[PATCH 05/11] quota: Add quota reservation support

Source: http://linux.derkeiler.com/Mailing−Lists/Kernel/2009−01/msg08031.html

- *From*: Jan Kara <jack@xxxxxxx>
- *Date*: Fri, 16 Jan 2009 19:08:13 +0100

From: Mingming Cao <cmm@xxxxxxxxxx>

Delayed allocation defers the block allocation at the dirty pages
flush−out time, doing quota charge/check at that time is too late.
But we can't charge the quota blocks until blocks are really allocated,
otherwise users could get overcharged after reboot from system crash.

This patch adds quota reservation for delayed allocation. Quota blocks
are reserved in memory, inode and quota won't gets dirtied until later
block allocation time.

Signed−off−by: Mingming Cao <cmm@xxxxxxxxxx>
Signed−off−by: Jan Kara <jack@xxxxxxx>
−−−
fs/dquot.c | 117 ++++++++++++++++++++++++++++++++−−−−−−−−−−−−
include/linux/quota.h | 3 +
include/linux/quotaops.h | 21 ++++++++
3 files changed, 110 insertions(+), 31 deletions(−)

diff −−git a/fs/dquot.c b/fs/dquot.c
index bca3cac..9b1c4d3 100644
−−− a/fs/dquot.c
+++ b/fs/dquot.c
@@ −899,6 +899,11 @@ static inline void dquot_incr_space(struct dquot *dquot, qsize_t number)
dquot−>dq_dqb.dqb_curspace += number;
}

+static inline void dquot_resv_space(struct dquot *dquot, qsize_t number)
+{
+ dquot−>dq_dqb.dqb_rsvspace += number;
+}
+
static inline void dquot_decr_inodes(struct dquot *dquot, qsize_t number)
{
if (sb_dqopt(dquot−>dq_sb)−>flags & DQUOT_NEGATIVE_USAGE ||
@@ −1068,7 +1073,11 @@ err_out:
kfree_skb(skb);
}

```
#endif
−
+/*
+ * Write warnings to the console and send warning messages over netlink.
+ *
+ * Note that this function can sleep.
+ */
static inline void flush_warnings(struct dquot * const *dquots, char *warntype)
{
int i;
@@ −1129,13 +1138,18 @@ static int check_idq(struct dquot *dquot, qsize_t inodes, char *warntype)
/* needs dq_data_lock */
static int check_bdq(struct dquot *dquot, qsize_t space, int prealloc, char *warntype)
{
+ qsize_t tspace;
+
*warntype = QUOTA_NL_NOWARN;
if (!sb_has_quota_limits_enabled(dquot−>dq_sb, dquot−>dq_type) ||
test_bit(DQ_FAKE_B, &dquot−>dq_flags))
return QUOTA_OK;

+ tspace = dquot−>dq_dqb.dqb_curspace + dquot−>dq_dqb.dqb_rsvspace
+ + space;
+
if (dquot−>dq_dqb.dqb_bhardlimit &&
− dquot−>dq_dqb.dqb_curspace + space > dquot−>dq_dqb.dqb_bhardlimit &&
+ tspace > dquot−>dq_dqb.dqb_bhardlimit &&
!ignore_hardlimit(dquot)) {
if (!prealloc)
*warntype = QUOTA_NL_BHARDWARN;
@@ −1143,7 +1157,7 @@ static int check_bdq(struct dquot *dquot, qsize_t space, int prealloc, char *war
}

if (dquot−>dq_dqb.dqb_bsoftlimit &&
− dquot−>dq_dqb.dqb_curspace + space > dquot−>dq_dqb.dqb_bsoftlimit &&
+ tspace > dquot−>dq_dqb.dqb_bsoftlimit &&
dquot−>dq_dqb.dqb_btime && get_seconds() >= dquot−>dq_dqb.dqb_btime &&
!ignore_hardlimit(dquot)) {
if (!prealloc)
@@ −1152,7 +1166,7 @@ static int check_bdq(struct dquot *dquot, qsize_t space, int prealloc, char *war
}

if (dquot−>dq_dqb.dqb_bsoftlimit &&
− dquot−>dq_dqb.dqb_curspace + space > dquot−>dq_dqb.dqb_bsoftlimit &&
+ tspace > dquot−>dq_dqb.dqb_bsoftlimit &&
dquot−>dq_dqb.dqb_btime == 0) {
if (!prealloc) {
*warntype = QUOTA_NL_BSOFTWARN;
@@ −1306,51 +1320,92 @@ void vfs_dq_drop(struct inode *inode)
/*
* This operation can block, but only after everything is updated
```

```
*/
−int dquot_alloc_space(struct inode *inode, qsize_t number, int warn)
+int __dquot_alloc_space(struct inode *inode, qsize_t number,
+ int warn, int reserve)
{
− int cnt, ret = NO_QUOTA;
+ int cnt, ret = QUOTA_OK;
char warntype[MAXQUOTAS];

− /* First test before acquiring mutex − solves deadlocks when we
− * re−enter the quota code and are already holding the mutex */
− if (IS_NOQUOTA(inode)) {
−out_add:
− inode_add_bytes(inode, number);
− return QUOTA_OK;
− }
for (cnt = 0; cnt < MAXQUOTAS; cnt++)
warntype[cnt] = QUOTA_NL_NOWARN;

− down_read(&sb_dqopt(inode−>i_sb)−>dqptr_sem);
− if (IS_NOQUOTA(inode)) { /* Now we can do reliable test... */
− up_read(&sb_dqopt(inode−>i_sb)−>dqptr_sem);
− goto out_add;
− }
spin_lock(&dq_data_lock);
for (cnt = 0; cnt < MAXQUOTAS; cnt++) {
if (inode−>i_dquot[cnt] == NODQUOT)
continue;
− if (check_bdq(inode−>i_dquot[cnt], number, warn, warntype+cnt) == NO_QUOTA)
− goto warn_put_all;
+ if (check_bdq(inode−>i_dquot[cnt], number, warn, warntype+cnt)
+ == NO_QUOTA) {
+ ret = NO_QUOTA;
+ goto out_unlock;
+ }
}
for (cnt = 0; cnt < MAXQUOTAS; cnt++) {
if (inode−>i_dquot[cnt] == NODQUOT)
continue;
− dquot_incr_space(inode−>i_dquot[cnt], number);
+ if (reserve)
+ dquot_resv_space(inode−>i_dquot[cnt], number);
+ else
+ dquot_incr_space(inode−>i_dquot[cnt], number);
}
− inode_add_bytes(inode, number);
− ret = QUOTA_OK;
−warn_put_all:
+ if (!reserve)
+ inode_add_bytes(inode, number);
+out_unlock:
```

```
spin_unlock(&dq_data_lock);
− if (ret == QUOTA_OK)
− /* Dirtify all the dquots − this can block when journalling */
− for (cnt = 0; cnt < MAXQUOTAS; cnt++)
− if (inode−>i_dquot[cnt])
− mark_dquot_dirty(inode−>i_dquot[cnt]);
flush_warnings(inode−>i_dquot, warntype);
+ return ret;
+}
+
+int dquot_alloc_space(struct inode *inode, qsize_t number, int warn)
+{
+ int cnt, ret = QUOTA_OK;
+
+ /*
+ * First test before acquiring mutex − solves deadlocks when we
+ * re−enter the quota code and are already holding the mutex
+ */
+ if (IS_NOQUOTA(inode)) {
+ inode_add_bytes(inode, number);
+ goto out;
+ }
+
+ down_read(&sb_dqopt(inode−>i_sb)−>dqptr_sem);
+ if (IS_NOQUOTA(inode)) {
+ inode_add_bytes(inode, number);
+ goto out_unlock;
+ }
+
+ ret = __dquot_alloc_space(inode, number, warn, 0);
+ if (ret == NO_QUOTA)
+ goto out_unlock;
+
+ /* Dirtify all the dquots − this can block when journalling */
+ for (cnt = 0; cnt < MAXQUOTAS; cnt++)
+ if (inode−>i_dquot[cnt])
+ mark_dquot_dirty(inode−>i_dquot[cnt]);
+out_unlock:
up_read(&sb_dqopt(inode−>i_sb)−>dqptr_sem);
+out:
+ return ret;
+}
+
+int dquot_reserve_space(struct inode *inode, qsize_t number, int warn)
+{
+ int ret = QUOTA_OK;
+
+ if (IS_NOQUOTA(inode))
+ goto out;
+
+ down_read(&sb_dqopt(inode−>i_sb)−>dqptr_sem);
```

```
+ if (IS_NOQUOTA(inode))
+ goto out_unlock;
+
+ ret = __dquot_alloc_space(inode, number, warn, 1);
+out_unlock:
+ up_read(&sb_dqopt(inode−>i_sb)−>dqptr_sem);
+out:
return ret;
}
+EXPORT_SYMBOL(dquot_reserve_space);

/*
* This operation can block, but only after everything is updated
@@ −2057,7 +2112,7 @@ static void do_get_dqblk(struct dquot *dquot, struct if_dqblk *di)
spin_lock(&dq_data_lock);
di−>dqb_bhardlimit = stoqb(dm−>dqb_bhardlimit);
di−>dqb_bsoftlimit = stoqb(dm−>dqb_bsoftlimit);
− di−>dqb_curspace = dm−>dqb_curspace;
+ di−>dqb_curspace = dm−>dqb_curspace + dm−>dqb_rsvspace;
di−>dqb_ihardlimit = dm−>dqb_ihardlimit;
di−>dqb_isoftlimit = dm−>dqb_isoftlimit;
di−>dqb_curinodes = dm−>dqb_curinodes;
@@ −2097,7 +2152,7 @@ static int do_set_dqblk(struct dquot *dquot, struct if_dqblk *di)

spin_lock(&dq_data_lock);
if (di−>dqb_valid & QIF_SPACE) {
− dm−>dqb_curspace = di−>dqb_curspace;
+ dm−>dqb_curspace = di−>dqb_curspace − dm−>dqb_rsvspace;
check_blim = 1;
__set_bit(DQ_LASTSET_B + QIF_SPACE_B, &dquot−>dq_flags);
}
diff −−git a/include/linux/quota.h b/include/linux/quota.h
index d72d5d8..54b837f 100644
−−− a/include/linux/quota.h
+++ b/include/linux/quota.h
@@ −198,6 +198,7 @@ struct mem_dqblk {
qsize_t dqb_bhardlimit; /* absolute limit on disk blks alloc */
qsize_t dqb_bsoftlimit; /* preferred limit on disk blks */
qsize_t dqb_curspace; /* current used space */
+ qsize_t dqb_rsvspace; /* current reserved space for delalloc*/
qsize_t dqb_ihardlimit; /* absolute limit on allocated inodes */
qsize_t dqb_isoftlimit; /* preferred inode limit */
qsize_t dqb_curinodes; /* current # allocated inodes */
@@ −308,6 +309,8 @@ struct dquot_operations {
int (*release_dquot) (struct dquot *); /* Quota is going to be deleted from disk */
int (*mark_dirty) (struct dquot *); /* Dquot is marked dirty */
int (*write_info) (struct super_block *, int); /* Write of quota "superblock" */
+ /* reserve quota for delayed block allocation */
+ int (*reserve_space) (struct inode *, qsize_t, int);
};
```

```
/* Operations handling requests from userspace */
diff −−git a/include/linux/quotaops.h b/include/linux/quotaops.h
index 0b35b3a..3e3a0d2 100644
−−− a/include/linux/quotaops.h
+++ b/include/linux/quotaops.h
@@ −183,6 +183,16 @@ static inline int vfs_dq_alloc_space(struct inode *inode, qsize_t nr)
return ret;
}

+static inline int vfs_dq_reserve_space(struct inode *inode, qsize_t nr)
+{
+ if (sb_any_quota_active(inode−>i_sb)) {
+ /* Used space is updated in alloc_space() */
+ if (inode−>i_sb−>dq_op−>reserve_space(inode, nr, 0) == NO_QUOTA)
+ return 1;
+ }
+ return 0;
+}
+
static inline int vfs_dq_alloc_inode(struct inode *inode)
{
if (sb_any_quota_active(inode−>i_sb)) {
@@ −339,6 +349,11 @@ static inline int vfs_dq_alloc_space(struct inode *inode, qsize_t nr)
return 0;
}

+static inline int vfs_dq_reserve_space(struct inode *inode, qsize_t nr)
+{
+ return 0;
+}
+
static inline void vfs_dq_free_space_nodirty(struct inode *inode, qsize_t nr)
{
inode_sub_bytes(inode, nr);
@@ −376,6 +391,12 @@ static inline int vfs_dq_alloc_block(struct inode *inode, qsize_t nr)
nr << inode−>i_sb−>s_blocksize_bits);
}

+static inline int vfs_dq_reserve_block(struct inode *inode, qsize_t nr)
+{
+ return vfs_dq_reserve_space(inode,
+ nr << inode−>i_blkbits);
+}
+
static inline void vfs_dq_free_block_nodirty(struct inode *inode, qsize_t nr)
{
vfs_dq_free_space_nodirty(inode, nr << inode−>i_sb−>s_blocksize_bits);
−−
1.6.0.2

−−
```