

# [17/17] xfs: Variable order page cache support

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-04/msg08796.html>

---

- *From:* [clameter@xxxxxxx](mailto:clameter@xxxxxxx)
  - *Date:* Tue, 24 Apr 2007 15:21:22 -0700
- 

From: David Chinner <dgc@xxxxxxx>

Patch is attached that converts the XFS data path to use large order page cache pages.

I haven't tested this on a real system yet but it works on UML. I've tested it with fsx and it seems to do everything it is supposed to. Data is actually written to the block device as it persists across mount and unmount, so that appears to be working as well.

– Lets try to keep scope as small as possible.

Hence I haven't tried to convert anything on the metadata side of XFS to use the high order page cache – the XFS buffer cache takes care of that for us right now and it's not a simple change like the data path is.

Cheers,

Dave.

---

Dave Chinner  
Principal Engineer  
SGI Australian Software Group

----

```

fs/xfs/linux-2.6/xfs_aops.c | 53 ++++++-----
fs/xfs/linux-2.6/xfs_file.c | 22 ++++++
fs/xfs/linux-2.6/xfs_iops.h | 1
fs/xfs/linux-2.6/xfs_lrw.c | 6 ++--
fs/xfs/linux-2.6/xfs_super.c | 5 +++-
fs/xfs/xfs_mount.c | 13 -----
6 files changed, 58 insertions(+), 42 deletions(-)

```

Index: linux-2.6.21-rc7/fs/xfs/linux-2.6/xfs\_aops.c

=====

--- linux-2.6.21-rc7.orig/fs/xfs/linux-2.6/xfs\_aops.c 2007-04-23 22:26:17.000000000 -0700

## [17/17] xfs: Variable order page cache support

```
+++ linux-2.6.21-rc7/fs/xfs/linux-2.6/xfs_aops.c 2007-04-23 22:27:52.000000000 -0700
@@ -74,7 +74,7 @@ xfs_page_trace(
xfs_inode_t *ip;
bhv_vnode_t *vp = vn_from_inode(inode);
loff_t isize = i_size_read(inode);
- loff_t offset = page_offset(page);
+ loff_t offset = page_cache_offset(page->mapping);
int delalloc = -1, unmapped = -1, unwritten = -1;

if (page_has_buffers(page))
@@ -547,7 +547,7 @@ xfs_probe_page(
break;
} while ((bh = bh->b_this_page) != head);
} else
- ret = mapped ? 0 : PAGE_CACHE_SIZE;
+ ret = mapped ? 0 : page_cache_size(page->mapping);
}

return ret;
@@ -574,7 +574,7 @@ xfs_probe_cluster(
} while ((bh = bh->b_this_page) != head);

/* if we reached the end of the page, sum forwards in following pages */
- tlast = i_size_read(inode) >> PAGE_CACHE_SHIFT;
+ tlast = page_cache_index(inode->i_mapping, i_size_read(inode));
tindex = startpage->index + 1;

/* Prune this back to avoid pathological behavior */
@@ -592,14 +592,14 @@ xfs_probe_cluster(
size_t pg_offset, len = 0;

if (tindex == tlast) {
- pg_offset =
- i_size_read(inode) & (PAGE_CACHE_SIZE - 1);
+ pg_offset = page_cache_offset(inode->i_mapping,
+ i_size_read(inode));
if (!pg_offset) {
done = 1;
break;
}
} else
- pg_offset = PAGE_CACHE_SIZE;
+ pg_offset = page_cache_size(inode->i_mapping);

if (page->index == tindex && !TestSetPageLocked(page)) {
len = xfs_probe_page(page, pg_offset, mapped);
@@ -681,7 +681,8 @@ xfs_convert_page(
int bbits = inode->i_blkbits;
int len, page_dirty;
int count = 0, done = 0, uptodate = 1;
- xfs_off_t offset = page_offset(page);
```

## [17/17] xfs: Variable order page cache support

```
+ struct address_space *map = inode->i_mapping;
+ xfs_off_t offset = page_cache_pos(map, page->index, 0);

if (page->index != tindex)
goto fail;
@@ -689,7 +690,7 @@ xfs_convert_page(
goto fail;
if (PageWriteback(page))
goto fail_unlock_page;
- if (page->mapping != inode->i_mapping)
+ if (page->mapping != map)
goto fail_unlock_page;
if (!xfs_is_delayed_page(page, (*ioendp)->io_type))
goto fail_unlock_page;
@@ -701,20 +702,20 @@ xfs_convert_page(
* Derivation:
*
* End offset is the highest offset that this page should represent.
- * If we are on the last page, (end_offset & (PAGE_CACHE_SIZE - 1))
- * will evaluate non-zero and be less than PAGE_CACHE_SIZE and
+ * If we are on the last page, (end_offset & page_cache_mask())
+ * will evaluate non-zero and be less than page_cache_size() and
* hence give us the correct page_dirty count. On any other page,
* it will be zero and in that case we need page_dirty to be the
* count of buffers on the page.
*/
end_offset = min_t(unsigned long long,
- (xfs_off_t)(page->index + 1) << PAGE_CACHE_SHIFT,
+ (xfs_off_t)(page->index + 1) << page_cache_shift(map),
i_size_read(inode));

len = 1 << inode->i_blkbits;
- p_offset = min_t(unsigned long, end_offset & (PAGE_CACHE_SIZE - 1),
- PAGE_CACHE_SIZE);
- p_offset = p_offset ? roundup(p_offset, len) : PAGE_CACHE_SIZE;
+ p_offset = min_t(unsigned long, page_cache_offset(map, end_offset),
+ page_cache_size(map));
+ p_offset = p_offset ? roundup(p_offset, len) : page_cache_size(map);
page_dirty = p_offset / len;

bh = head = page_buffers(page);
@@ -870,6 +871,7 @@ xfs_page_state_convert(
int page_dirty, count = 0;
int trylock = 0;
int all_bh = unmapped;
+ struct address_space *map = inode->i_mapping;

if (startio) {
if (wbc->sync_mode == WB_SYNC_NONE && wbc->nonblocking)
@@ -878,11 +880,11 @@ xfs_page_state_convert(
```

```

/* Is this page beyond the end of the file? */
offset = i_size_read(inode);
- end_index = offset >> PAGE_CACHE_SHIFT;
- last_index = (offset - 1) >> PAGE_CACHE_SHIFT;
+ end_index = page_cache_index(map, offset);
+ last_index = page_cache_index(map, (offset - 1));
if (page->index >= end_index) {
if ((page->index >= end_index + 1) ||
- !(i_size_read(inode) & (PAGE_CACHE_SIZE - 1))) {
+ !(page_cache_offset(map, i_size_read(inode)))) {
if (startio)
unlock_page(page);
return 0;
@@ -896,22 +898,23 @@ xfs_page_state_convert(
* Derivation:
*
* End offset is the highest offset that this page should represent.
- * If we are on the last page, (end_offset & (PAGE_CACHE_SIZE - 1))
- * will evaluate non-zero and be less than PAGE_CACHE_SIZE and
+ * If we are on the last page, (end_offset & page_cache_mask())
+ * will evaluate non-zero and be less than page_cache_size() and
* hence give us the correct page_dirty count. On any other page,
* it will be zero and in that case we need page_dirty to be the
* count of buffers on the page.
*/
end_offset = min_t(unsigned long long,
- (xfs_off_t)(page->index + 1) << PAGE_CACHE_SHIFT, offset);
+ (xfs_off_t)(page->index + 1) << page_cache_shift(map),
+ offset);
len = 1 << inode->i_blkbits;
- p_offset = min_t(unsigned long, end_offset & (PAGE_CACHE_SIZE - 1),
- PAGE_CACHE_SIZE);
- p_offset = p_offset ? roundup(p_offset, len) : PAGE_CACHE_SIZE;
+ p_offset = min_t(unsigned long, page_cache_offset(map, end_offset),
+ page_cache_size(map));
+ p_offset = p_offset ? roundup(p_offset, len) : page_cache_size(map);
page_dirty = p_offset / len;

bh = head = page_buffers(page);
- offset = page_offset(page);
+ offset = page_cache_pos(map, page->index, 0);
flags = -1;
type = 0;

@@ -1040,7 +1043,7 @@ xfs_page_state_convert(

if (ioend && iomap_valid) {
offset = (iomap.iomap_offset + iomap.iomap_bsize - 1) >>
- PAGE_CACHE_SHIFT;
+ page_cache_shift(map);
tlast = min_t(pgoff_t, offset, last_index);

```

[17/17] xfs: Variable order page cache support

```
xfs_cluster_write(inode, page->index + 1, &iomap, &ioend,  
wbc, startio, all_bh, tlast);
```

```
Index: linux-2.6.21-rc7/fs/xfs/linux-2.6/xfs_lrw.c
```

```
=====  
--- linux-2.6.21-rc7.orig/fs/xfs/linux-2.6/xfs_lrw.c 2007-04-23 22:26:17.000000000 -0700  
+++ linux-2.6.21-rc7/fs/xfs/linux-2.6/xfs_lrw.c 2007-04-23 22:27:52.000000000 -0700  
@@ -143,9 +143,9 @@ xfs_iozero(  
do {  
    unsigned long index, offset;  
  
    - offset = (pos & (PAGE_CACHE_SIZE - 1)); /* Within page */  
    - index = pos >> PAGE_CACHE_SHIFT;  
    - bytes = PAGE_CACHE_SIZE - offset;  
    + offset = page_cache_offset(mapping, pos); /* Within page */  
    + index = page_cache_index(mapping, pos);  
    + bytes = page_cache_size(mapping) - offset;  
    if (bytes > count)  
        bytes = count;
```

```
Index: linux-2.6.21-rc7/fs/xfs/linux-2.6/xfs_file.c
```

```
=====  
--- linux-2.6.21-rc7.orig/fs/xfs/linux-2.6/xfs_file.c 2007-04-23 22:26:17.000000000 -0700  
+++ linux-2.6.21-rc7/fs/xfs/linux-2.6/xfs_file.c 2007-04-23 22:27:52.000000000 -0700  
@@ -469,6 +469,28 @@ const struct file_operations xfs_file_op  
#endif  
};  
  
+const struct file_operations xfs_no_mmap_file_operations = {  
+ .llseek = generic_file_llseek,  
+ .read = do_sync_read,  
+ .write = do_sync_write,  
+ .aio_read = xfs_file_aio_read,  
+ .aio_write = xfs_file_aio_write,  
+ .sendfile = xfs_file_sendfile,  
+ .splice_read = xfs_file_splice_read,  
+ .splice_write = xfs_file_splice_write,  
+ .unlocked_ioctl = xfs_file_ioctl,  
+ #ifdef CONFIG_COMPAT  
+ .compat_ioctl = xfs_file_compat_ioctl,  
+ #endif  
+ .open = xfs_file_open,  
+ .flush = xfs_file_close,  
+ .release = xfs_file_release,  
+ .fsync = xfs_file_fsync,  
+ #ifdef HAVE_FOP_OPEN_EXEC  
+ .open_exec = xfs_file_open_exec,  
+ #endif  
+};  
+  
const struct file_operations xfs_invis_file_operations = {  
    .llseek = generic_file_llseek,
```

## [17/17] xfs: Variable order page cache support

```
.read = do_sync_read,
```

```
Index: linux-2.6.21-rc7/fs/xfs/linux-2.6/xfs_iops.h
```

```
----- linux-2.6.21-rc7.orig/fs/xfs/linux-2.6/xfs_iops.h 2007-04-23 22:26:17.000000000 -0700  
+++ linux-2.6.21-rc7/fs/xfs/linux-2.6/xfs_iops.h 2007-04-23 22:27:52.000000000 -0700
```

```
@@ -23,6 +23,7 @@ extern const struct inode_operations xfs  
extern const struct inode_operations xfs_symlink_inode_operations;
```

```
extern const struct file_operations xfs_file_operations;  
+extern const struct file_operations xfs_no_mmap_file_operations;  
extern const struct file_operations xfs_dir_file_operations;  
extern const struct file_operations xfs_invis_file_operations;
```

```
Index: linux-2.6.21-rc7/fs/xfs/linux-2.6/xfs_super.c
```

```
----- linux-2.6.21-rc7.orig/fs/xfs/linux-2.6/xfs_super.c 2007-04-23 22:26:17.000000000 -0700  
+++ linux-2.6.21-rc7/fs/xfs/linux-2.6/xfs_super.c 2007-04-23 22:34:50.000000000 -0700
```

```
@@ -125,8 +125,11 @@ xfs_set_inodeops(  
{  
switch (inode->i_mode & S_IFMT) {  
case S_IFREG:  
+ if (mapping_order(inode->i_mapping))  
+ inode->i_fop = &xfs_no_mmap_file_operations;  
+ else  
+ inode->i_fop = &xfs_file_operations;  
inode->i_op = &xfs_inode_operations;  
- inode->i_fop = &xfs_file_operations;  
inode->i_mapping->a_ops = &xfs_address_space_operations;  
break;  
case S_IFDIR:
```

```
Index: linux-2.6.21-rc7/fs/xfs/xfs_mount.c
```

```
----- linux-2.6.21-rc7.orig/fs/xfs/xfs_mount.c 2007-04-23 22:26:17.000000000 -0700  
+++ linux-2.6.21-rc7/fs/xfs/xfs_mount.c 2007-04-23 22:27:52.000000000 -0700
```

```
@@ -315,19 +315,6 @@ xfs_mount_validate_sb(  
return XFS_ERROR(ENOSYS);  
}
```

```
- /*  
- * Until this is fixed only page-sized or smaller data blocks work.  
- */  
- if (unlikely(sbp->sb_blocksize > PAGE_SIZE)) {  
- xfs_fs_mount_cmn_err(flags,  
- "file system with blocksize %d bytes",  
- sbp->sb_blocksize);  
- xfs_fs_mount_cmn_err(flags,  
- "only pagesize (%ld) or less will currently work.",  
- PAGE_SIZE);  
- return XFS_ERROR(ENOSYS);  
- }  
-
```

```
return 0;  
}
```

--

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in  
the body of a message to majordomo@xxxxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

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