Subject: [PATCH] incorrect direct io error handling
Posted by Dmitriy Monakhov on Mon, 18 Dec 2006 13:22:44 GMT
View Forum Message <> Reply to Message

This patch is result of discussion started week ago here:
http://lkml.org/lkml/2006/12/11/66
changes from original patch:
 - Update wrong comments about i_mutex locking.
 - Add BUG_ON(!mutex_is_locked(..)) for non blkdev.
 - vmtruncate call only for non blockdev
LOG:
If generic_file_direct_write() has fail (ENOSPC condition) inside
__generic_file_aio_write_nolock() it may have instantiated
a few blocks outside i_size. And fsck will complain about wrong i_size
(ext2, ext3 and reiserfs interpret i_size and biggest block difference as error),
after fsck will fix error i_size will be increased to the biggest block,
but this blocks contain gurbage from previous write attempt, this is not
information leak, but its silence file data corruption. This issue affect
fs regardless the values of blocksize or pagesize.
We need truncate any block beyond i_size after write have failed , do in simular
generic_file_buffered_write() error path. If host is !S_ISBLK i_mutex always
held inside generic_file_aio_write_nolock() and we may safely call vmtruncate().
Some fs (XFS at least) may directly call generic_file_direct_write()with
i_mutex not held. There is no general scenario in this case. This fs have to
handle generic_file_direct_write() error by its own specific way (place).

Issue was found during OpenVZ kernel testing.

Exampe:
open("mnt2/FILE3", O_WRONLY|O_CREAT|O_DIRECT, 0666) = 3
write(3, "aaaaaa"..., 4096) = -1 ENOSPC (No space left on device)

stat mnt2/FILE3
File: `mnt2/FILE3'
Size: 0 Blocks: 4 IO Block: 4096 regular empty file
>>>>>>>>>>>>>>>>>>>>>>^^^^^^^^^^ file size is less than biggest block idx
Device: 700h/1792d Inode: 14 Links: 1
Access: (0644/-rw-r--r--) Uid: (0/ root) Gid: (0/ root)

fsck.ext2 -f -n mnt1/fs_img
Pass 1: Checking inodes, blocks, and sizes
Inode 14, i_size is 0, should be 2048. Fix? no

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

diff --git a/mm/filemap.c b/mm/filemap.c
index 8332c77..7c571dd 100644

```
--- a/mm/filemap.c
+++ b/mm/filemap.c
@@ -2044,8 +2044,9 @@ generic_file_direct_write(struct kiocb *
  /*
   * Sync the fs metadata but not the minor inode changes and
   * of course not the data as we did direct DMA for the IO.
-  * i_mutex is held, which protects generic_osync_inode() from
-  * livelocking.  AIO O_DIRECT ops attempt to sync metadata here.
+  * i_mutex may not being held (XFS does this), if so some specific locking
+  * ordering must protect generic_osync_inode() from livelocking.
+  * AIO O_DIRECT ops attempt to sync metadata here.
   */
  if ((written >= 0 || written == -EIOCBQUEUED) &&
      ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
@@ -2279,6 +2280,17 @@ __generic_file_aio_write_nolock(struct k

   written = generic_file_direct_write(iocb, iov, &nr_segs, pos,
        ppos, count, ocount);
+  /*
+   * If host is not S_ISBLK generic_file_direct_write() may
+   * have instantiated a few blocks outside i_size  files
+   * Trim these off again.
+   */
+  if (unlikely(written < 0) && !S_ISBLK(inode->i_mode)) {
+   loff_t isize = i_size_read(inode);
+   if (pos + count > isize)
+    vmtruncate(inode, isize);
+  }
+
   if (written < 0 || written == count)
    goto out;
   /*
@@ -2341,6 +2353,13 @@ ssize_t generic_file_aio_write_nolock(st
  ssize_t ret;

  BUG_ON(iocb->ki_pos != pos);
+ /*
+  *  generic_file_buffered_write() may be called inside
+  *  __generic_file_aio_write_nolock() even in case of
+  *  O_DIRECT for non S_ISBLK files. So i_mutex must be held.
+  */
+ if (!S_ISBLK(inode->i_mode))
+  BUG_ON(!mutex_is_locked(&inode->i_mutex));

  ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
    &iocb->ki_pos);
@@ -2383,8 +2402,8 @@ ssize_t generic_file_aio_write(struct ki
 EXPORT_SYMBOL(generic_file_aio_write);
```

```
 /*
- * Called under i_mutex for writes to S_ISREG files.   Returns -EIO if something
- * went wrong during pagecache shootdown.
+ * May be called without i_mutex for writes to S_ISREG files. XFS does this.
+ * Returns -EIO if something went wrong during pagecache shootdown.
  */
 static ssize_t
 generic_file_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
```

Subject: RE: [PATCH] incorrect direct io error handling
Posted by kenneth.w.chen on Mon, 18 Dec 2006 19:56:36 GMT
View Forum Message <> Reply to Message

Dmitriy Monakhov wrote on Monday, December 18, 2006 5:23 AM
> This patch is result of discussion started week ago here:
> http://lkml.org/lkml/2006/12/11/66
> changes from original patch:
> - Update wrong comments about i_mutex locking.
> - Add BUG_ON(!mutex_is_locked(..)) for non blkdev.
> - vmtruncate call only for non blockdev
> LOG:
> If generic_file_direct_write() has fail (ENOSPC condition) inside
> __generic_file_aio_write_nolock() it may have instantiated
> a few blocks outside i_size. And fsck will complain about wrong i_size
> (ext2, ext3 and reiserfs interpret i_size and biggest block difference as error),
> after fsck will fix error i_size will be increased to the biggest block,
> but this blocks contain gurbage from previous write attempt, this is not
> information leak, but its silence file data corruption. This issue affect
> fs regardless the values of blocksize or pagesize.
> We need truncate any block beyond i_size after write have failed , do in simular
> generic_file_buffered_write() error path. If host is !S_ISBLK i_mutex always
> held inside generic_file_aio_write_nolock() and we may safely call vmtruncate().
> Some fs (XFS at least) may directly call generic_file_direct_write()with
> i_mutex not held. There is no general scenario in this case. This fs have to
> handle generic_file_direct_write() error by its own specific way (place).

I'm puzzled that if ext2 is able to instantiate some blocks, then why does it
return no space error? Where is the error coming from?

Subject: Re: [PATCH] incorrect direct io error handling
Posted by David Chinner on Mon, 18 Dec 2006 22:15:15 GMT
View Forum Message <> Reply to Message

On Mon, Dec 18, 2006 at 04:22:44PM +0300, Dmitriy Monakhov wrote:
> diff --git a/mm/filemap.c b/mm/filemap.c
> index 8332c77..7c571dd 100644
> --- a/mm/filemap.c
> +++ b/mm/filemap.c
> @@ -2044,8 +2044,9 @@ generic_file_direct_write(struct kiocb *
> /*
> * Sync the fs metadata but not the minor inode changes and
> * of course not the data as we did direct DMA for the IO.
> - * i_mutex is held, which protects generic_osync_inode() from
> - * livelocking. AIO O_DIRECT ops attempt to sync metadata here.
> + * i_mutex may not being held (XFS does this), if so some specific locking
> + * ordering must protect generic_osync_inode() from livelocking.
> + * AIO O_DIRECT ops attempt to sync metadata here.
> */
> if ((written >= 0 || written == -EIOCBQUEUED) &&
> ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
> @@ -2279,6 +2280,17 @@ __generic_file_aio_write_nolock(struct k
>
> written = generic_file_direct_write(iocb, iov, &nr_segs, pos,
> ppos, count, ocount);
> + /*
> + * If host is not S_ISBLK generic_file_direct_write() may
> + * have instantiated a few blocks outside i_size files
> + * Trim these off again.
> + */
> + if (unlikely(written < 0) && !S_ISBLK(inode->i_mode)) {
> + loff_t isize = i_size_read(inode);
> + if (pos + count > isize)
> + vmtruncate(inode, isize);
> + }
> +
> if (written < 0 || written == count)
> goto out;

You comment in the first hunk that i_mutex may not be held here,
but there's no comment in __generic_file_aio_write_nolock() that the
i_mutex must be held for !S_ISBLK devices.

> @@ -2341,6 +2353,13 @@ ssize_t generic_file_aio_write_nolock(st
> ssize_t ret;
>
> BUG_ON(iocb->ki_pos != pos);
> + /*
> + * generic_file_buffered_write() may be called inside
> + * __generic_file_aio_write_nolock() even in case of
> + * O_DIRECT for non S_ISBLK files. So i_mutex must be held.
> + */

> + if (!S_ISBLK(inode->i_mode))
> + BUG_ON(!mutex_is_locked(&inode->i_mutex));
>
>   ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
>   &iocb->ki_pos);

I note that you comment here in generic_file_aio_write_nolock(),
but it's not immediately obvious that this is refering to the
vmtruncate() call in __generic_file_aio_write_nolock().

IOWs, wouldn't it be better to put this comment and check in
__generic_file_aio_write_nolock() directly above the vmtruncate()
call that cares about this?

> @@ -2383,8 +2402,8 @@ ssize_t generic_file_aio_write(struct ki
> EXPORT_SYMBOL(generic_file_aio_write);
>
> /*
> - * Called under i_mutex for writes to S_ISREG files.   Returns -EIO if something
> - * went wrong during pagecache shootdown.
> + * May be called without i_mutex for writes to S_ISREG files. XFS does this.
> + * Returns -EIO if something went wrong during pagecache shootdown.
>   */

Not sure you need to say "XFS does this" - other filesystems may do this
in the future.....

Cheers,

Dave.
--
Dave Chinner
Principal Engineer
SGI Australian Software Group

Subject: Re: [PATCH] incorrect direct io error handling
Posted by Dmitriy Monakhov on Tue, 19 Dec 2006 06:07:12 GMT
View Forum Message <> Reply to Message

David Chinner <dgc@sgi.com> writes:

> On Mon, Dec 18, 2006 at 04:22:44PM +0300, Dmitriy Monakhov wrote:
>> diff --git a/mm/filemap.c b/mm/filemap.c
>> index 8332c77..7c571dd 100644
>> --- a/mm/filemap.c
>> +++ b/mm/filemap.c
>> @@ -2044,8 +2044,9 @@ generic_file_direct_write(struct kiocb *

```
>>   /*
>>    * Sync the fs metadata but not the minor inode changes and
>>    * of course not the data as we did direct DMA for the IO.
>> -  * i_mutex is held, which protects generic_osync_inode() from
>> -  * livelocking.  AIO O_DIRECT ops attempt to sync metadata here.
>> +  * i_mutex may not being held (XFS does this), if so some specific locking
>> +  * ordering must protect generic_osync_inode() from livelocking.
>> +  * AIO O_DIRECT ops attempt to sync metadata here.
>>    */
>>   if ((written >= 0 || written == -EIOCBQUEUED) &&
>>       ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
>> @@ -2279,6 +2280,17 @@ __generic_file_aio_write_nolock(struct k
>>
>>    written = generic_file_direct_write(iocb, iov, &nr_segs, pos,
>>         ppos, count, ocount);
>> +  /*
>> +   * If host is not S_ISBLK generic_file_direct_write() may
>> +   * have instantiated a few blocks outside i_size  files
>> +   * Trim these off again.
>> +   */
>> +  if (unlikely(written < 0) && !S_ISBLK(inode->i_mode)) {
>> +   loff_t isize = i_size_read(inode);
>> +   if (pos + count > isize)
>> +    vmtruncate(inode, isize);
>> +  }
>> +
>>    if (written < 0 || written == count)
>>     goto out;
>
> You comment in the first hunk that i_mutex may not be held here,
> but there's no comment in __generic_file_aio_write_nolock() that the
> i_mutex must be held for !S_ISBLK devices.
Any one may call directly call generic_file_direct_write() with i_mutex not held.
>
>> @@ -2341,6 +2353,13 @@ ssize_t generic_file_aio_write_nolock(st
>>   ssize_t ret;
>>
>>   BUG_ON(iocb->ki_pos != pos);
>> + /*
>> +  *  generic_file_buffered_write() may be called inside
>> +  *  __generic_file_aio_write_nolock() even in case of
>> +  *  O_DIRECT for non S_ISBLK files. So i_mutex must be held.
>> +  */
>> + if (!S_ISBLK(inode->i_mode))
>> +  BUG_ON(!mutex_is_locked(&inode->i_mutex));
>>
>>   ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
>>     &iocb->ki_pos);
```

>
> I note that you comment here in generic_file_aio_write_nolock(),
> but it's not immediately obvious that this is refering to the
> vmtruncate() call in __generic_file_aio_write_nolock().
This is not about vmtruncate(). __generic_file_aio_write_nolock() may
call generic_file_buffered_write() even in case of O_DIRECT for !S_ISBLK, and
generic_file_buffered_write() has documented locking rules (i_mutex held).
IMHO it is important to explicitly document this . And after we realize
that i_mutex always held, vmtruncate() may be safely called.
>
> IOWs, wouldn't it be better to put this comment and check in
> __generic_file_aio_write_nolock() directly above the vmtruncate()
> call that cares about this?
>
>> @@ -2383,8 +2402,8 @@ ssize_t generic_file_aio_write(struct ki
>> EXPORT_SYMBOL(generic_file_aio_write);
>>
>> /*
>> - * Called under i_mutex for writes to S_ISREG files.   Returns -EIO if something
>> - * went wrong during pagecache shootdown.
>> + * May be called without i_mutex for writes to S_ISREG files. XFS does this.
>> + * Returns -EIO if something went wrong during pagecache shootdown.
>>   */
>
> Not sure you need to say "XFS does this" - other filesystems may do this
> in the future.....
Yes, but where are multiple comments about "reiserfs does this" in fs/buffer.c

>
> Cheers,
>
> Dave.
> --
> Dave Chinner
> Principal Engineer
> SGI Australian Software Group
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/

Subject: Re: [PATCH] incorrect direct io error handling
Posted by Dmitriy Monakhov on Tue, 19 Dec 2006 06:31:15 GMT
View Forum Message <> Reply to Message

"Chen, Kenneth W" <kenneth.w.chen@intel.com> writes:

> Dmitriy Monakhov wrote on Monday, December 18, 2006 5:23 AM
>> This patch is result of discussion started week ago here:
>> http://lkml.org/lkml/2006/12/11/66
>> changes from original patch:
>> - Update wrong comments about i_mutex locking.
>> - Add BUG_ON(!mutex_is_locked(..)) for non blkdev.
>> - vmtruncate call only for non blockdev
>> LOG:
>> If generic_file_direct_write() has fail (ENOSPC condition) inside
>> __generic_file_aio_write_nolock() it may have instantiated
>> a few blocks outside i_size. And fsck will complain about wrong i_size
>> (ext2, ext3 and reiserfs interpret i_size and biggest block difference as error),
>> after fsck will fix error i_size will be increased to the biggest block,
>> but this blocks contain gurbage from previous write attempt, this is not
>> information leak, but its silence file data corruption. This issue affect
>> fs regardless the values of blocksize or pagesize.
>> We need truncate any block beyond i_size after write have failed , do in simular
>> generic_file_buffered_write() error path. If host is !S_ISBLK i_mutex always
>> held inside generic_file_aio_write_nolock() and we may safely call vmtruncate().
>> Some fs (XFS at least) may directly call generic_file_direct_write()with
>> i_mutex not held. There is no general scenario in this case. This fs have to
>> handle generic_file_direct_write() error by its own specific way (place).
>
>
> I'm puzzled that if ext2 is able to instantiate some blocks, then why does it
> return no space error? Where is the error coming from?
generic_file_aio_write_nolock()
 ->generic_file_direct_write()
   ->generic_file_direct_IO()
     ->ext2_direct_IO(WRITE,...)
       ->blockdev_direct_IO(....,ext2_get_block,...)

Subject: Re: [PATCH] incorrect direct io error handling
Posted by David Chinner on Wed, 20 Dec 2006 14:26:31 GMT
View Forum Message <> Reply to Message

On Tue, Dec 19, 2006 at 09:07:12AM +0300, Dmitriy Monakhov wrote:
> David Chinner <dgc@sgi.com> writes:
> > On Mon, Dec 18, 2006 at 04:22:44PM +0300, Dmitriy Monakhov wrote:
> >> diff --git a/mm/filemap.c b/mm/filemap.c
> >> index 8332c77..7c571dd 100644
> >> --- a/mm/filemap.c
> >> +++ b/mm/filemap.c

<snip stuff>

> > You comment in the first hunk that i_mutex may not be held here,
> > but there's no comment in __generic_file_aio_write_nolock() that the
> > i_mutex must be held for !S_ISBLK devices.
> Any one may call directly call generic_file_direct_write() with i_mutex not held.

Only block devices based on the implementation (i.e. buffered I/O is
done here). but one can't call vmtruncate without the i_mutex held,
so if a filesystem is calling generic_file_direct_write() it won't
be able to use __generic_file_aio_write_nolock() without the i_mutex
held (because it can right now if it doesn't need the buffered I/O
fallback path), then

> >
> >> @@ -2341,6 +2353,13 @@ ssize_t generic_file_aio_write_nolock(st
> >> ssize_t ret;
> >>
> >> BUG_ON(iocb->ki_pos != pos);
> >> + /*
> >> + * generic_file_buffered_write() may be called inside
> >> + * __generic_file_aio_write_nolock() even in case of
> >> + * O_DIRECT for non S_ISBLK files. So i_mutex must be held.
> >> + */
> >> + if (!S_ISBLK(inode->i_mode))
> >> + BUG_ON(!mutex_is_locked(&inode->i_mutex));
> >>
> >> ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
> >> &iocb->ki_pos);
> >
> > I note that you comment here in generic_file_aio_write_nolock(),
> > but it's not immediately obvious that this is refering to the
> > vmtruncate() call in __generic_file_aio_write_nolock().
> This is not about vmtruncate(). __generic_file_aio_write_nolock() may
> call generic_file_buffered_write() even in case of O_DIRECT for !S_ISBLK, and

No, the need for i_mutex is currently dependent on doing direct I/O
and the return value from generic_file_buffered_write().
A filesystem that doesn't fall back to buffered I/O (e.g. XFS) can currently
use generic_file_aio_write_nolock() without needing to hold i_mutex.

Your change prevents that by introducing a vmtruncate() before the
generic_file_buffered_write() return value check, which means that a
filesystem now _must_ hold the i_mutex when calling
generic_file_aio_write_nolock() even when it doesn't do buffered I/O
through this path.

> generic_file_buffered_write() has documented locking rules (i_mutex held).
> IMHO it is important to explicitly document this . And after we realize
> that i_mutex always held, vmtruncate() may be safely called.

I don't think changing the locking semantics of
generic_file_aio_write_nolock() to require a lock for all
filesystem-based users is a good way to fix a filesystem specific
direct I/O problem which can be easily fixed in filesystem specific
code - i.e. call vmtruncate() in ext3_file_write() on failure....

Cheers,

Dave.
--
Dave Chinner
Principal Engineer
SGI Australian Software Group

Subject: Re: [PATCH] incorrect direct io error handling
Posted by Dmitriy Monakhov on Wed, 10 Jan 2007 14:36:57 GMT
View Forum Message <> Reply to Message

Sorry for long delay (russian holidays are very hard time :))

David Chinner <dgc@sgi.com> writes:
> On Tue, Dec 19, 2006 at 09:07:12AM +0300, Dmitriy Monakhov wrote:
>> David Chinner <dgc@sgi.com> writes:
>> > On Mon, Dec 18, 2006 at 04:22:44PM +0300, Dmitriy Monakhov wrote:
>> >> diff --git a/mm/filemap.c b/mm/filemap.c
>> >> index 8332c77..7c571dd 100644
>> >> --- a/mm/filemap.c
>> >> +++ b/mm/filemap.c
>
> <snip stuff>
>
>> > You comment in the first hunk that i_mutex may not be held here,
>> > but there's no comment in __generic_file_aio_write_nolock() that the
>> > i_mutex must be held for !S_ISBLK devices.
>> Any one may call directly call generic_file_direct_write() with i_mutex not held.
>
> Only block devices based on the implementation (i.e. buffered I/O is
> done here). but one can't call vmtruncate without the i_mutex held,
> so if a filesystem is calling generic_file_direct_write() it won't
> be able to use __generic_file_aio_write_nolock() without the i_mutex
> held (because it can right now if it doesn't need the buffered I/O
> fallback path), then
>
>> >
>> >> @@ -2341,6 +2353,13 @@ ssize_t generic_file_aio_write_nolock(st
>> >>   ssize_t ret;

>> >>
>> >>   BUG_ON(iocb->ki_pos != pos);
>> >> + /*
>> >> + * generic_file_buffered_write() may be called inside
>> >> + * __generic_file_aio_write_nolock() even in case of
>> >> + * O_DIRECT for non S_ISBLK files. So i_mutex must be held.
>> >> + */
>> >> + if (!S_ISBLK(inode->i_mode))
>> >> + BUG_ON(!mutex_is_locked(&inode->i_mutex));
>> >>
>> >>   ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
>> >>   &iocb->ki_pos);
>> >
>> > I note that you comment here in generic_file_aio_write_nolock(),
>> > but it's not immediately obvious that this is refering to the
>> > vmtruncate() call in __generic_file_aio_write_nolock().
>> This is not about vmtruncate(). __generic_file_aio_write_nolock() may
>> call generic_file_buffered_write() even in case of O_DIRECT for !S_ISBLK, and
>
> No, the need for i_mutex is currently dependent on doing direct I/O
> and the return value from generic_file_buffered_write().
> A filesystem that doesn't fall back to buffered I/O (e.g. XFS) can currently
> use generic_file_aio_write_nolock() without needing to hold i_mutex.
> use generic_file_aio_write_nolock() without needing to hold i_mutex.
But it doesn't use it. XFS implement it's own write method with it's own locking
rules and explicitly call generic_file_direct_write() in case of O_DIRECT.
BTW XFS correctly handling ENOSPC in case of O_DIRECT (fs corruption not happend
after error occur).

>
> Your change prevents that by introducing a vmtruncate() before the
> generic_file_buffered_write() return value check, which means that a
> filesystem now _must_ hold the i_mutex when calling
> generic_file_aio_write_nolock() even when it doesn't do buffered I/O
> through this path.
Yes it's so. But it is just explicitly document the fact that every fs call
generic_file_aio_write_nolock() with i_mutex held (where is no any fs that
invoke it without i_mutex). As i understand Andrew Morton think so too:
http://lkml.org/lkml/2006/12/12/67
<snip>
  I guess we can make that a rule (document it, add
  BUG_ON(!mutex_is_locked(..)) if it isn't a blockdev) if needs be. After
  really checking that this matches reality for all callers.
<snip>

>
>> generic_file_buffered_write() has documented locking rules (i_mutex held).
>> IMHO it is important to explicitly document this . And after we realize

>> that i_mutex always held, vmtruncate() may be safely called.
>
> I don't think changing the locking semantics of
> generic_file_aio_write_nolock() to require a lock for all
> filesystem-based users is a good way to fix a filesystem specific
> direct I/O problem which can be easily fixed in filesystem specific
> code - i.e. call vmtruncate() in ext3_file_write() on failure....
Where are more than 10 filesystems where we have to fix it then.
And fix is almost the same for all fs, so we have to do many copy/paste work
IMHO fix it inside generic_file_aio_write_nolock is realy straightforward way.
What do you think?
>
> Cheers,
>
> Dave.
> --
> Dave Chinner
> Principal Engineer
> SGI Australian Software Group