

[Patch 7/10]: ext3 online resize: SMP locking for group metadata

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-09/9038.html>

From: Stephen Tweedie (sct_at_redhat.com)

Date: 09/30/04

Date: Thu, 30 Sep 2004 14:23:50 +0100

To: linux-kernel@vger.kernel.org, Andrew Morton <akpm@osdl.org>, Andreas Dilger <[Fix the SMP locking for group metadata in online resize.](mailto:adilger@clusterf</p></div><div data-bbox=)

Most of the resize is simply not dangerous from an SMP point of view, as we're adding new data beyond the end of the device and so we're guaranteed that no existing CPUs can already be using that data. But we need to be extremely careful about the ordering when enabling that new data.

The key to this is `sb->s_groups_count`; when that is raised, it enables new space on a filesystem, and the kernel will suddenly start accessing all of the newly-created tables for that space.

The precise rules we use are:

- * Writers of `s_groups_count` *must* hold `lock_super`
- AND
- * Writers must perform a `smp_wmb()` after updating all dependent data and before modifying the groups count

- * Readers must hold `lock_super()` over the access
- OR
- * Readers must perform an `smp_rmb()` after reading the groups count and before reading any dependent data.

This leaves the hot path, where `s_groups_count` is referenced, requiring an `smp_rmb()`; but no other locking on that hot path is required.

Signed-off-by: Stephen Tweedie <sct@redhat.com>

```
--- linux-2.6.9-rc2-mm4/fs/ext3/balloc.c.=K0006=.orig
+++ linux-2.6.9-rc2-mm4/fs/ext3/balloc.c
@@ -54,6 +54,7 @@ struct ext3_group_desc * ext3_get_group_

    return NULL;
}
```

```

+ smp_rmb();

    group_desc = block_group / EXT3_DESC_PER_BLOCK(sb);
    desc = block_group % EXT3_DESC_PER_BLOCK(sb);
@@ -1140,6 +1141,8 @@ int ext3_new_block(handle_t *handle, str
#ifdef EXT3FS_DEBUG
    static int goal_hits, goal_attempts;
#endif
+ unsigned long ngroups;
+
    *errp = -ENOSPC;
    sb = inode->i_sb;
    if (!sb) {
@@ -1194,13 +1197,16 @@ retry:
        goto allocated;
    }

+ ngroups = EXT3_SB(sb)->s_groups_count;
+ smp_rmb();
+
    /*
     * Now search the rest of the groups. We assume that
     * i and gdp correctly point to the last group visited.
     */
- for (bgi = 0; bgi < EXT3_SB(sb)->s_groups_count; bgi++) {
+ for (bgi = 0; bgi < ngroups; bgi++) {
        group_no++;
- if (group_no >= EXT3_SB(sb)->s_groups_count)
+ if (group_no >= ngroups)
        group_no = 0;
        gdp = ext3_get_group_desc(sb, group_no, &gdp_bh);
        if (!gdp) {
@@ -1346,6 +1352,7 @@ unsigned long ext3_count_free_blocks(str
    unsigned long desc_count;
    struct ext3_group_desc *gdp;
    int i;
+ unsigned long ngroups;
#ifdef EXT3FS_DEBUG
    struct ext3_super_block *es;
    unsigned long bitmap_count, x;
@@ -1378,7 +1385,9 @@ unsigned long ext3_count_free_blocks(str
    return bitmap_count;
#else
    desc_count = 0;
- for (i = 0; i < EXT3_SB(sb)->s_groups_count; i++) {
+ ngroups = EXT3_SB(sb)->s_groups_count;
+ smp_rmb();
+ for (i = 0; i < ngroups; i++) {
        gdp = ext3_get_group_desc(sb, i, NULL);
        if (!gdp)
            continue;

```

Linux-Kernel: [Patch 7/10]: ext3 online resize: SMP locking for group metadata

```

--- linux-2.6.9-rc2-mm4/fs/ext3/inode.c.=K0006=.orig
+++ linux-2.6.9-rc2-mm4/fs/ext3/inode.c
@@ -2239,6 +2239,7 @@ static unsigned long ext3_get_inode_block
        "group >= groups count");
        return 0;
    }
+ smp_rmb();
    group_desc = block_group >> EXT3_DESC_PER_BLOCK_BITS(sb);
    desc = block_group & (EXT3_DESC_PER_BLOCK(sb) - 1);
    bh = EXT3_SB(sb)->s_group_desc[group_desc];
--- linux-2.6.9-rc2-mm4/fs/ext3/resize.c.=K0006=.orig
+++ linux-2.6.9-rc2-mm4/fs/ext3/resize.c
@@ -784,7 +784,26 @@ int ext3_group_add(struct super_block *s
    } else if ((err = add_new_gdb(handle, inode, input, &primary)))
        goto exit_journal;

- /* Finally update group descriptor block for new group */
+ /*
+ * OK, now we've set up the new group. Time to make it active.
+ *
+ * Current kernels don't lock all allocations via lock_super(),
+ * so we have to be safe wrt. concurrent accesses the group
+ * data. So we need to be careful to set all of the relevant
+ * group descriptor data etc. *before* we enable the group.
+ *
+ * The key field here is EXT3_SB(sb)->s_groups_count: as long as
+ * that retains its old value, nobody is going to access the new
+ * group.
+ *
+ * So first we update all the descriptor metadata for the new
+ * group; then we update the total disk blocks count; then we
+ * update the groups count to enable the group; then finally we
+ * update the free space counts so that the system can start
+ * using the new disk blocks.
+ */
+
+ /* Update group descriptor block for new group */
    gdp = (struct ext3_group_desc *)primary->b_data + gdb_off;

    gdp->bg_block_bitmap = cpu_to_le32(input->block_bitmap);
@@ -793,22 +812,61 @@ int ext3_group_add(struct super_block *s
    gdp->bg_free_blocks_count = cpu_to_le16(input->free_blocks_count);
    gdp->bg_free_inodes_count = cpu_to_le16(EXT3_INODES_PER_GROUP(sb));

+ /*
+ * Make the new blocks and inodes valid next. We do this before
+ * increasing the group count so that once the group is enabled,
+ * all of its blocks and inodes are already valid.
+ *
+ * We always allocate group-by-group, then block-by-block or
+ * inode-by-inode within a group, so enabling these

```

Linux-Kernel: [Patch 7/10]: ext3 online resize: SMP locking for group metadata

```
+ * blocks/inodes before the group is live won't actually let us
+ * allocate the new space yet.
+ */
+ es->s_blocks_count = cpu_to_le32(le32_to_cpu(es->s_blocks_count) +
+ input->blocks_count);
+ es->s_inodes_count = cpu_to_le32(le32_to_cpu(es->s_inodes_count) +
+ EXT3_INODES_PER_GROUP(sb));
+
+ /*
+ * We need to protect s_groups_count against other CPUs seeing
+ * inconsistent state in the superblock.
+ *
+ * The precise rules we use are:
+ *
+ * * Writers of s_groups_count *must* hold lock_super
+ * AND
+ * * Writers must perform a smp_wmb() after updating all dependent
+ * data and before modifying the groups count
+ *
+ * * Readers must hold lock_super() over the access
+ * OR
+ * * Readers must perform an smp_rmb() after reading the groups count
+ * and before reading any dependent data.
+ *
+ * NB. These rules can be relaxed when checking the group count
+ * while freeing data, as we can only allocate from a block
+ * group after serialising against the group count, and we can
+ * only then free after serialising in turn against that
+ * allocation.
+ */
+ smp_wmb();
+
+ /* Update the global fs size fields */
+     EXT3_SB(sb)->s_groups_count++;
+
+     ext3_journal_dirty_metadata(handle, primary);
+
+ /* Update superblock with new block counts */
+ es->s_blocks_count = cpu_to_le32(le32_to_cpu(es->s_blocks_count) +
+ input->blocks_count);
+ es->s_free_blocks_count =
+ cpu_to_le32(le32_to_cpu(es->s_free_blocks_count) +
+ input->free_blocks_count);
+ /* Update the reserved block counts only once the new group is
+ * active. */
+     es->s_r_blocks_count = cpu_to_le32(le32_to_cpu(es->s_r_blocks_count) +
+     input->reserved_blocks);
+ es->s_inodes_count = cpu_to_le32(le32_to_cpu(es->s_inodes_count) +
+ EXT3_INODES_PER_GROUP(sb));
+ es->s_free_inodes_count =
+ cpu_to_le32(le32_to_cpu(es->s_free_inodes_count) +
```

Linux-Kernel: [Patch 7/10]: ext3 online resize: SMP locking for group metadata

```

- EXT3_INODES_PER_GROUP(sb));
+
+ /* Update the free space counts */
+ percpu_counter_mod(&sbi->s_freeblocks_counter,
+ input->free_blocks_count);
+ percpu_counter_mod(&sbi->s_freeinodes_counter,
+ EXT3_INODES_PER_GROUP(sb));
+
+     ext3_journal_dirty_metadata(handle, EXT3_SB(sb)->s_sbh);
+     sb->s_dirt = 1;

@@ -849,6 +907,9 @@ int ext3_group_extend(struct super_block
    handle_t *handle;
    int err;

+ /* We don't need to worry about locking wrt other resizers just
+ * yet: we're going to revalidate es->s_blocks_count after
+ * taking lock_super() below. */
+     o_blocks_count = le32_to_cpu(es->s_blocks_count);
+     o_groups_count = EXT3_SB(sb)->s_groups_count;

---- linux-2.6.9-rc2-mm4/fs/ext3/super.c=K0006=.orig
+++ linux-2.6.9-rc2-mm4/fs/ext3/super.c
@@ -1014,6 +1014,7 @@ static int ext3_setup_super(struct super
    return res;
}

+ /* Called at mount-time, super-block is locked */
static int ext3_check_descriptors (struct super_block * sb)
{
    struct ext3_sb_info *sbi = EXT3_SB(sb);
@@ -2139,6 +2140,10 @@ int ext3_statfs (struct super_block * sb
    if (test_opt (sb, MINIX_DF))
        overhead = 0;
    else {
+ unsigned long ngroups;
+ ngroups = EXT3_SB(sb)->s_groups_count;
+ smp_rmb();
+
+     /*
+      * Compute the overhead (FS structures)
+      */
@@ -2154,7 +2159,7 @@ int ext3_statfs (struct super_block * sb
+     /* block group descriptors. If the sparse superblocks
+      * feature is turned on, then not all groups have this.
+      */
- for (i = 0; i < EXT3_SB(sb)->s_groups_count; i++) {
+ for (i = 0; i < ngroups; i++) {
+     overhead += ext3_bg_has_super(sb, i) +
+         ext3_bg_num_gdb(sb, i);
+     cond_resched();

```

Linux-Kernel: [Patch 7/10]: ext3 online resize: SMP locking for group metadata

```
@@ -2164,8 +2169,7 @@ int ext3_statfs (struct super_block * sb
    * Every block group has an inode bitmap, a block
    * bitmap, and an inode table.
    */
- overhead += (EXT3_SB(sb)->s_groups_count *
- (2 + EXT3_SB(sb)->s_itb_per_group));
+ overhead += (ngroups * (2 + EXT3_SB(sb)->s_itb_per_group));
    }

    buf->f_type = EXT3_SUPER_MAGIC;
```

-
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>