

Re: [RFC] JBD ordered mode rewrite

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-03/msg03978.html>

- *From:* Jan Kara <jack@xxxxxxx>
 - *Date:* Mon, 10 Mar 2008 19:29:03 +0100
-

Hi Mingming,

On Fri 07-03-08 02:55:01, Mingming Cao wrote:

On Thu, 2008-03-06 at 18:42 +0100, Jan Kara wrote:

Below is my rewrite of ordered mode in JBD. Now we don't have a list of data buffers that need syncing on transaction commit but a list of inodes that need writeout during commit. This brings all sorts of advantages such as possibility to get rid of journal heads and buffer heads for data buffers in ordered mode, better ordering of writes on transaction commit, simplification of some JBD code, no more anonymous pages when truncate of data being committed happens. The patch has survived some light testing but it still has some potential of eating your data so beware :) I've run dbench to see whether we didn't decrease performance by different handling of truncate and the throughput I'm getting on my machine is the same (OK, is lower by 0.5%) if I disable the code in truncate waiting for commit to finish... Also the throughput of dbench is about 2% better with my patch than with current JBD.

I know ext4 is keep changing that it's a bit hard to create patch against ext4, but I feel features like especially rewrite the default ordered mode should done in ext4/jbd2. I could port to current ext4 and JBD2 if you agrees with this.

I definitely agree with you :). This was just the first version of the patch, more like a proof-of-concept, and it was easier to code against jbd/ext3 for me :) But when I have something more definitive, I'll port it against jbd2/ext4.

Also, would it make sense to create a new ordered mode writepage routines, and keep the old ordered mode code there for a while, to allow easy comparison? This could a good transition for people to start experiment this ordered mode without worrying about put data in danger by default.

Re: [RFC] JBD ordered mode rewrite

This is an interesting idea. I'll have a look how hard would it be to do that. It would be also good because we could separate the patch into several chunks and it would still compile / work between them.

Any comments or testing most welcom

[...]

```
/*
 * Note that we always start a transaction even if we're not journaling
 * data. This is to preserve ordering: any hole instantiation within
 @@ -1465,15 +1444,11 @@ static int journal_dirty_data_fn(handle_t
 *handle, struct buffer_head *bh)
 * We don't honour synchronous mounts for writepage(). That would be
 * disastrous. Any write() or metadata operation will sync the fs for
 * us.
 - *
 - * AKPM2: if all the page's buffers are mapped to disk and !data=journal,
 - * we don't need to open a transaction here.
 */
static int ext3_ordered_writepage(struct page *page,
struct writeback_control *wbc)
{
struct inode *inode = page->mapping->host;
- struct buffer_head *page_bufs;
handle_t *handle = NULL;
int ret = 0;
int err;
 @@ -1487,46 +1462,49 @@ static int ext3_ordered_writepage(struct page
 *page,
if (ext3_journal_current_handle())
goto out_fail;

- handle = ext3_journal_start(inode, ext3_writepage_trans_blocks(inode));
-
- if (IS_ERR(handle)) {
- ret = PTR_ERR(handle);
- goto out_fail;
+ /*
+ * Now there are two different reasons why we can be called:
+ * 1) write out during commit
+ * 2) fsync / writeout to free memory
+ *
+ * In the first case, we just need to write the buffer to disk, in the
+ * second case we may need to do hole filling and attach the inode to
+ * the transaction. Note that even in the first case, we may get an
+ * unmapped buffer (hole fill with data via mmap) but we don't have to
+ * write it - actually, we can't because from a transaction commit we
+ * cannot start a new transaction or we could deadlock.
```

Re: [RFC] JBD ordered mode rewrite

+ */

Any thoughts how to handle the unmapped page under case 1)? Right now I see it fails. Your comments here saying that we still have the issue that "can't start a new transaction while committing", but likely, with delayed allocation, starting a new transaction could to happen a lot to do defered block allocation.

I really hope this new mode could be easy to add delayed allocation support. Any thoughts that we could workaround the locking in the JBD2 layer?

"can't start a new transaction while committing" is a fundamental problem that you cannot add to a journal when you are trying to clean it up. It's not just a locking problem :). As Mark pointed out, there's another problem with the fact that we cannot afford to take page_lock while committing because that's essentially a clasical lock inversion between a transaction start and a page lock. How to solve that one, I don't know yet. To your question about delayed allocation – currently, we just skip those buffers so as you write in some other email, you basically get the guarantees of writeback mode. Actually, I guess the result is the same with the old way of ordered–mode handling since you cannot afford to do the block allocation from the commit code as well... We could get around this limitation (actually easier than in the old code) if we were sure we have enough credits in the transaction to really do the allocation – we would do the allocation in the writepage() and attach changed buffers to the committing transaction.

```
+ if (!wbc->skip_unmapped) {
+ handle = ext3_journal_start(inode,
+ ext3_writepage_trans_blocks(inode));
+ if (IS_ERR(handle)) {
+ ret = PTR_ERR(handle);
+ goto out_fail;
+ }
+ }
+ else if (!PageMappedToDisk(page))
+ goto out_fail;
```

Honza

—

Jan Kara <jack@xxxxxxx>
SUSE Labs, CR

—

To unsubscribe from this list: send the line "unsubscribe linux–kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo–info.html>

Re: [RFC] JBD ordered mode rewrite

Re: [RFC] JBD ordered mode rewrite

Please read the FAQ at <http://www.tux.org/lkml/>