Subject: [PATCH 0/4] fuse: optimize scatter-gather direct IO
Posted by Maxim Patlasov on Fri, 20 Jul 2012 11:50:07 GMT
View Forum Message <> Reply to Message

Hi,

Existing fuse implementation processes scatter-gather direct IO in suboptimal way: fuse_direct_IO passes iovec[] to fuse_loop_dio and the latter calls fuse_direct_read/write for each iovec from iovec[] array. Thus we have as many submitted fuse-requests as the number of elements in iovec[] array. This is pure waste of resources and affects performance negatively especially for the case of many small chunks (e.g. page-size) packed in one iovec[] array.

The patch-set amends situation in a natural way: let's simply pack as many iovec[] segments to every fuse-request as possible.

To estimate performance improvement I used slightly modified fusexmp over tmpfs (clearing O_DIRECT bit from fi->flags in xmp_open). The test opened a file with O_DIRECT, then called readv/writev in a loop. An iovec[] for readv/writev consisted of 32 segments of 4K each. The throughput on some commodity (rather feeble) server was (in MB/sec):

```
        original / patched
writev: ~107     / ~480
readv:  ~114     / ~569
```

We're exploring possiblity to use fuse for our own distributed storage implementation and big iovec[] arrays of many page-size chunks is typical use-case for device virtualization thread performing i/o on behalf of virtual-machine it serves.

Thanks,
Maxim

Maxim Patlasov (4):
      fuse: add basic support of iovec[] to fuse_req
      fuse: re-work fuse_get_user_pages() to operate on iovec[]
      fuse: re-work fuse_direct_io() to operate on iovec[]
      fuse: re-work fuse_direct_IO()

```
 fs/fuse/dev.c    |   52 +++++++++++++++++++-
 fs/fuse/file.c   |  145 +++++++++++++++++++++++++++++++++++++++----------------------
 fs/fuse/fuse_i.h |   12 ++++
 3 files changed, 140 insertions(+), 69 deletions(-)
```

Subject: [PATCH 1/4] fuse: add basic support of iovec[] to fuse_req
Posted by Maxim Patlasov on Fri, 20 Jul 2012 11:50:23 GMT
View Forum Message <> Reply to Message

The patch allows fuse_req to refer to array of iovec-s describing
layout of user-data over req->pages. fuse_copy_pages() is re-worked to
support both cased: former layout where pages[] corresponded to <buf, len>
and newer one where pages[] corresponds to iovec[].

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
---
 fs/fuse/dev.c | 52 ++---
 fs/fuse/fuse_i.h | 12 ++++++++++--
 2 files changed, 59 insertions(+), 5 deletions(-)

diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
index 7df2b5e..cdae525 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
@@ -850,9 +850,9 @@ static int fuse_copy_page(struct fuse_copy_state *cs, struct page
**pagep,
  return 0;
 }

-/* Copy pages in the request to/from userspace buffer */
-static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned nbytes,
-      int zeroing)
+/* Start from addr(pages[0]) + page_offset. No holes in the middle. */
+static int fuse_copy_pages_for_buf(struct fuse_copy_state *cs, unsigned nbytes,
+   int zeroing)
 {
  unsigned i;
  struct fuse_req *req = cs->req;
@@ -874,6 +874,52 @@ static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned
nbytes,
  return 0;
 }

+/* Take iov_offset as offset in iovec[0]. Iterate based on iovec[].iov_len */
+static int fuse_copy_pages_for_iovec(struct fuse_copy_state *cs,
+   unsigned nbytes, int zeroing)
+{
+ unsigned i;
+ struct fuse_req *req = cs->req;
+ const struct iovec *iov = req->iovec;
+ unsigned iov_offset = req->iov_offset;
+
+ for (i = 0; i < req->num_pages && (nbytes || zeroing); i++) {
+ int err;

```
+  unsigned long user_addr = (unsigned long)iov->iov_base +
+       iov_offset;
+  unsigned offset = user_addr & ~PAGE_MASK;
+  unsigned count = min_t(size_t, PAGE_SIZE - offset,
+           iov->iov_len - iov_offset);
+  count = min(nbytes, count);
+
+  err = fuse_copy_page(cs, &req->pages[i], offset, count,
+         zeroing);
+  if (err)
+   return err;
+
+  nbytes -= count;
+
+  if (count < iov->iov_len - iov_offset) {
+   iov_offset += count;
+  } else {
+   iov++;
+   iov_offset = 0;
+  }
+ }
+
+ return 0;
+}
+
+/* Copy pages in the request to/from userspace buffer */
+static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned nbytes,
+       int zeroing)
+{
+ if (cs->req->iovec)
+  return fuse_copy_pages_for_iovec(cs, nbytes, zeroing);
+ else
+  return fuse_copy_pages_for_buf(cs, nbytes, zeroing);
+}
+
 /* Copy a single argument in the request to/from userspace buffer */
 static int fuse_copy_one(struct fuse_copy_state *cs, void *val, unsigned size)
 {
diff --git a/fs/fuse/fuse_i.h b/fs/fuse/fuse_i.h
index 771fb63..255b7cd 100644
--- a/fs/fuse/fuse_i.h
+++ b/fs/fuse/fuse_i.h
@@ -296,8 +296,16 @@ struct fuse_req {
  /** number of pages in vector */
  unsigned num_pages;

- /** offset of data on first page */
- unsigned page_offset;
```

```
+ /** If set, it describes layout of user-data in pages[] */
+ const struct iovec *iovec;
+
+ union {
+  /** offset of data on first page */
+  unsigned page_offset;
+
+  /** or in first iovec */
+  unsigned iov_offset;
+ };

  /** File used in the request (or NULL) */
  struct fuse_file *ff;
```

Subject: [PATCH 2/4] fuse: re-work fuse_get_user_pages() to operate on iovec[]
Posted by Maxim Patlasov on Fri, 20 Jul 2012 11:50:33 GMT
View Forum Message <> Reply to Message

Now fuse_get_user_pages() takes iovec[] as argument and packs so much data
from it to req->pages[] as possible.

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>

```
---
 fs/fuse/file.c |   64 +++++++++++++++++++++++++++++++++++++++++++++++---------------
 1 files changed, 47 insertions(+), 17 deletions(-)

diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index b321a68..d84416d 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -1020,41 +1020,71 @@ static void fuse_release_user_pages(struct fuse_req *req, int write)
  }
 }

-static int fuse_get_user_pages(struct fuse_req *req, const char __user *buf,
+static int fuse_get_user_pages(struct fuse_req *req,
+         const struct iovec **iov_pp,
+         unsigned long *nr_segs_p,
+         size_t *iov_offset_p,
          size_t *nbytesp, int write)
 {
- size_t nbytes = *nbytesp;
- unsigned long user_addr = (unsigned long) buf;
- unsigned offset = user_addr & ~PAGE_MASK;
- int npages;
+ size_t nbytes = 0;  /* # bytes already packed in req */
```

```
  /* Special case for kernel I/O: can copy directly into the buffer */
  if (segment_eq(get_fs(), KERNEL_DS)) {
+  BUG_ON(*iov_offset_p);
   if (write)
-   req->in.args[1].value = (void *) user_addr;
+   req->in.args[1].value = (*iov_pp)->iov_base;
   else
-   req->out.args[0].value = (void *) user_addr;
+   req->out.args[0].value = (*iov_pp)->iov_base;

+  (*iov_pp)++;
+  (*nr_segs_p)--;
   return 0;
  }

- nbytes = min_t(size_t, nbytes, FUSE_MAX_PAGES_PER_REQ << PAGE_SHIFT);
- npages = (nbytes + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
- npages = clamp(npages, 1, FUSE_MAX_PAGES_PER_REQ);
- npages = get_user_pages_fast(user_addr, npages, !write, req->pages);
- if (npages < 0)
-  return npages;
+ req->iovec = *iov_pp;
+ req->iov_offset = *iov_offset_p;

- req->num_pages = npages;
- req->page_offset = offset;
+ while (nbytes < *nbytesp && req->num_pages < FUSE_MAX_PAGES_PER_REQ) {
+  int npages;
+  unsigned long user_addr = (unsigned long)(*iov_pp)->iov_base +
+       *iov_offset_p;
+  unsigned offset = user_addr & ~PAGE_MASK;
+  size_t frag_size = min_t(size_t,
+      (*iov_pp)->iov_len - *iov_offset_p,
+      *nbytesp - nbytes);
+
+  int n = FUSE_MAX_PAGES_PER_REQ - req->num_pages;
+  frag_size = min_t(size_t, frag_size, n << PAGE_SHIFT);
+
+  npages = (frag_size + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
+  npages = clamp(npages, 1, n);
+
+  npages = get_user_pages_fast(user_addr, npages, !write,
+          &req->pages[req->num_pages]);
+  if (npages < 0)
+   return npages;
+
+  frag_size = min_t(size_t, frag_size,
+      (npages << PAGE_SHIFT) - offset);
```

```
+  nbytes += frag_size;
+
+  if (frag_size < (*iov_pp)->iov_len - *iov_offset_p) {
+   *iov_offset_p += frag_size;
+  } else {
+   (*iov_pp)++;
+   (*nr_segs_p)--;
+   *iov_offset_p = 0;
+  }
+
+  req->num_pages += npages;
+ }

  if (write)
   req->in.argpages = 1;
  else
   req->out.argpages = 1;

- nbytes = (req->num_pages << PAGE_SHIFT) - req->page_offset;
- *nbytesp = min(*nbytesp, nbytes);
+ *nbytesp = nbytes;

  return 0;
 }
```

Subject: [PATCH 3/4] fuse: re-work fuse_direct_io() to operate on iovec[]
Posted by Maxim Patlasov on Fri, 20 Jul 2012 11:50:45 GMT
View Forum Message <> Reply to Message

The patch transparently passes iovec[] argument from fuse_direct_io()
to fuse_get_user_pages() hoping that the latter would fill req with
some part of iovec[].

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
---
 fs/fuse/file.c | 17 ++++++++++++----
 1 files changed, 13 insertions(+), 4 deletions(-)

```
diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index d84416d..6c8e24f 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -1089,8 +1089,9 @@ static int fuse_get_user_pages(struct fuse_req *req,
  return 0;
 }

-ssize_t fuse_direct_io(struct file *file, const char __user *buf,
```

```
-         size_t count, loff_t *ppos, int write)
+static ssize_t __fuse_direct_io(struct file *file, const struct iovec *iov,
+    unsigned long nr_segs, size_t count,
+    loff_t *ppos, int write)
 {
  struct fuse_file *ff = file->private_data;
  struct fuse_conn *fc = ff->fc;
@@ -1098,6 +1099,7 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
  loff_t pos = *ppos;
  ssize_t res = 0;
  struct fuse_req *req;
+ size_t iov_offset = 0;

  req = fuse_get_req(fc);
  if (IS_ERR(req))
@@ -1107,7 +1109,8 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
   size_t nres;
   fl_owner_t owner = current->files;
   size_t nbytes = min(count, nmax);
-  int err = fuse_get_user_pages(req, buf, &nbytes, write);
+  int err = fuse_get_user_pages(req, &iov, &nr_segs, &iov_offset,
+            &nbytes, write);
   if (err) {
    res = err;
    break;
@@ -1130,7 +1133,6 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
   count -= nres;
   res += nres;
   pos += nres;
-  buf += nres;
   if (nres != nbytes)
    break;
   if (count) {
@@ -1147,6 +1149,13 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,

  return res;
 }
+
+ssize_t fuse_direct_io(struct file *file, const char __user *buf,
+         size_t count, loff_t *ppos, int write)
+{
+ struct iovec iov = { .iov_base = (void *)buf, .iov_len = count };
+ return __fuse_direct_io(file, &iov, 1, count, ppos, write);
+}
 EXPORT_SYMBOL_GPL(fuse_direct_io);

 static ssize_t fuse_direct_read(struct file *file, char __user *buf,
```

Subject: [PATCH 4/4] fuse: re-work fuse_direct_IO()
Posted by Maxim Patlasov on Fri, 20 Jul 2012 11:50:55 GMT
View Forum Message <> Reply to Message

The patch eliminates fuse_loop_dio() by passing iovec[] transparently from fuse_direct_IO() to __fuse_direct_io(). The latter is responsible now for processing elements of iovec[]. This allows __fuse_direct_io() to pack many iovec-s to each fuse_req, effectively minimizing number of fuse_req-s required.

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>

```
---
 fs/fuse/file.c |   64 +++++++++++++++++++++-------------------------------------
 1 files changed, 21 insertions(+), 43 deletions(-)

diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index 6c8e24f..0355128 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -1158,8 +1158,8 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
 }
 EXPORT_SYMBOL_GPL(fuse_direct_io);

-static ssize_t fuse_direct_read(struct file *file, char __user *buf,
-          size_t count, loff_t *ppos)
+static ssize_t __fuse_direct_read(struct file *file, const struct iovec *iov,
+      unsigned long nr_segs, loff_t *ppos)
 {
  ssize_t res;
  struct inode *inode = file->f_path.dentry->d_inode;
@@ -1167,22 +1167,31 @@ static ssize_t fuse_direct_read(struct file *file, char __user *buf,
  if (is_bad_inode(inode))
   return -EIO;

- res = fuse_direct_io(file, buf, count, ppos, 0);
+ res = __fuse_direct_io(file, iov, nr_segs, iov_length(iov, nr_segs),
+          ppos, 0);

  fuse_invalidate_attr(inode);

  return res;
 }

-static ssize_t __fuse_direct_write(struct file *file, const char __user *buf,
-       size_t count, loff_t *ppos)
+static ssize_t fuse_direct_read(struct file *file, char __user *buf,
+         size_t count, loff_t *ppos)
+{
+ struct iovec iov = { .iov_base = (void *)buf, .iov_len = count };
```

```
+ return __fuse_direct_read(file, &iov, 1, ppos);
+}
+
+static ssize_t __fuse_direct_write(struct file *file, const struct iovec *iov,
+       unsigned long nr_segs, loff_t *ppos)
 {
  struct inode *inode = file->f_path.dentry->d_inode;
+ size_t count = iov_length(iov, nr_segs);
  ssize_t res;

  res = generic_write_checks(file, ppos, &count, 0);
  if (!res) {
-  res = fuse_direct_io(file, buf, count, ppos, 1);
+  res = __fuse_direct_io(file, iov, nr_segs, count, ppos, 1);
   if (res > 0)
    fuse_write_update_size(inode, *ppos);
  }
@@ -1195,6 +1204,7 @@ static ssize_t __fuse_direct_write(struct file *file, const char __user
*buf,
 static ssize_t fuse_direct_write(struct file *file, const char __user *buf,
      size_t count, loff_t *ppos)
 {
+ struct iovec iov = { .iov_base = (void *)buf, .iov_len = count };
  struct inode *inode = file->f_path.dentry->d_inode;
  ssize_t res;

@@ -1203,7 +1213,7 @@ static ssize_t fuse_direct_write(struct file *file, const char __user *buf,

  /* Don't allow parallel writes to the same file */
  mutex_lock(&inode->i_mutex);
- res = __fuse_direct_write(file, buf, count, ppos);
+ res = __fuse_direct_write(file, &iov, 1, ppos);
  mutex_unlock(&inode->i_mutex);

  return res;
@@ -2161,41 +2171,6 @@ int fuse_notify_poll_wakeup(struct fuse_conn *fc,
  return 0;
 }

-static ssize_t fuse_loop_dio(struct file *filp, const struct iovec *iov,
-       unsigned long nr_segs, loff_t *ppos, int rw)
-{
- const struct iovec *vector = iov;
- ssize_t ret = 0;
-
- while (nr_segs > 0) {
-  void __user *base;
-  size_t len;
```

```
-  ssize_t nr;
-
-  base = vector->iov_base;
-  len = vector->iov_len;
-  vector++;
-  nr_segs--;
-
-  if (rw == WRITE)
-   nr = __fuse_direct_write(filp, base, len, ppos);
-  else
-   nr = fuse_direct_read(filp, base, len, ppos);
-
-  if (nr < 0) {
-   if (!ret)
-    ret = nr;
-   break;
-  }
-  ret += nr;
-  if (nr != len)
-   break;
- }
-
- return ret;
-}
-
-
 static ssize_t
 fuse_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
    loff_t offset, unsigned long nr_segs)
@@ -2207,7 +2182,10 @@ fuse_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
  file = iocb->ki_filp;
  pos = offset;

- ret = fuse_loop_dio(file, iov, nr_segs, &pos, rw);
+ if (rw == WRITE)
+  ret = __fuse_direct_write(file, iov, nr_segs, &pos);
+ else
+  ret = __fuse_direct_read(file, iov, nr_segs, &pos);

  return ret;
 }
```

Subject: Re: [PATCH 1/4] fuse: add basic support of iovec[] to fuse_req
Posted by Miklos Szeredi on Wed, 08 Aug 2012 16:02:17 GMT
View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

> The patch allows fuse_req to refer to array of iovec-s describing
> layout of user-data over req->pages. fuse_copy_pages() is re-worked to
> support both cased: former layout where pages[] corresponded to <buf, len>
> and newer one where pages[] corresponds to iovec[].
>
> Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
> ---
> fs/fuse/dev.c | 52 ++---
> fs/fuse/fuse_i.h | 12 ++++++++++--
> 2 files changed, 59 insertions(+), 5 deletions(-)
>
> diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
> index 7df2b5e..cdae525 100644
> --- a/fs/fuse/dev.c
> +++ b/fs/fuse/dev.c
> @@ -850,9 +850,9 @@ static int fuse_copy_page(struct fuse_copy_state *cs, struct page
**pagep,
>   return 0;
> }
>
> -/* Copy pages in the request to/from userspace buffer */
> -static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned nbytes,
> -   int zeroing)
> +/* Start from addr(pages[0]) + page_offset. No holes in the middle. */
> +static int fuse_copy_pages_for_buf(struct fuse_copy_state *cs, unsigned nbytes,
> +   int zeroing)
> {
>   unsigned i;
>   struct fuse_req *req = cs->req;
> @@ -874,6 +874,52 @@ static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned
nbytes,
>   return 0;
> }
>
> +/* Take iov_offset as offset in iovec[0]. Iterate based on iovec[].iov_len */
> +static int fuse_copy_pages_for_iovec(struct fuse_copy_state *cs,
> +   unsigned nbytes, int zeroing)
> +{
> + unsigned i;
> + struct fuse_req *req = cs->req;
> + const struct iovec *iov = req->iovec;
> + unsigned iov_offset = req->iov_offset;
> +
> + for (i = 0; i < req->num_pages && (nbytes || zeroing); i++) {
> + int err;
> + unsigned long user_addr = (unsigned long)iov->iov_base +
> +   iov_offset;

> + unsigned offset = user_addr & ~PAGE_MASK;
> + unsigned count = min_t(size_t, PAGE_SIZE - offset,
> +   iov->iov_len - iov_offset);

It would be much cleaner if we didn't have to deal with the original iovec here, but only offset and length relative to the page.

I understand that that would mean allocating an array for these. In the other thread I mentioned the possibility of allocating the page array. Instead of a page array, we could have an array of (pageptr, offset, len) which would simplify the whole thing.

Thanks,
Miklos
