author	Josef Bacik <josef@toxicpanda.com>	2022-10-14 08:52:46 -0400
committer	David Sterba <dsterba@suse.com>	2022-10-24 15:28:07 +0200
commit	968b71583130b6104c9f33ba60446d598e327a8b (patch)	
tree	2494d18a237aaf7b28841d2a6aa9a40604a89f80	
parent	ae0e5df4d1a4a2694c9c203cc25334aaaf9f2dfa (diff)	
download	linux-968b71583130b6104c9f33ba60446d598e327a8b.tar.gz	

diff options

context: 3

space: include

mode: unified

btrfs: fix tree mod log mishandling of reallocated nodes

We have been seeing the following panic in production

```
  kernel BUG at fs/btrfs/tree-mod-log.c:677!
  invalid opcode: 0000 [#1] SMP
  RIP: 0010:tree_mod_log_rewind+0x1b4/0x200
  RSP: 0000:ffffc9002c02f890 EFLAGS: 00010293
  RAX: 0000000000000003 RBX: ffff8882b448c700 RCX: 0000000000000000
  RDX: 0000000000008000 RSI: 00000000000000a7 RDI: ffff88877d831c00
  RBP: 0000000000000002 R08: 000000000000009f R09: 0000000000000000
  R10: 0000000000000000 R11: 0000000000100c40 R12: 0000000000000001
  R13: ffff8886c26d6a00 R14: ffff88829f5424f8 R15: ffff88877d831a00
  FS:  00007fee1d80c780(0000) GS:ffff8890400c0000(0000) knlGS:0000000000000000
  CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
  CR2: 00007fee1963a020 CR3: 0000000434f33002 CR4: 00000000007706e0
  DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
  DR3: 0000000000000000 DR6: 00000000fffe0ff0 DR7: 0000000000000400
  PKRU: 55555554
  Call Trace:
   btrfs_get_old_root+0x12b/0x420
   btrfs_search_old_slot+0x64/0x2f0
   ? tree_mod_log_oldest_root+0x3d/0xf0
   resolve_indirect_ref+0xfd/0x660
   ? ulist_alloc+0x31/0x60
   ? kmem_cache_alloc_trace+0x114/0x2c0
   find_parent_nodes+0x97a/0x17e0
   ? ulist_alloc+0x30/0x60
   btrfs_find_all_roots_safe+0x97/0x150
   iterate_extent_inodes+0x154/0x370
   ? btrfs_search_path_in_tree+0x240/0x240
   iterate_inodes_from_logical+0x98/0xd0
   ? btrfs_search_path_in_tree+0x240/0x240
   btrfs_ioctl_logical_to_ino+0xd9/0x180
   btrfs_ioctl+0xe2/0x2ec0
   ? __mod_memcg_lruvec_state+0x3d/0x280
   ? do_sys_openat2+0x6d/0x140
   ? kretprobe_dispatcher+0x47/0x70
   ? kretprobe_rethook_handler+0x38/0x50
   ? rethook_trampoline_handler+0x82/0x140
   ? arch_rethook_trampoline_callback+0x3b/0x50
   ? kmem_cache_free+0xfb/0x270
   ? do_sys_openat2+0xd5/0x140
```

```
   __x64_sys_ioctl+0x71/0xb0
   do_syscall_64+0x2d/0x40
```

Which is this code in tree_mod_log_rewind()

```
        switch (tm->op) {
        case BTRFS_MOD_LOG_KEY_REMOVE_WHILE_FREEING:
                BUG_ON(tm->slot < n);
```

This occurs because we replay the nodes in order that they happened, and when we do a REPLACE we will log a REMOVE_WHILE_FREEING for every slot, starting at 0. 'n' here is the number of items in this block, which in this case was 1, but we had 2 REMOVE_WHILE_FREEING operations.

The actual root cause of this was that we were replaying operations for a block that shouldn't have been replayed. Consider the following sequence of events

1. We have an already modified root, and we do a btrfs_get_tree_mod_seq().
2. We begin removing items from this root, triggering KEY_REPLACE for it's child slots.
3. We remove one of the 2 children this root node points to, thus triggering the root node promotion of the remaining child, and freeing this node.
4. We modify a new root, and re-allocate the above node to the root node of this other root.

The tree mod log looks something like this

```
        logical 0       op KEY_REPLACE (slot 1)                 seq 2
        logical 0       op KEY_REMOVE (slot 1)                  seq 3
        logical 0       op KEY_REMOVE_WHILE_FREEING (slot 0)    seq 4
        logical 4096    op LOG_ROOT_REPLACE (old logical 0)     seq 5
        logical 8192    op KEY_REMOVE_WHILE_FREEING (slot 1)    seq 6
        logical 8192    op KEY_REMOVE_WHILE_FREEING (slot 0)    seq 7
        logical 0       op LOG_ROOT_REPLACE (old logical 8192)  seq 8
```

>From here the bug is triggered by the following steps

1. Call btrfs_get_old_root() on the new_root.
2. We call tree_mod_log_oldest_root(btrfs_root_node(new_root)), which is currently logical 0.
3. tree_mod_log_oldest_root() calls tree_mod_log_search_oldest(), which gives us the KEY_REPLACE seq 2, and since that's not a LOG_ROOT_REPLACE we incorrectly believe that we don't have an old root, because we expect that the most recent change should be a LOG_ROOT_REPLACE.
4. Back in tree_mod_log_oldest_root() we don't have a LOG_ROOT_REPLACE, so we don't set old_root, we simply use our existing extent buffer.
5. Since we're using our existing extent buffer (logical 0) we call tree_mod_log_search(0) in order to get the newest change to start the rewind from, which ends up being the LOG_ROOT_REPLACE at seq 8.
6. Again since we didn't find an old_root we simply clone logical 0 at it's current state.
7. We call tree_mod_log_rewind() with the cloned extent buffer.
8. Set n = btrfs_header_nritems(logical 0), which would be whatever the original nritems was when we COWed the original root, say for this example it's 2.
9. We start from the newest operation and work our way forward, so we see LOG_ROOT_REPLACE which we ignore.
10. Next we see KEY_REMOVE_WHILE_FREEING for slot 0, which triggers the BUG_ON(tm->slot < n), because it expects if we've done this we have a completely empty extent buffer to replay completely.

The correct thing would be to find the first LOG_ROOT_REPLACE, and then
get the old_root set to logical 8192. In fact making that change fixes
this particular problem.

However consider the much more complicated case. We have a child node
in this tree and the above situation. In the above case we freed one
of the child blocks at the seq 3 operation. If this block was also
re-allocated and got new tree mod log operations we would have a
different problem. btrfs_search_old_slot(orig root) would get down to
the logical 0 root that still pointed at that node. However in
btrfs_search_old_slot() we call tree_mod_log_rewind(buf) directly. This
is not context aware enough to know which operations we should be
replaying. If the block was re-allocated multiple times we may only
want to replay a range of operations, and determining what that range is
isn't possible to determine.

We could maybe solve this by keeping track of which root the node
belonged to at every tree mod log operation, and then passing this
around to make sure we're only replaying operations that relate to the
root we're trying to rewind.

However there's a simpler way to solve this problem, simply disallow
reallocations if we have currently running tree mod log users. We
already do this for leaf's, so we're simply expanding this to nodes as
well. This is a relatively uncommon occurrence, and the problem is
complicated enough I'm worried that we will still have corner cases in
the reallocation case. So fix this in the most straightforward way
possible.

Fixes: bd989ba359f2 ("Btrfs: add tree modification log functions")
CC: stable@vger.kernel.org # 3.3+
Reviewed-by: Filipe Manana <fdmanana@suse.com>
Signed-off-by: Josef Bacik <josef@toxicpanda.com>
Signed-off-by: David Sterba <dsterba@suse.com>

Diffstat

-rw-r--r-- fs/btrfs/extent-tree.c 25

1 files changed, 13 insertions, 12 deletions

```
diff --git a/fs/btrfs/extent-tree.c b/fs/btrfs/extent-tree.c
index cd2d36580f1ac1..2801c991814f57 100644
--- a/fs/btrfs/extent-tree.c
+++ b/fs/btrfs/extent-tree.c
@@ -3295,21 +3295,22 @@ void btrfs_free_tree_block(struct btrfs_trans_handle *trans,
                }

                /*
-                * If this is a leaf and there are tree mod log users, we may
-                * have recorded mod log operations that point to this leaf.
-                * So we must make sure no one reuses this leaf's extent before
-                * mod log operations are applied to a node, otherwise after
-                * rewinding a node using the mod log operations we get an
-                * inconsistent btree, as the leaf's extent may now be used as
-                * a node or leaf for another different btree.
+                * If there are tree mod log users we may have recorded mod log
+                * operations for this node.  If we re-allocate this node we
+                * could replay operations on this node that happened when it
+                * existed in a completely different root.  For example if it
+                * was part of root A, then was reallocated to root B, and we
+                * are doing a btrfs_old_search_slot(root b), we could replay
```

```
+                 * operations that happened when the block was part of root A,
+                 * giving us an inconsistent view of the btree.
+                 *
                  * We are safe from races here because at this point no other
                  * node or root points to this extent buffer, so if after this
-                 * check a new tree mod log user joins, it will not be able to
-                 * find a node pointing to this leaf and record operations that
-                 * point to this leaf.
+                 * check a new tree mod log user joins we will not have an
+                 * existing log of operations on this node that we have to
+                 * contend with.
                  */
-                if (btrfs_header_level(buf) == 0 &&
-                    test_bit(BTRFS_FS_TREE_MOD_LOG_USERS, &fs_info->flags))
+                if (test_bit(BTRFS_FS_TREE_MOD_LOG_USERS, &fs_info->flags))
                         must_pin = true;

                 if (must_pin || btrfs_is_zoned(fs_info)) {
```