Subject: Re: [PATCH] retries in ext3_prepare_write() violate ordering requirements
Posted by dev on Mon, 13 Nov 2006 12:32:37 GMT
View Forum Message <> Reply to Message

Andrew,

>>in journal=ordered or journal=data mode retry in ext3_prepare_write()
>>breaks the requirements of journaling of data with respect to metadata.
>>The fix is to call commit_write to commit allocated zero blocks before
>>retry.
>>
>
>
> How was this problem detected? (ie: why was block_prepare_write() failing?)
purely theoretically while hunting for other bugs related to ext3 and quota.
block_prepare_write() can fail e.g. if quota returns -EDQUOT in ext3_alloc_blocks().

> How was the patch tested?
1. it was tested as part of OpenVZ kernel
2. there were ext3 stress test done with lots of disk activity by Dmitry Monakhov.

> Was nobh-mode also tested?
I will ask to perform some more tests 100% triggering ext3_prepare_failure()
and with NOBH mode.

Thanks,
Kirill

>>--- ./fs/ext3/inode.c.ext3pw 2006-11-08 17:44:14.000000000 +0300
>>+++ ./fs/ext3/inode.c 2006-11-08 17:48:59.000000000 +0300
>>@@ -1148,37 +1148,89 @@ static int do_journal_get_write_access(h
>> return ext3_journal_get_write_access(handle, bh);
>> }
>>
>>+/*
>>+ * The idea of this helper function is following:
>>+ * if prepare_write has allocated some blocks, but not all of them, the
>>+ * transaction must include the content of the newly allocated blocks.
>>+ * This content is expected to be set to zeroes by block_prepare_write().
>>+ * 2006/10/14 SAW
>>+ */
>>+static int ext3_prepare_failure(struct file *file, struct page *page,
>>+   unsigned from, unsigned to)
>>+{
>>+ struct address_space *mapping;
>>+ struct buffer_head *bh, *head, *next;
>>+ unsigned block_start, block_end;

```
>>+ unsigned blocksize;
>>+
>>+ mapping = page->mapping;
>>+ if (ext3_should_writeback_data(mapping->host)) {
>>+  /* optimization: no constraints about data */
>>+skip:
>>+  ext3_journal_stop(ext3_journal_current_handle());
>>+  return 0;
>
>
> Should this be `return ext3_journal_stop(...);'?
>
>
>>+ }
>>+
>>+ head = page_buffers(page);
>>+ blocksize = head->b_size;
>>+ for ( bh = head, block_start = 0;
>>+  bh != head || !block_start;
>>+      block_start = block_end, bh = next)
>>+ {
>>+  next = bh->b_this_page;
>>+  block_end = block_start + blocksize;
>>+  if (block_end <= from)
>>+   continue;
>>+  if (block_start >= to) {
>>+   block_start = to;
>>+   break;
>>+  }
>>+  if (!buffer_mapped(bh))
>>+   break;
>
>
> What is the significance of buffer_mapped() here?  Outside EOF or into a
> hole?  If so, then block_start >= to, and we can't get here??
>
>
>>+ }
>>+ if (block_start <= from)
>>+  goto skip;
>>+
>>+ /* commit allocated and zeroed buffers */
>>+ return mapping->a_ops->commit_write(file, page, from, block_start);
>>+}
>>+
>> static int ext3_prepare_write(struct file *file, struct page *page,
>>          unsigned from, unsigned to)
>> {
```

```
>>  struct inode *inode = page->mapping->host;
>>- int ret, needed_blocks = ext3_writepage_trans_blocks(inode);
>>+ int ret, ret2;
>>+ int needed_blocks = ext3_writepage_trans_blocks(inode);
>>  handle_t *handle;
>>  int retries = 0;
>>
>> retry:
>>  handle = ext3_journal_start(inode, needed_blocks);
>>- if (IS_ERR(handle)) {
>>-  ret = PTR_ERR(handle);
>>-  goto out;
>>- }
>>+ if (IS_ERR(handle))
>>+  return PTR_ERR(handle);
>>  if (test_opt(inode->i_sb, NOBH) && ext3_should_writeback_data(inode))
>>   ret = nobh_prepare_write(page, from, to, ext3_get_block);
>>  else
>>   ret = block_prepare_write(page, from, to, ext3_get_block);
>>  if (ret)
>>-  goto prepare_write_failed;
>>+  goto failure;
>>
>>  if (ext3_should_journal_data(inode)) {
>>   ret = walk_page_buffers(handle, page_buffers(page),
>>     from, to, NULL, do_journal_get_write_access);
>>+  if (ret)
>>+   /* fatal error, just put the handle and return */
>>+   journal_stop(handle);
>>  }
>>-prepare_write_failed:
>>- if (ret)
>>-  ext3_journal_stop(handle);
>>+ return ret;
>>+
>>+failure:
>>+ ret2 = ext3_prepare_failure(file, page, from, to);
>>+ if (ret2 < 0)
>>+  return ret2;
>>  if (ret == -ENOSPC && ext3_should_retry_alloc(inode->i_sb, &retries))
>>   goto retry;
>>-out:
>>+ /* retry number exceeded, or other error like -EDQUOT */
>>  return ret;
>> }
>>
>
>
```