock(&obd->obd_dev_lock);
      list_for_each_entry(tmp,&obd->obd_exports,exp_obd_chain){
           if (tmp->exp_imp_reverse == imp){
                imp->imp_connect_data.ocd_connect_flags = tmp->exp_connect_flags;
                spin_unlock(&obd->obd_dev_lock);
                if (tmp->exp_connect_flags & OBD_CONNECT_QUOTA64)
                      return 0;
                else
                      return 1;
           }
      }
      spin_unlock(&obd->obd_dev_lock);
      CDEBUG(D_QUOTA,"don't find the corresponding export!");
```

```
        return 0;
}
```

3.6 mds_lov_connect(changed)

```
int mds_lov_connect(struct obd_device *obd, char * lov_name){
....
-   data->ocd_connect_flags = OBD_CONNECT_VERSION | OBD_CONNECT_INDEX;
+   data->ocd_connect_flags = OBD_CONNECT_VERSION | OBD_CONNECT_INDEX | OBD_CONNECT_QUO
...
}
```

When a master(using osc on mds) connects to a slave(ost), it will call mds_lov_connect, changing it like above;

3.7 filter_connect_internal

```
static int filter_connect_internal(struct obd_export *exp,
struct obd_connect_data *data)
{
      ......
      exp->exp_connect_flags = data->ocd_connect_flags;
+     if (exp->exp_imp_reverse)
+         exp->exp_imp_reverse->imp_connect_data.ocd_connect_flags =       data->ocd_co
      data->ocd_version = LUSTRE_VERSION_CODE;
      ......
}
```

A slave(ost) will receive the connect data —-"data", it will change export's flag.
I add several lines to let the corresponding import's flag equal export's flag;

3.8 schedule_dqacq(changed)

```
static int schedule_dqacq(struct obd_device *obd, struct lustre_quota_ctxt *qctxt, stru
         ....
          LASSERT(qctxt->lqc_import);
          req = ptlrpc_prep_req(qctxt->lqc_import, LUSTRE_MDS_VERSION, opc, 1, &size, N
          if (!req) {
                dqacq_completion(obd, qctxt, qdata, -ENOMEM, opc);
                RETURN(-ENOMEM);
          }

   +      LASSERT(!should_translate_quota(qctxt->lqc_import) || qdata->qd_count <= MAX_
   +      if (should_translate_quota(qctxt->lqc_import) ||
```

```
    +            OBD_FAIL_CHECK(OBD_FAIL_QUOTA_QD_COUNT_32BIT)) {
    +            struct qunit_data_old *reqdata_old, *tmp;
    +            reqdata_old = lustre_msg_buf(req->rq_reqmsg, 0, sizeof(*reqdata_old));
    +            tmp = quota_new_to_old(qdata);
    +            *reqdata_old = *tmp;
    +            size = sizeof(*reqdata_old);
    +     } else {
    +            reqdata = lustre_msg_buf(req->rq_reqmsg, 0, sizeof(*reqdata));
    +            *reqdata = *qdata;
    +            size = sizeof(*reqdata);
    +     }
}
```

When a slave(ost) sends a acquire/release request to a master, it will call schedule_dqacq. This function is charged of whether translating the qunit_data into qunit_data_old. If needing to translate to old format, the qdata->qd_count must be less than MAX_QUOTA_COUNT32. It doesn't do split. Now split_before_schedule_dqacq does the work of split, schedule_dqacq does the work of translation. In function split_before_schedule_dqacq, it will call schedule_dqacq to send a quota request.

3.9 target_handle_dqacq_callback(changed)

```
int target_handle_dqacq_callback(struct ptlrpc_request *req) {
        ......
  -     struct qunit_data *qdata,*rep;
  +     struct qunit_data *qdata;
  +     void *rep;
  +     struct qunit_data_old *qdata_old;
        ......
  +     LASSERT(req->rq_export);
  +     if ((req->rq_export->exp_connect_flags & OBD_CONNECT_QUOTA64) &&
             !OBD_FAIL_CHECK(OBD_FAIL_QUOTA_QD_COUNT_32BIT)){
  +          rep = lustre_msg_buf(req->rq_repmsg, 0, sizeof(struct qunit_data));
  +          LASSERT(rep);
  +          qdata = lustre_swab_reqbuf(req, 0, sizeof(*qdata), lustre_swab_qdata);
  +     } else {
  +          rep = lustre_msg_buf(req->rq_repmsg, 0, sizeof(struct qunit_data_old));
  +          LASSERT(rep);
  +          qdata_old = lustre_swab_reqbuf(req, 0, sizeof(*qdata_old), lustre_swab_qd
  +          qdata = quota_old_to_new(qdata_old);
  +     }
        ......
  -     memcpy(rep,qdata,sizeof(*rep));
  +     if ((req->rq_export->exp_connect_flags & OBD_CONNECT_QUOTA64) &&             !
```

```
  +           memcpy(rep,qdata,sizeof(*qdata));
  +      } else {
  +           qdata_old = quota_new_to_old(qdata);
  +           memcpy(rep,qdata_old,sizeof(*qdata_old));
  +      }
         ......
}
```

When a master(mds) receives a quota request, this function will be called.

3.10 dqacq_interpret(changed)

```
static int dqacq_interpret(struct ptlrpc_request *req, void *data, int rc){
       ......
       struct qunit_data *qdata=NULL;
  +    struct qunit_data_old *qdata_old=NULL;
       ......
       ENTRY;
  -    qdata = lustre_swab_repbuf(req, 0, sizeof(*qdata), lustre_swab_qdata);
  +    LASSERT(req);
  +    LASSERT(req->rq_import);
  +    if ((req->rq_import->imp_connect_data.ocd_connect_flags & OBD_CONNECT_QUOTA64) &
  +             qdata = lustre_swab_reqbuf(req, 0, sizeof(*qdata), lustre_swab_qdata);
  +    } else {
  +             qdata_old = lustre_swab_reqbuf(req, 0, sizeof(struct qunit_data_old), l
  +             qdata = quota_old_to_new(qdata_old);
  +    }
       ......
}
When a reply from the master(mds) returns to the slave(ost), this function will be call
```

3.11 split_before_schedule_dqacq(a new function)

```
static int split_before_schedule_dqacq(struct obd_device *obd, struct lustre_quota_ctxt
           int rc = 0, ret;
           LASSERT(qdata);
           if (qctxt->lqc_import)
                while (should_translate_quota(qctxt->lqc_import) && \
                          qdata->qd_count > MAX_QUOTA_COUNT32) {
                     struct qunit_data tmp_qdata;
                     tmp_qdata = *qdata;
                     tmp_qdata.qd_count = MAX_QUOTA_COUNT32;
                     qdata->qd_count -= tmp_qdata.qd_count;
                     ret = schedule_dqacq(obd, qctxt, &tmp_data, opc, wait);
```

```
                        if (!rc)
                            rc = ret;
                    }
                if (qdata->qd_count){
                      ret = schedule_dqacq(obd, qctxt, qdata, opc, wait);
                      if (!rc)
                          rc = ret;
                }

                RETURN(rc);
    }
```

3.12 qctxt_adjust_qunit(changed)

```
    int qctxt_adjust_qunit(struct obd_device *obd, struct lustre_quota_ctxt *qctxt, uid_t u
            ......
            ret = check_cur_qunit(obd, qctxt, &qdata[i]);
            if (ret > 0) {
                 int opc;
                 /* need acquire or release */
                 opc = ret == 1 ? QUOTA_DQACQ : QUOTA_DQREL;
      -          ret = schedule_dqacq(obd, qctxt, &qdata[i], opc, wait);
      +          ret = split_before_schedule_dqacq(obd, qctxt, &qdata[i], opc, wait);
            ......
    }
```

This function will call split_before_schedule_dqacq. Function split_before_schedule_dqacq is charged of whether split a single qunit_data into multi qunit_data_old. The other two functions calling split_before_schedule_dqacq are: dqacq_completion, qslave_recovery_main. They will do the same things as qctxt_adjust_qunit, so I won't list the code again. Now split_before_schedule_dqacq does the work of split, schedule_dqacq does the work of translation.

4 Environment

4.1 Network Protocol Compatibility - New slaves, Old masters

1. In the initial stage, the slave and master won't set OBD_CONNECT_QUOTA64 bit on their corresponding export and import.
2. When a new slave acquires/releases quota, it will create a qunit_data.
3. The new slave checks the corresponding import and finds OBD_CONNECT_QUOTA64 bit isn't set.

4. If quota size more than 0xffffffffUL, it will split quota size into multi qunit_data_old and send them; if quota size less than 0xffffffffUL, it will translate into qunit_data_old and send it.

5. The old master understands qunit_data_old. It handles the quota request and sends it back.

6. The new slave checks the corresponding import and finds OBD_CONNECT_QUOTA64 bit isn't set.

7. The new slave will translate it from qunit_data_old into qunit_data and handle it.

4.2 Network Protocol Compatibility - Old slaves, New masters

1. In the initial stage, the slave and master won't set OBD_CONNECT_QUOTA64 bit on their corresponding export and import.

2. When an old slave acquires/releases quota, it will create a qunit_data_old and send it to the new master.

3. The new master checks the corresponding export and finds OBD_CONNECT_QUOTA64 bit isn't set. It will translate the qunit_data_old into qunit_data and handle it.

4. Before returning the reply to the old slave, it will translate the qunit_data into qunit_data_old and send it.

5. The old slave receives the reply in qunit_data_old format. It knows how to handle it.