DLD for bug10600

Tian Zhiyong

March 25, 2007

1 Functional Specification

1.1 Describe the problem

Currently, block quota is managed in qunit(default 100MB) granularity:

- Each slave(OST,MDS) acquires 1 qunit from master(MDS) at initialization for future write.
- Slave acquires 1 qunit from master when available quota is less than btune(default is half of qunit size).
- Slave release qunits to master when it's available quota more than btune + 1 qunit.

As above design, we'll meet several troubles while using quota:

- User can't set small quota limit($<$ ost num * qunit size). Each slave acquires 1 qunit at quota setting, if the limit is too low, some slave can't acquire qunit, then objects can't be created on them.
- If user don't use OSTs equally, some qunits allocated to the not used slaves will never be utilized. For example: there are 100 OSTs, and each one get one qunit at initialization, but user only use OST1, then the 99 qunits on other OSTs will never be used.

1.2 The solution to fix it

Shrink the qunit size automatically when the remaining quota on master is too low; get back the qunit size when quota is recycled. This idea boils down to six points:

- set the lowest and highest limitation of block/inode qunit size. Abbreviate the lowest and highest limitations for block to LLB and HLB; abbreviate the lowest and highest limitations for inode to LLI and HLI. (The lowest and highest limitations for block are LEAST_OF_BLOCK_QUNIT_SIZE and original block qunit size; the lowest and highest limitations for inode are LEAST_OF_INODE_QUNIT_SIZE and original inode qunit size)

- set the critical point for adjusting the qunit size. I call it CP.(CP = shrink_qunit_limit * ost num * current qunit size)
- when remaining quota is less than CP, shrink the corresponding qunit size to half of itself until qunit size reaches a reasonable and stable value. During this procedure, the current qunit size must >= LLI/LLB. When this happens, a broadcast to all the relative slaves will be observed.
- when remaining quota is less than shrink_qunit_limit * ost num * LEAST_OF_BLOCK_QUNIT_SIZE the master get all the free qunit from slaves.
- when quota limit or remaining quota is more than 2 * CP, double the corresponding qunit size until qunit size reaches a reasonable and stable value. During this procedure, the current qunit size must <= HLI/HLB. This information is just transferred to the active slaves and, a broadcast WON'T happen.
- In the initial phrase, every slave will get a new qunit. When qunit size has already reached the LLI/LLB and this slave don't use the quota for a long time(default 5 minutes), release that qunit.

In short, there are two states in lustre now. If remaining quota > shrink_qunit_limit * ost num * LEAST_OF_BLOCK_QUNIT_SIZE/LEAST_OF_INODE_QUNIT_SIZE, slaves have quota buffer(btune < remaining quota < 1 qunit + btune), called FULLBUF; if not, slaves haven't quota buffer like before(quota buffer is just less than LEAST_OF_BLOCK_QUNIT_SIZE or LEAST_OF_INODE_QUNIT_SIZE) and it just apply quota from master when they need, called LESSBUF. If lustre is in LESSBUF state, ONLY when remaining quota > 8 * shrink_qunit_limit * ost num * LEAST_OF_BLOCK_QUNIT_SIZE/LEAST_OF_INODE_QUNIT_SIZE, lustre will enter into FULLBUF.

1.3 Data structure

1.3.1 enable_modify_qunit_size

```
int enable_modify_qunit_size = 1;
```

If this value is not zero, lustre will change qunit size; otherwise, lustre won't change qunit size. This value exists in proc filesystem, which can be read and written by user. Default value is 1.

1.3.2 OST_QUOTA_ADJUST_QUNIT

```
#define OST_QUOTA_ADJUST_QUNIT 20
```

Adding a new RPC to ost. This one will adjust the qunit size. When needed, A master will send OST_QUOTA_ADJUST_QUNIT RPC to all of slaves to change their qunit size.

1.3.3 OBD_CONNECT_CHANGE_QS

```
#define OBD_CONNECT_CHANGE_QS 0x400000ULL /* shrink/enlarge qunit size b=10600 */
```

In order to handle this compatible problem, define this macro.

1.3.4 ADJUST_BLOCK_QUNIT_SIZE

```
#define ADJUST_BLOCK_QUNIT_SIZE 1
```

When block quota has been acquired/released, block qunit size maybe need to be changed. This macro indicates function *dquot_adjust_qunit_size* to do this; When needing to change block qunit size, this bit will be set.(this bit is at qaq_flags in struct obd_quota_adjust_qunit)

1.3.5 ADJUST_INODE_QUNIT_SIZE

```
#define ADJUST_INODE_QUNIT_SIZE 2
```

when file quota has been acquired/released, file qunit size maybe need to be changed. This macro indicates function *dquot_adjust_qunit_size* to do this; when needing to change inode qunit size, this bit will be set.(this bit is at qaq_flags in struct obd_quota_adjust_qunit)

1.3.6 shrink_qunit_limit

```
unsigned long shrink_qunit_limit = 4;
```

When the remaining quota on master is less than shrink_qunit_limit * ost num * qunit size, we will begin to shrink the qunit. Default value is 4. This value can be adjusted in proc filesystem.

1.3.7 SHRINK_QUNIT_PERCENT

```
#define SHRINK_QUNIT_FACTOR 2
```

When we begin to shrink the qunit, we will change *qunit size* to *qunit size / SHRINK_QUNIT_FACTOR*; when we begin to increase the qunit, we will change *qunit size* to *qunit size * SHRINK_QUNIT_*FACTOR.

1.3.8 LEAST_OF_BLOCK_QUNIT_SIZE

```
#define LEAST_OF_BLOCK_QUNIT_SIZE PTLRPC_MAX_BRW_SIZE
```

We can't reduce block qunit size infinitely, this is the smallest value for block qunit size. In b1_5, PTLRPC_MAX_BRW_SIZE is 1M bytes.

1.3.9 LEAST_OF_INODE_QUNIT_SIZE

```
#define LEAST_OF_INODE_QUNIT_SIZE 64
```

We can't reduce inode qunit size infinitely, this is the smallest value for inode qunit size.

1.3.10 DQ_BLKLESSBUF_B

```
#define DQ_BLKLESSBUF_B 31
```

When block of lustre is in LESSBUF state, this bit will be set on lqs_flags of struct lustre_qunit_size, on dq_flags of struct lustre_dquot, and on qd_flags of struct qunit_data.

1.3.11 DQ_INOLESSBUF_B

```
#define DQ_INOLESSBUF_B 30
```

When inode of lustre is in LESSBUF state, this bit will be set in lqs_flags in struct lustre_qunit_size, and in dq_flags in struct lustre_dquot.

1.3.12 lustre_dquot

```
struct lustre_dquot {
        /* Hash list in memory, protect by dquot_hash_lock */
        struct list_head dq_hash;
        /* Protect the data in lustre_dquot */
        struct semaphore dq_sem;
        /* Use count */
        int dq_refcnt;
        /* Pointer of quota info it belongs to */
        struct lustre_quota_info *dq_info;
        loff_t dq_off; /* Offset of dquot on disk */
        unsigned int dq_id; /* ID this applies to (uid, gid) */
        int dq_type; /* Type fo quota (USRQUOTA, GRPQUOUTA) */
        unsigned short dq_status; /* See DQ_STATUS_ */
        unsigned long dq_flags; /* See DQ_ in quota.h */
        struct mem_dqblk dq_dqb; /* Diskquota usage */
+       unsigned long dq_bunit_sz_curr;
+       unsigned long dq_iunit_sz_curr;
};
```

Adding two items to this structure and these two items aren't stored in the disk. dq_bunit_sz_curr and dq_iunit_sz_curr equal lqc_iunit_sz and lqc_iunit_sz in structure lustre_quota_ctxt initially. When quota is acquired/released, dq_bunit_sz_curr and dq_iunit_sz_curr may be changed. A master needs them.

1.3.13 lustre_qunit_size

```
struct lustre_qunit_size {
        struct list_head lqs_hash;      /* the hash entry */
        unsigned int lqs_id;            /* id of user/group */
        unsigned long lqs_flags;        /* is user/group; FULLBUF or LESSBUF */
        unsigned long lqs_iunit_sz;     /* Unit size of file quota currently */
        unsigned long lqs_itune_sz;     /* Trigger dqacq when available file quota less
                                         * this value, trigger dqrel when
                                         * more than this value + 1 iunit */
        unsigned long lqs_bunit_sz;     /* Unit size of block quota currently */
        unsigned long lqs_btune_sz;     /* See comment of lqs_itune_sz */
}
```

Now, because qunit size can be changed, every user/group must have its own qunit size. On every slave, using such a structure to present qunit size of a group/user.

1.3.14 lustre_quota_ctxt

```
struct lustre_quota_ctxt {
         struct super_block *lqc_sb;     /* superblock this applies to */
         struct obd_import *lqc_import;  /* import used to send dqacq/dqrel RPC */
         dqacq_handler_t lqc_handler;    /* dqacq/dqrel RPC handler, only for quota mast
         unsigned long lqc_recovery:1;   /* Doing recovery */
         unsigned long lqc_iunit_sz;     /* Unit size of file quota originally */
         unsigned long lqc_itune_sz;     /* Trigger dqacq when available file quota less
                                          * this value, trigger dqrel when available fil
                                          * more than this value + 1 iunit */
         unsigned long lqc_bunit_sz;     /* Unit size of block quota originally*/
         unsigned long lqc_btune_sz;     /* See comment of lqc_itune_sz */
+        spinlock_t lqc_lqs_lock;        /* this lock protects lqc_lqs_hash list. */
+        struct list_head lqc_lqs_hash[NR_QUSHASH]; /* all lustre_qunit_size structure i
}
```

Every user and group will have a structure lustre_qunit_size, which is linked to this structure and will be found by hash method. When allocating/releasing quota, a slave will find the corresponding structure lustre_qunit_size and read it, which must hold the lock of lqc_lqs_lock during it; When shrinking or enlarging qunit size, a slave will find the corresponding structure lustre_qunit_size and change it, which must hold the lock of lqc_lqs_lock too during it. So lqc_lqs_lock ensures the operations in atomicity.

1.3.15 obd_quota_adjust_qunit

```
struct obd_quota_adjust_qunit {
        __u32 qaq_flags;
```

```
            __u32 qaq_type;
            __u32 qaq_id;
            __u32 qaq_bunit_sz;
            __u32 qaq_iunit_sz;
            __u32 padding;
    };
```

When a master sends the command to slaves to adjust qunit size, this structure will be used. qaq_flags will contains ADJUST_BLOCK_QUNIT_SIZE, ADJUST_INODE_QUNIT_SIZE or both of them and FULLBUF or LESSBUF state(including inode and block), which will instruct the slaves to adjust block/inode qunit size or both of them; qaq_type presents group or user; qaq_id presents group/user id; qaq_bunit_sz presents the new block qunit size; qaq_iunit_sz means new inode qunit size.

1.3.16 NR_QUSHASH

```
#define NR_QUSHASH NR_DQHASH
```

NR_QUSHASH presents the hash entry number of structure lustre_qunit_size in structure lustre_quota_ctxt.

1.4 Function name

1.4.1 lov_quota_adjust_qunit(new function)

int lov_quota_adjust_qunit(struct obd_export *exp, struct obd_quota_adjust_qunit *oqaq)

Parameters:

obd_export *exp: through it, a mds can can connect to the corresponding ost.

struct obd_quota_adjust_qunit *oqaq: the argument wanted when modifying the qunit size.

Return value:

return 0 if successful, otherwise return err code.

Description:

When a mds finds it necessary to change qunit size, it will call this function. Two functions which will call it are *mds_init_slave_blimits* and *target_handle_dqacq_callback*. This function will send enlarge/shrink qunit size command to every relative slave.

1.4.2 client_quota_adjust_qunit(new function)

int client_quota_adjust_qunit(struct obd_export *exp, struct obd_adjust_qunit *oqaq)

Parameters:

obd_export *exp: through it, a mds can can connect to the corresponding ost.

struct obd_quota_adjust_qunit *oqaq: the argument wanted when modifying the qunit size.

Return value:

return 0 if successful, otherwise return err code.

Description:

Function lov_quota_adjust_qunit will call this function, which will be in charge of sending shrink qunit size command to a relative slave.

1.4.3 ost_handle_quota_adjust_qunit(new function)

static int ost_handle_quota_adjust_qunit(struct ptlrpc_request *req)

Parameters:

struct ptlrpc_request *req: this is the request of enlarge/shrink qunit size which a master sends.

Return value:

return 0 if successful, otherwise return err code.

Description:

When an ost receives a shrink qunit size RPC, it will call this function.

1.4.4 filter_quota_adjust_qunit(new function)

int filter_quota_adjust_qunit(struct obd_export *exp, struct obd_quota_adjust_qunit *oqaq)

Parameters:

obd_export *exp: it is used to connect with the master(mds).

struct obd_quota_adjust_qunit *oqaq: the argument wanted when modifying the qunit size.

Return value:

return 0 if successful, otherwise return err code.

Description:

Function ost_handle_quota_adjust_qunit will call this function, which will change local qunit size and allocate/release the quota after that.

1.4.5 dquot_adjust_qunit_size(new function)

int dquot_adjust_qunit_size(struct lustre_quota_ctxt *qctxt, struct lustre_dquot *dquot, __u32 ost_num, __u32 mdt_num, int type)

Parameters:

struct lustre_quota_ctxt *qctxt: The quota's context in the slaves. We can get original qunit size from it.

struct lustre_dquot *dquot: the current qunit size information is recorded in this structure.

__u32 ost_num: the number of ost slaves this master uses.

__u32 mdt_num: the number of mds slaves this master uses. This value is usually 1.

int type: instruct this function to adjust the block qunit size, inode qunit size or both of them.

Return value:

return 0 if qunit size don't need to be changed; return negative value if any error happens; return the value more than zero if qunit size has been changed.

Description:

when a user/group initializes quota or quota is allocated/released, this function will be called in a master. This function just checks whether qunit size needs to be changed. If qunit size needs to be changed, the new value is recorded in dquot.

1.4.6 quota_adjust_slave_qs(new function)

int quota_adjust_slave_qs(struct obd_quota_adjust_qunit *oqaq, struct lustre_quota_ctxt *qctxt)

Parameters:

struct obd_quota_adjust_qunit *oqaq: the enlarge/shrink qunit size information the master sent is included in this structure.

struct lustre_quota_ctxt *qctxt: the qdata context in a slave.

Return value:

return 0 if successful, otherwise return err code.

Description:

when a master finds that it is necessary to shrink qunit size, it will send a RPC(OST_QUOTA_ADJUST_QUNIT) to all of the corresponding slaves. Then the slaves call this function to change their own qunit size.

1.4.7 mds_set_dqblk

int mds_set_dqblk(struct obd_device *obd, struct obd_quotactl *oqctl)

Parameters:

struct obd_device *obd: the corresponding structure mds_obd information gets from it.

struct obd_quotactl *oqctl: the information of the user setting quota is included in this.

Return value:

return 0 if successful, otherwise return err code.

Description:

when a user sets block/inode quota on a user/group, this function will be called. When it finds that the quota limitation is too small, it will change qunit size. This function will call function mds_init_slave_blimits and function mds_init_slave_ilimits to demand the corresponding slaves to change their own qunit size.

1.4.8 target_handle_dqacq_callback

int target_handle_dqacq_callback(struct ptlrpc_request *req)

Parameters:

struct ptlrpc_request *req: it is a allocate/release quota request from a slave.

Return value:
return 0 if successful, otherwise return err code.
Description:
when a slave sends a quota request(allocate/release) to a master, this function handles the request. In this function, I have added code so that when qunit size is needed to be shrank, a master records the new qunit size in dquot and sends the command(OST_QUOTA_ADJUST_QUNIT) to all the corresponding slaves; if qunit size needs to increase, it will inform the slave by the reply.

2 Use Cases

2.1 Adjusting block qunit size when a user sets small quota limit

1. if (quota limit is less than shrink_qunit_limit * ost num * LEAST_OF_BLOCK_QUNIT_SIZE/LEAST
   Lustre will go into LESSBUF state and goto step 4.
2. the master reduces qunit size to half of itself.
3. if ((quota limit is less than CP) && (qunit size is more than LEAST_OF_BLOCK_QUNIT_SIZE)), do step 2 again. Otherwise, go to step 4.
4. the master informs all slaves to change their own qunit size(function mds_set_dqblk -> mds_init_slave_blimits -> lov_quota_adjust_qunit -> client_quota_adjust_qunit).
5. slaves receive a request to shrink their own qunit size(function ost_handle_quota_adjust_qunit -> filter_quota_adjust_qunit -> quota_adjust_slave_qs).
6. slaves adjust their own quota with new smaller qunit size after shrinking(function filter_quota_adjust_qunit -> qctxt_adjust_qunit). In this case, slaves will almost release quota from a master.

2.2 Adjusting block qunit size when a slave allocates quota

1. after a slave allocates some quota, if (quota limit is less than shrink_qunit_limit * ost num * LEAST_OF_BLOCK_QUNIT_SIZE/LEAST_OF_INODE_QUNIT_SIZE), Lustre will go into LESSBUF state and goto step 5.
2. after a slave allocates some quota, if ((quota limit is less than CP) && (qunit size is more than original block LEAST_OF_BLOCK_QUNIT_SIZE)), the master decreases qunit size to half of itself.
3. if ((quota limit is less than CP) && (qunit size is more than original block LEAST_OF_BLOCK_QUNIT_SIZE)), do step 2 again. Otherwise, go to step 4.

4. if qunit size is less than LEAST_OF_BLOCK_QUNIT_SIZE, adjusting finishes and exit(steps from 1 to 4 are done in function target_handle_dqacq_callback -> dquot_adjust_qunit_size).

5. the master informs all slaves to change their own qunit size(function target_handle_dqacq_callback -> lov_quota_adjust_qunit -> client_quota_adjust_qunit).

6. slaves receive a request to shrink their qunit size(function ost_handle_quota_adjust_qunit -> filter_quota_adjust_qunit -> quota_adjust_slave_qs).

7. slaves adjust their own quota with new smaller qunit size after shrinking(function filter_quota_adjust_qunit -> qctxt_adjust_qunit). In this case, slaves will almost release quota from a master.

2.3 Adjusting block qunit size when a slave releases quota

1. a slave releases some quota.

2. if ((quota limit is more than 2 * CP) && (qunit size is less than original block qunit size)), the master increases qunit size to twice of itself;

3. if ((quota limit is more than 2 * CP) && (qunit size is less than original block qunit size)), do step 2 again. During the procedure, if (remaining quota > 8 * shrink_qunit_limit * ost num * LEAST_OF_BLOCK_QUNIT_SIZE/LEAST_OF_INOD && (lustre is in LESSBUF state), lustre goes into FULLBUF state.

4. if qunit size is more than its original block qunit size, qunit size equals its original qunit size. (steps from 1 to 4 are done in function target_handle_dqacq_callback -> dquot_adjust_qunit_size).

5. the master informs active slaves to change their own qunit size by a reply(function target_handle_dqacq_callback -> lov_quota_adjust_qunit -> client_quota_adjust_qunit).

6. slaves receive the reply to enlarge their own qunit size(function ost_handle_quota_adjust_qunit -> filter_quota_adjust_qunit -> quota_adjust_slave_qs).

7. slaves adjust their own quota with new larger qunit_size after enlarging(function filter_quota_adjust_qunit -> qctxt_adjust_qunit). In this case, slaves will almost get quota from a master.

3 Logic Specification

3.1 lov_quota_adjust_qunit(new function)

```
int lov_quota_adjust_qunit(struct obd_export *exp, struct
                           obd_quota_adjust_qunit *oqaq)
{
    struct obd_device *obd = class_exp2obd(exp);
    struct lov_obd *lov = &obd->u.lov;
    int i, rc = 0;
    ENTRY;
    if (!(oqaq->qaq_flags & (ADJUST_BLOCK_QUNIT_SIZE |
        ADJUST_INODE_QUNIT_SIZE)))
    {
        CERROR("bad qaq_flags %x for lov obd", oqaq->qaq_flags);
        RETURN(-EFAULT);
    }
    for (i = 0; i < lov->desc.ld_tgt_count; i++) {
        int err;
        if (!lov->lov_tgts[i] || !lov->lov_tgts[i]->ltd_active) {
            CDEBUG(D_HA, "ost %d is inactive\n", i);
            continue;
         }
         err = obd_quota_adjust_qunit(lov->lov_tgts[i]->ltd_exp, oqaq);
         if (err) {
            if (lov->lov_tgts[i]->ltd_active && !rc)
                rc = err;
            continue;
         }
    }
    RETURN(rc);
}
```

3.2 client_quota_adjust_qunit(new function)

```
int client_quota_adjust_qunit(struct obd_export *exp, struct
                              obd_adjust_qunit *oqaq)
{
   struct ptlrpc_request *req;
   struct obd_adjust_qunit *oqa;
   int size[2] = { sizeof(struct ptlrpc_body), sizeof(*oqaq) };
   int ver, opc, rc = 0;
   ENTRY;

   /* client don't support this kind of operation, abort it */
   if (!(exp->exp_connect_flags & OBD_CONNECT_CHANGE_QS))
       RETURN(rc);
   if (!strcmp(exp->exp_obd->obd_type->typ_name, LUSTRE_OSC_NAME)) {
      ver = LUSTRE_OST_VERSION,
      opc = OST_QUOTA_ADJUST_QUNIT;
   } else {
      RETURN(-EINVAL);
```

```
    }
    req = ptlrpc_prep_req(class_exp2cliimp(exp), ver, opc, 2, size, NULL);
    if (!req) {
        ptlrpc_req_finished(req);
        GOTO(out, rc = -ENOMEM);
    }
    oqa = lustre_msg_buf(req->rq_reqmsg, REQ_REC_OFF, sizeof(*oqaq));
    *oqa = *oqaq;
    ptlrpc_req_set_repsize(req, 2, size);
    rc = ptlrpcd_add_req(req);
  out:
    RETURN (rc);
}
```

3.3 ost_handle_quota_adjust_qunit(new function)

```
static int ost_handle_quota_adjust_qunit(struct ptlrpc_request *req)
{
        struct obd_quota_adjust_qunit *oqaq, *repoqa;
        int rc, size[2] = { sizeof(struct ptlrpc_body), sizeof(*repoqa) };
        ENTRY;
        oqaq = lustre_swab_reqbuf(req, REQ_REC_OFF, sizeof(*oqaq),
                                  lustre_swab_obd_quota_adjust_qunit);
        if (oqaq == NULL)
                GOTO(out, rc = -EPROTO);
        rc = lustre_pack_reply(req, 2, size, NULL);
        if (rc)
                GOTO(out, rc);
        repoqa = lustre_msg_buf(req->rq_repmsg, REPLY_REC_OFF, sizeof(*repoqa));
        req->rq_status = obd_quota_adjust_qunit(req->rq_export, oqaq);
        *repoqa = *oqaq;
out:
        RETURN(rc);
}
```

3.4 filter_quota_adjust_qunit(new function)

```
int filter_quota_adjust_qunit(struct obd_export *exp, struct
                              obd_quota_adjust_qunit *oqaq)
{
        struct obd_device *obd = exp->exp_obd;
        struct lustre_quota_ctxt *qctxt = &obd->u.obt.obt_qctxt;
        unsigned int uid = 0, gid = 0;
        int rc = 0;
        ENTRY;

        LASSERT(oqaq);
        LASSERT(oqaq->qaq_flags & ADJUST_BLOCK_QUNIT_SIZE);
        rc = quota_adjust_slave_qs(oqaq, qctxt);
        if (rc < 0) {
            CERROR("adjust mds slave's qunit size failed!
                   (rc:%d)\n", rc);
            RETURN(rc);
        }
        if (oqaq->qaq_type & QUOTA_IS_GRP)
            gid = oqaq->qaq_id;
        else
            uid = oqaq->qaq_id;
        rc = qctxt_adjust_qunit(obd, qctxt, uid, gid, 1, 0);
        if (rc)
            CERROR("error slave adjust local file quota! (rc:%d)\n", rc);

        RETURN(rc);
}
```

3.5 quota_adjust_slave_qs(new function)

```
int quota_adjust_slave_qs(struct obd_quota_adjust_qunit *oqaq, struct
                          lustre_quota_ctxt *qctxt)
{
   struct lustre_qunit_size *lqs;
   unsigned int hashent = qs_hashfn(qctxt, oqaq->qaq_id, oqaq->qaq_type);
   ENTRY;

   spin_lock(&qctxt->lqc_lqs_lock);
   lqs = find_qs_nolock(hashent, qctxt, oqaq->qaq_id, oqaq->qaq_type);
   if (!lqs) {
        lqs = alloc_qs(qctxt, oqaq->qaq_id, oqaq->qaq_type);
        if (!lqs) {
              CERROR("can't find the lustre_qunit_size!");
              spin_unlock(&qctxt->lqc_lqs_lock);
              RETURN(-ENOMEM);
        }
        insert_qs_nolock(qctxt, lqs);
   }

   /* adjust the slave's block qunit size */
   if ((oqaq->qaq_flags & ADJUST_BLOCK_QUNIT_SIZE) &&
        lqs->lqs_bunit_sz != oqaq->qaq_bunit_sz) {
                if (test_bit(DQ_BLKLESSBUF_B, oqaq->qaq_flags)) {
                    if (!test_bit(DQ_BLKLESSBUF_B,lqs->flags)) {
                           set_bit(DQ_BLKLESSBUF_B,lqs->flags);
                           lqs->lqs_bunit_sz = LEAST_OF_BLOCK_QUNIT_SIZE;
                           lqs->lqs_btune_sz = lqs->lqs_bunit_sz * qctxt->lqc_btune_sz / qctxt-
                    }
                } else {
                    if (test_bit(DQ_BLKLESSBUF_B,lqs->flags))
                           clear_bit(DQ_BLKLESSBUF_B,lqs->flags);
                    lqs->lqs_bunit_sz = oqaq->qaq_bunit_sz;
                    lqs->lqs_btune_sz = lqs->lqs_bunit_sz * qctxt->lqc_btune_sz / qctxt->lqc_bu
                }
           }

           /* adjust the slave's file qunit size */
           if ((oqaq->qaq_flags & ADJUST_INODE_QUNIT_SIZE) &&
                lqs->lqs_iunit_sz != oqaq->qaq_iunit_sz) {
                if (test_bit(DQ_INOLESSBUF_B, oqaq->qaq_flags)) {
                    if (!test_bit(DQ_INOLESSBUF_B,lqs->flags)) {
                           set_bit(DQ_INOLESSBUF_B,lqs->flags);
                           lqs->lqs_iunit_sz = LEAST_OF_INODE_QUNIT_SIZE;
                           lqs->lqs_itune_sz = lqs->lqs_iunit_sz * qctxt->lqc_itune_sz / qctxt-
                    }
                } else {
                    if (test_bit(DQ_INOLESSBUF_B,lqs->flags))
                           clear_bit(DQ_INOLESSBUF_B,lqs->flags);
                    lqs->lqs_iunit_sz = oqaq->qaq_iunit_sz;
                    lqs->lqs_itune_sz = lqs->lqs_iunit_sz * qctxt->lqc_itune_sz / qctxt->lqc_iu
                }
           }

           spin_unlock(&qctxt->lqc_lqs_lock);
           RETURN(0);
        }
```

3.6 dquot_adjust_qunit_size(new function)

```
        /* when entering this function, dquot->dq_sem must be down */
        int dquot_adjust_qunit_size(struct lustre_quota_ctxt *qctxt, struct lustre_dquot *dquot
        {
           __u32 ihardlimit, bhardlimit, curinodes;
           unsigned long bunit_orig, *bunit_curr, iunit_orig, *iunit_curr;
           __u64 curspace;
           int rc = 0;
           ENTRY;
        if (!dquot)
             RETURN(-EINVAL);

        /* don't change qunit size */
        if (!enable_modify_qunit_size)
             RETURN(rc);
        ihardlimit = dquot->dq_dqb.dqb_ihardlimit;
        bhardlimit = dquot->dq_dqb.dqb_bhardlimit;
        curinodes  = dquot->dq_dqb.dqb_curinodes;
        curspace   = dquot->dq_dqb.dqb_curspace;
        bunit_orig = qctxt->lqc_bunit_sz;
        bunit_curr = &dquot->dq_bunit_sz_curr;
        iunit_orig = qctxt->lqc_iunit_sz;
        iunit_curr = &dquot->dq_iunit_sz_curr;

        if (type & ADJUST_BLOCK_QUNIT_SIZE) {
            /* shrink block qunit size */
            while (bhardlimit && (*bunit_curr) / SHRINK_QUNIT_FACTOR >=
                   LEAST_OF_BLOCK_QUNIT_SIZE && (bhardlimit - curspace) <
                   (*bunit_curr) * ost_num * shrink_qunit_limit) {
                          *bunit_curr = QUSG((*bunit_curr) /
                                        SHRINK_QUNIT_FACTOR, 1) <<
                                        QUOTABLOCK_BITS;
                          rc |= ADJUST_BLOCK_QUNIT_SIZE;
            }

            /* enlarge block qunit size */
            while (bhardlimit && (*bunit_curr < bunit_orig) && (bhardlimit -
                   curspace) > 2 *(*bunit_curr) * ost_num * shrink_qunit_limit) {
                    if ((*bunit_curr) * SHRINK_QUNIT_FACTOR < bunit_orig)
                        *bunit_curr = QUSG((*bunit_curr) * SHRINK_QUNIT_FACTOR,
                                      1) << QUOTABLOCK_BITS;
                    else
                        *bunit_curr = bunit_orig;
            }
        }

        if (type & ADJUST_INODE_QUNIT_SIZE) {
            /* shrink file qunit size */
            while (ihardlimit && (*iunit_curr) / SHRINK_QUNIT_FACTOR >=
                   LEAST_OF_INODE_QUNIT_SIZE && (ihardlimit - curinodes) <
                   (*iunit_curr) * mdt_num * shrink_qunit_limit)    {
                          *iunit_curr = (*iunit_curr) / SHRINK_QUNIT_FACTOR;
                          rc |= ADJUST_INODE_QUNIT_SIZE;
            }
            /* enlarge file qunit size */
            while (ihardlimit && (*iunit_curr < iunit_orig) && (ihardlimit -
                   curinodes) > 2 * (*iunit_curr) * mdt_num *
                   shrink_qunit_limit) {
                         if ((*iunit_curr) * SHRINK_QUNIT_FACTOR < iunit_orig)
                             *iunit_curr = (*iunit_curr) * SHRINK_QUNIT_FACTOR;
                         else
                             *iunit_curr = iunit_orig;
            }
        }

        RETURN(rc);
    }
```

3.7 mds_set_dqblk

```
    int mds_set_dqblk(struct obd_device *obd, struct obd_quotactl *oqctl)
    {
        struct mds_obd *mds = &obd->u.mds;
    +   struct lustre_quota_ctxt *qctxt = &mds->mds_obt.obt_qctxt;
    +   struct obd_device *obd = class_exp2obd(mds->mds_osc_exp);
    +   struct lov_obd *lov = &obd->u.lov;
    +   __u32 ost_num = lov->desc.ld_tgt_count, mdt_num = 1;
    +   struct  obd_quota_adjust_qunit *oqaq = NULL;
    +   int rc1 = 0;
        ... ...
        dquot->dq_status |= DQ_STATUS_SET;
        ihardlimit = dquot->dq_dqb.dqb_ihardlimit;
        isoftlimit = dquot->dq_dqb.dqb_isoftlimit;
        bhardlimit = dquot->dq_dqb.dqb_bhardlimit;
        bsoftlimit = dquot->dq_dqb.dqb_bsoftlimit;
        ... ...
        if (dqblk->dqb_valid & QIF_BLIMITS) {
             dquot->dq_dqb.dqb_bhardlimit = dqblk->dqb_bhardlimit;
             dquot->dq_dqb.dqb_bsoftlimit = dqblk->dqb_bsoftlimit;
             /* clear usage (limit pool) */
             if (!dquot->dq_dqb.dqb_bhardlimit && !dquot->dq_dqb.dqb_bsoftlimit)
                     dquot->dq_dqb.dqb_curspace = 0;
                     /* clear grace time */
                     if (!dqblk->dqb_bsoftlimit ||
                         toqb(dquot->dq_dqb.dqb_curspace) <=
                              dqblk->dqb_bsoftlimit)
                         dquot->dq_dqb.dqb_btime = 0;
                         /* set grace only if user hasn't provided his own */
                     else if (!(dqblk->dqb_valid & QIF_BTIME))
```

```
                            dquot->dq_dqb.dqb_btime = cfs_time_current_sec() +
                                    qinfo->qi_info[dquot->dq_type].dqi_bgrace;
+               rc1 = dquot_adjust_qunit_size(qctxt, dquot, ost_num, 1,
                                                    ADJUST_BLOCK_QUNIT_SIZE);
+               if (rc1 < 0)
+                       CERROR("adjust block qunit size failed! (rc:%d)\n",
                                rc);
+               if (rc1 > 0 && (rc1 & ADJUST_BLOCK_QUNIT_SIZE)) {
+                       OBD_ALLOC_PTR(oqaq);
+                       oqaq->qaq_bunit_sz = dquot->dq_bunit_sz_curr;
+                       oqaq->qaq_flags |= ADJUST_BLOCK_QUNIT_SIZE;
+                       if (dquot->dq_dqb.dqb_curspace >= LEAST_OF_BLOCK_QUNIT_SIZE * ost
+                            shrink_qunit_limit) {
+                               clear_bit(DQ_BLKLESSBUF_B,dquot->dq_flags);
+                               clear_bit(DQ_BLKLESSBUF_B,oqaq->flags);
+                       } else {
+                               set_bit(DQ_BLKLESSBUF_B,dquot->dq_flags);
+                               set_bit(DQ_BLKLESSBUF_B,oqaq->flags);
+                       }
+                }
        }
        if (dqblk->dqb_valid & QIF_ILIMITS) {
                dquot->dq_dqb.dqb_ihardlimit = dqblk->dqb_ihardlimit;
                dquot->dq_dqb.dqb_isoftlimit = dqblk->dqb_isoftlimit;
                /* clear usage (limit pool) */
                if (!dquot->dq_dqb.dqb_ihardlimit &&
                    !dquot->dq_dqb.dqb_isoftlimit)
                        dquot->dq_dqb.dqb_curinodes = 0;
                if (!dqblk->dqb_isoftlimit ||
                    dquot->dq_dqb.dqb_curinodes <= dqblk->dqb_isoftlimit)
                        dquot->dq_dqb.dqb_itime = 0;
                else if (!(dqblk->dqb_valid & QIF_ITIME))
                        dquot->dq_dqb.dqb_itime = cfs_time_current_sec() +
                                qinfo->qi_info[dquot->dq_type].dqi_igrace;
+               rc1 = dquot_adjust_qunit_size(qctxt, dquot, mdt_num, 1,
                                                    ADJUST_INODE_QUNIT_SIZE);
+               if (rc1 < 0)
+                       CERROR("adjust inode qunit size failed! (rc:%d)\n", rc);
+               if (rc1 > 0 && (rc1 & ADJUST_INODE_QUNIT_SIZE)) {
+                       if (!oqaq)
+                            OBD_ALLOC_PTR(oqaq);
+                       oqaq->qaq_iunit_sz = dquot->dq_iunit_sz_curr;
+                       oqaq->qaq_flags |= ADJUST_INODE_QUNIT_SIZE;
+                       if (dquot->dq_dqb.dqb_curinodes >= LEAST_OF_INODE_QUNIT_SIZE * md
+                           shrink_qunit_limit) {
+                               clear_bit(DQ_INOLESSBUF_B,dquot->dq_flags);
+                               clear_bit(DQ_INOLESSBUF_B,oqaq->flags);
+                       } else {
+                               set_bit(DQ_INOLESSBUF_B,dquot->dq_flags);
+                               set_bit(DQ_INOLESSBUF_B,oqaq->flags);
+                       }
+                  }
        }
   ... ...
        up(&mds->mds_qonoff_sem);
        if (dqblk->dqb_valid & QIF_ILIMITS) {
                set = !(ihardlimit || isoftlimit);
-               rc = mds_init_slave_ilimits(obd, oqctl, set);
+               rc = mds_init_slave_ilimits(obd, oqctl, set, oqaq);
                if (rc) {
                        CERROR("init slave ilimits failed! (rc:%d)\n", rc);
                        goto revoke_out;
                }
        }
        if (dqblk->dqb_valid & QIF_BLIMITS) {
                set = !(bhardlimit || bsoftlimit);
-               rc = mds_init_slave_blimits(obd, oqctl, set);
+               rc = mds_init_slave_blimits(obd, oqctl, set, oqaq);
                if (rc) {
                        CERROR("init slave blimits failed! (rc:%d)\n", rc);
                        goto revoke_out;
                }
        }
+       if (oqaq)
+               OBD_FREE_PTR(oqaq);
        down(&mds->mds_qonoff_sem);
  ... ...

}
```

3.8 mds_init_slave_blimits

```
static int mds_init_slave_blimits(struct obd_device *obd,
                                  struct obd_quotactl *oqctl, int set,
                                  struct obd_quota_adjust_qunit *oqaq){
+    struct obd_device_target *obt = &obd->u.obt;
+    struct lustre_quota_ctxt *qctxt = &obt->obt_qctxt;
     ... ...
     ioqc->qc_cmd = Q_INITQUOTA;
     ioqc->qc_id = oqctl->qc_id;
     ioqc->qc_type = oqctl->qc_type;
```

```
          ioqc->qc_dqblk.dqb_valid = QIF_BLIMITS;
          ioqc->qc_dqblk.dqb_bhardlimit = set ? MIN_QLIMIT : 0;
+         if (oqaq->qaq_flags & ADJUST_BLOCK_QUNIT_SIZE) {
              /* adjust the mds slave's block qunit_size */
+             rc = quota_adjust_slave_qs(oqaq, qctxt);
+             if (rc < 0)
+                CERROR("adjust mds slave's block qunit size failed! (rc:%d)\n", rc);
+         }
          ... ...
          /* initialize all slave's limit */
          rc = obd_quotactl(mds->mds_osc_exp, ioqc);

+         /* adjust all slave's qunit size */
+         if (oqaq && (oqaq->qaq_flags & ADJUST_BLOCK_QUNIT_SIZE))
+              rc = obd_quota_adjust_qunit(mds->mds_osc_exp, oqaq);
          EXIT;
     out:
          OBD_FREE_PTR(ioqc);
          return rc;
     }
```

3.9 mds_init_slave_ilimits

```
     static int mds_init_slave_ilimits(struct obd_device *obd,
                                       struct obd_quotactl *oqctl, int set,
                                       struct obd_quota_adjust_qunit *oqaq){
     +  struct obd_device_target *obt = &obd->u.obt;
     +  struct lustre_quota_ctxt *qctxt = &obt->obt_qctxt;
        ... ...
        ioqc->qc_cmd = Q_INITQUOTA;
        ioqc->qc_id = oqctl->qc_id;
        ioqc->qc_type = oqctl->qc_type;
        ioqc->qc_dqblk.dqb_valid = QIF_ILIMITS;
        ioqc->qc_dqblk.dqb_ihardlimit = MIN_QLIMIT;

     +  if (oqaq->qaq_flags & ADJUST_INODE_QUNIT_SIZE) {
            /* adjust the mds slave's inode qunit_size */
     +      rc = quota_adjust_slave_qs(oqaq, qctxt);
     +      if (rc < 0)
     +         CERROR("adjust mds slave's inode qunit size failed! (rc:%d)\n",
                      rc);
     +  }
        ... ...
     }
```

3.10 target_handle_dqacq_callback

```
int target_handle_dqacq_callback(struct ptlrpc_request *req)
{
#ifdef __KERNEL__
   struct obd_device *obd = req->rq_export->exp_obd;
   struct obd_device *master_obd = obd->obd_observer->obd_observer;
+  struct mds_obd *mds = &master_obd->u.mds;
+  struct obd_device *lov_mds_obd = class_exp2obd(mds->mds_osc_exp);
+  struct lov_obd *lov = &lov_mds_obd->u.lov;
+  __u32 ost_num = lov->desc.ld_tgt_count, mdt_num = 1;
+  struct obd_quota_adjust_quota *oqaq;
+  struct lustre_dquot *dquot = NULL;
+  unsigned int uid = 0, gid = 0;
+  __u32 is_blk = 0;
+  __u32 qdata_type = 0;
+  int rc1 = 0;
   ... ...
   if (qdata == NULL) {
                CERROR("Can't unpack qunit_data\n");
                RETURN(-EPROTO);
   }

+  is_blk = (qdata->qd_flags & QUOTA_IS_BLOCK) >> 1;
+  qdata_type = qdata->qd_flags & QUOTA_IS_GRP;
+  dquot = lustre_dqget(master_obd, &mds->mds_quota_info, qdata->qd_id, qdata_type);
+  if (IS_ERR(dquot))
+               RETURN(PTR_ERR(dquot));
   /* we use the observer */
   LASSERT(obd->obd_observer && obd->obd_observer->obd_observer);
   master_obd = obd->obd_observer->obd_observer;
   qctxt = &master_obd->u.obt.obt_qctxt;
   ... ...
   req->rq_status = rc;
   rc = ptlrpc_reply(req);

+  down(&dquot->dq_sem);
+  if (is_blk)
+         rc1 = dquot_adjust_qunit_size(qctxt, dquot, ost_num, 1,
+                                        ADJUST_BLOCK_QUNIT_SIZE);
+  else
+         rc1 = dquot_adjust_qunit_size(qctxt, dquot, ost_num, 1,
+                                        ADJUST_INODE_QUNIT_SIZE);
+  if (rc1 < 0)
+         CERROR("adjust qunit size failed! (rc:%d)\n", rc);
```

```
+
+  if (rc1 > 0) {
+     OBD_ALLOC_PTR(oqaq);
+     if (!oqaq)
+           RETURN(-ENOMEM);

+     if (is_blk)
+             oqaq->qaq_flags = ADJUST_BLOCK_QUNIT_SIZE;
+     else
+             oqaq->qaq_flags = ADJUST_INODE_QUNIT_SIZE;
+     oqaq->qaq_id = qdata->qd_id;
+     oqaq->qaq_type = qdata_type;
+     if (rc1 == ADJUST_BLOCK_QUNIT_SIZE)
+           oqaq->qaq_bunit_sz = dquot->dq_bunit_sz_curr;
+     if (rc1 == ADJUST_INODE_QUNIT_SIZE)
+           oqaq->qaq_iunit_sz = dquot->dq_iunit_sz_curr;
+
+     /* adjust the mds slave qunit size */
+     rc = quota_adjust_slave_qs(oqaq, qctxt);
+     if (rc < 0)
+           CERROR("adjust mds slave's qunit size failed!
                  (rc:%d)\n", rc);
+     if (qdata_type)
+           gid = qdata->qd_id;
+     else
+           uid = qdata->qd_id;
+     rc = qctxt_adjust_qunit(master_obd, qctxt, uid, gid, is_blk, 0);
+     if (rc)
+           CERROR("error mds adjust local file quota! (rc:%d)\n", rc);
+
+     /* adjust all ost slave's limit */
+     if (rc1 & ADJUST_BLOCK_QUNIT_SIZE)
+           rc = obd_quota_adjust_quint(mds->mds_osc_exp, oqaq);
+
+     if ((rc1 & ADJUST_BLOCK_QUNIT_SIZE) && (dquot->dq_dqb.dqb_curspace >= LEAST_OF_BL
+              if (test_bit(DQ_BLKLESSBUF_B,dquot->dq_flags)) {
+                    if (dquot->dq_dqb.dqb_curspace >= 8 * LEAST_OF_BLOCK_QUNIT_SIZE *
+                        shrink_qunit_limit||dquot->dq_bunit_sz_curr >= qctxt->lqc_buni
+                        clear_bit(DQ_BLKLESSBUF_B,dquot->dq_flags);
+              }
+     } else {
+              if (!test_bit(DQ_BLKLESSBUF_B,dquot->dq_flags))
+                    set_bit(DQ_BLKLESSBUF_B,dquot->dq_flags);
+     }
+
+     if ((rc1 & ADJUST_INODE_QUNIT_SIZE) && (dquot->dq_dqb.dqb_curinodes >= LEAST_OF_I
+               if (test_bit(DQ_INOLESSBUF_B,dquot->dq_flags)) {
+                     if (dquot->dq_dqb.dqb_curinodes >= 8 * LEAST_OF_INODE_QUNIT_SIZE *
+                         shrink_qunit_limit|| dquot->dq_iunit_sz_curr >= qctxt->lqc_iun
+                         clear_bit(DQ_INOLESSBUF_B,dquot->dq_flags);
+               }
+      } else {
+               if (!test_bit(DQ_INOLESSBUF_B,dquot->dq_flags))
+                     set_bit(DQ_INOLESSBUF_B,dquot->dq_flags);
+      }
+   }
+   up(&dquot->dq_sem);
+   lustre_dqput(dquot);

    /* the qd_count might be changed in lqc_handler */
    if ((req->rq_export->exp_connect_flags & OBD_CONNECT_QUOTA64) &&
         !OBD_FAIL_CHECK(OBD_FAIL_QUOTA_QD_COUNT_32BIT)) {
+           if (req->rq_export->exp_connect_flags & OBD_CONNECT_CHANGE_QS)
+                    qdata->qd_flags |= dquot->dq_flags & DQ_BLKLESSBUF_B;
            memcpy(rep,qdata,sizeof(*qdata));
    } else {
            qdata_old = lustre_quota_new_to_old(qdata);
            memcpy(rep,qdata_old,sizeof(*qdata_old));
    }
    req->rq_status = rc;
    rc = ptlrpc_reply(req);
    RETURN(rc);
    ... ...
}
```

4 Environment

4.1 Network Protocol Compatibility - initialization

The master and slaves will set OBD_CONNECT_CHANGE_QS bit on its export/import during the initialization. Later, a mater/slave will use different structure by checking this bit. The procedure of setting this bit is below:

1. At the beginning, a mds(master) will connect to an ost(slave). It will call mds_lov_connect, which will add OBD_CONNECT_CHANGE_QS to data->ocd_connect_flags. this data will be transferred to the ost(slave).

2. The ost receives the data(function filter_connect_internal). If it finds the data sets the OBD_CONNECT_CHANGE_QS bit, it will set its corresponding OBD_CONNECT_CHANGE_QS bit of export, and send it back.

3. The mds receives the reply(function ptlrpc_connect_interpret). If it finds the OBD_CONNECT_CHANGE_QS bit of data has been set, it will set the OBD_CONNECT_CHANGE_QS bit of relative import and export of this connection.

4.2 Network Protocol Compatibility - New slaves, Old master

Because a old master will never send a command of shrinking/enlarging the qunit size, the relative code in a new slave will never be executed.

4.3 Network Protocol Compatibility - New master, Old slaves

when a master needs to send a command of shrinking/enlarging the qunit size to a slave, it will check OBD_CONNECT_CHANGE_QS bit. If it finds this bit is set, it will send the request ; if not, it won't send the command. Because old slaves don't know how to shrink/enlarge qunit size, the new master will do in the old way(using unique qunit size for old slaves).

4.4 Documentation Changes

Adding two proc entry: shrink_qunit_limit, which can be modified by the user and should be declared in the user document; enable_modify_qunit_size, which instruct the master does shrinking/enlarging qunit size or doesn't.

5 Test plan

5.1 Unit test

I will write the test programs for "Describe the problem" and "Use cases" to test the correctness of this change and add them to sanity-quota.sh.

5.2 System test

I will run sanity-quota.sh for system test.

5.3 Integrate test

I will ran sanity.sh for integrate test.