Subject: [PATCH 1/1][RFC] EXT34 retry loop issue V(2)
Posted by Dmitriy Monakhov on Mon, 12 Feb 2007 09:46:58 GMT
View Forum Message <> Reply to Message

Patch depends on : "[PATCH 0/1][RFC] prepare_write positive return value V(2)"

This patch solve ext3/4 retry loop issue.
Issue description:
What we can do if block_prepare_write fail inside ext3_prepare_write ?
a) Stop transaction and do retry if possible, but what happend if
   reboot comes after journal_force_commit, but before we exhaust
   all retry attempts and generic_file_buffered_write call trancate?
   We may get allocated blocks outside i_size. Witch is bad.

b) Commit newly allocated blocks. This approach was used after Andrey's patch.
   But if reboot comes, or error happend, user will be surprised that i_size
   increased but blocks are zero filed. This is internal write operation state
   becames visiable to user. Witch is also bad.

c) Just return as match bytes as we can deal with rigth now back to
   caller, and let's caller handles it and than call commit. In this scenario
   we never stop journal in the midle of some internal fs operation.
   If reboot comes before commit trunsaction was't closed so it will
   be droped while journal replay.

Only (c) tend to garantie attomic data operation.

Also fix some issues introduced by
 retries-in-ext3_prepare_write-violate-ordering-requirements:
  i_size may increase after error happend.
  possible data corruption caused commiting non uptodate bh.

Signed-off-by: Dmitriy Monakhov <dmonakhov@openvz.org>

```
diff --git a/fs/ext3/inode.c b/fs/ext3/inode.c
index dba6dd2..4c5e9f7 100644
--- a/fs/ext3/inode.c
+++ b/fs/ext3/inode.c
@@ -1154,6 +1154,18 @@ static int do_journal_get_write_access(handle_t *handle,
  * 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
+ * What we can do if some blocks was allocated?
+ * a) Stop transaction and do retry if possible, but what happend if
+ *    reboot comes after journal_force_commit, but before we exhaust
+ *    all retry attempts and caller call trancate?
+ *    We may get allocated blocks outside i_size. Witch is bad.
+ * b) Commit newly allocated blocks. And if reboot comes, user will be
+ *    surprised that i_size increased but blocks are zero filed. Witch is
+ *    also bad.
+ * c) Just return as match bytes as we can deal with rigth now back to
+ *    caller, and if reboot comes trunsaction was't closed so it will
+ *    be droped while journal replay.
+ * Only (c) tend to garantie attomic data operation.
  */
 static int ext3_prepare_failure(struct file *file, struct page *page,
     unsigned from, unsigned to)
@@ -1186,7 +1198,7 @@ skip:
    block_start = to;
    break;
   }
-  if (!buffer_mapped(bh))
+  if (!buffer_mapped(bh) || !buffer_uptodate(bh))
   /* prepare_write failed on this bh */
    break;
   if (ext3_should_journal_data(mapping->host)) {
@@ -1204,8 +1216,8 @@ skip:
  if (block_start <= from)
   goto skip;

- /* commit allocated and zeroed buffers */
- return mapping->a_ops->commit_write(file, page, from, block_start);
+ /* return number of bytes till last mapped && uptodate block */
+ return block_start - from;
 }

 static int ext3_prepare_write(struct file *file, struct page *page,
@@ -1239,7 +1251,8 @@ retry:

 failure:
  ret2 = ext3_prepare_failure(file, page, from, to);
- if (ret2 < 0)
+ if (ret2)
+ /* ready to partial write, or fatal error */
   return ret2;
  if (ret == -ENOSPC && ext3_should_retry_alloc(inode->i_sb, &retries))
   goto retry;
diff --git a/fs/ext4/inode.c b/fs/ext4/inode.c
index 806eee1..da39f80 100644
--- a/fs/ext4/inode.c
+++ b/fs/ext4/inode.c
@@ -1153,6 +1153,18 @@ static int do_journal_get_write_access(handle_t *handle,
  * 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
```

```
+ * What we can do if some blocks was allocated?
+ * a) Stop transaction and do retry if possible, but what happend if
+ *    reboot comes after journal_force_commit, but before we exhaust
+ *    all retry attempts and caller call trancate?
+ *    We may get allocated blocks outside i_size. Witch is bad.
+ * b) Commit newly allocated blocks. And if reboot comes, user will be
+ *    surprised that i_size increased but blocks are zero filed. Witch is
+ *    also bad.
+ * c) Just return as match bytes as we can deal with rigth now back to
+ *    caller, and if reboot comes trunsaction was't closed so it will
+ *    be droped while journal replay.
+ * Only (c) tend to garantie attomic data operation.
  */
 static int ext4_prepare_failure(struct file *file, struct page *page,
     unsigned from, unsigned to)
@@ -1185,7 +1197,7 @@ skip:
    block_start = to;
    break;
   }
-  if (!buffer_mapped(bh))
+  if (!buffer_mapped(bh) || !buffer_uptodate(bh))
   /* prepare_write failed on this bh */
    break;
   if (ext4_should_journal_data(mapping->host)) {
@@ -1203,8 +1215,8 @@ skip:
  if (block_start <= from)
   goto skip;

- /* commit allocated and zeroed buffers */
- return mapping->a_ops->commit_write(file, page, from, block_start);
+ /* return number of bytes till last mapped && uptodate block */
+ return block_start - from;
 }

 static int ext4_prepare_write(struct file *file, struct page *page,
@@ -1238,7 +1250,8 @@ retry:

 failure:
  ret2 = ext4_prepare_failure(file, page, from, to);
- if (ret2 < 0)
+ if (ret2)
+ /* ready to partial write, or fatal error */
   return ret2;
  if (ret == -ENOSPC && ext4_should_retry_alloc(inode->i_sb, &retries))
   goto retry;
```