Subject: [PATCH 0/6] fuse: allocate req->pages[] dynamically
Posted by Maxim Patlasov on Fri, 07 Sep 2012 17:40:37 GMT
View Forum Message <> Reply to Message

Hi,

Currently, any fuse request always includes inline pages[] array of
FUSE_MAX_PAGES_PER_REQ elements. This is the waste of memory because
in many cases smaller size would suffice.

The patch-set tries to allocate only as many elements of pages[] array as
actaully needed. This will be even more useful in the future because of:

1. Mitsuo's patches making maximum read/write request size tunable.
2. My patches optimizing scatter-gather direct IO. To make them simplier I'll
need to substitute array of 'struct page *' with array of 'struct bio_vec'.
It would make memory overhead worse if implemented w/o this patch-set.

Thanks,
Maxim

Maxim Patlasov (6):
    fuse: general infrastructure for pages[] of variable size
    fuse: categorize fuse_get_req()
    fuse: rework fuse_retrieve()
    fuse: rework fuse_readpages()
    fuse: rework fuse_perform_write()
    fuse: rework fuse_do_ioctl()

 fs/fuse/cuse.c | 2 +-
 fs/fuse/dev.c | 70 ++++++

The patch removes inline array of FUSE_MAX_PAGES_PER_REQ page pointers from
fuse_req. Instead of that, req->pages may now point either to small inline
array or to an array allocated dynamically.

This essentially means that all callers of fuse_request_alloc[_nofs] should
pass the number of pages needed explicitly.

The patch doesn't make any logic changes.
---
 fs/fuse/dev.c | 40 ++++++++++++++++++++++++++++++++++++++----------
 fs/fuse/file.c | 4 ++--
 fs/fuse/fuse_i.h | 9 ++++++---
 fs/fuse/inode.c | 4 ++--
 4 files changed, 40 insertions(+), 17 deletions(-)

diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
index 7df2b5e..c0283a1 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
@@ -36,32 +36,52 @@ static struct fuse_conn *fuse_get_conn(struct file *file)

 static void fuse_request_init(struct fuse_req *req)
 {
+ struct page **pages = req->pages;
+
  memset(req, 0, sizeof(*req));
  INIT_LIST_HEAD(&req->list);
  INIT_LIST_HEAD(&req->intr_entry);
  init_waitqueue_head(&req->waitq);
  atomic_set(&req->count, 1);
+
+ req->pages = pages;
 }

-struct fuse_req *fuse_request_alloc(void)
+static struct fuse_req *__fuse_request_alloc(int npages, gfp_t flags)
 {
- struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, GFP_KERNEL);
- if (req)
+ struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, flags);
+ if (req) {
+ if (npages <= 1)
+   req->pages = req->inline_pages;
+ else
+   req->pages = kmalloc(sizeof(struct page *) * npages,
+   flags);
+
+ if (!req->pages) {

```
+   kmem_cache_free(fuse_req_cachep, req);
+   return NULL;
+  }
+
   fuse_request_init(req);
+ }
  return req;
 }
+
+struct fuse_req *fuse_request_alloc(int npages)
+{
+ return __fuse_request_alloc(npages, GFP_KERNEL);
+}
 EXPORT_SYMBOL_GPL(fuse_request_alloc);

-struct fuse_req *fuse_request_alloc_nofs(void)
+struct fuse_req *fuse_request_alloc_nofs(int npages)
 {
- struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, GFP_NOFS);
- if (req)
-  fuse_request_init(req);
- return req;
+ return __fuse_request_alloc(npages, GFP_NOFS);
 }

 void fuse_request_free(struct fuse_req *req)
 {
+ if (req->pages != req->inline_pages)
+  kfree(req->pages);
  kmem_cache_free(fuse_req_cachep, req);
 }

@@ -116,7 +136,7 @@ struct fuse_req *fuse_get_req(struct fuse_conn *fc)
  if (!fc->connected)
   goto out;

- req = fuse_request_alloc();
+ req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
  err = -ENOMEM;
  if (!req)
   goto out;
@@ -193,7 +213,7 @@ struct fuse_req *fuse_get_req_nofail(struct fuse_conn *fc, struct file *file)

  atomic_inc(&fc->num_waiting);
  wait_event(fc->blocked_waitq, !fc->blocked);
- req = fuse_request_alloc();
+ req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
  if (!req)
```

```
   req = get_reserved_req(fc, file);

diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index aba15f1..7423ea4 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -57,7 +57,7 @@ struct fuse_file *fuse_file_alloc(struct fuse_conn *fc)
   return NULL;

  ff->fc = fc;
- ff->reserved_req = fuse_request_alloc();
+ ff->reserved_req = fuse_request_alloc(0);
  if (unlikely(!ff->reserved_req)) {
   kfree(ff);
   return NULL;
@@ -1272,7 +1272,7 @@ static int fuse_writepage_locked(struct page *page)

  set_page_writeback(page);

- req = fuse_request_alloc_nofs();
+ req = fuse_request_alloc_nofs(1);
  if (!req)
   goto err;

diff --git a/fs/fuse/fuse_i.h b/fs/fuse/fuse_i.h
index e24dd74..5e78840 100644
--- a/fs/fuse/fuse_i.h
+++ b/fs/fuse/fuse_i.h
@@ -291,7 +291,10 @@ struct fuse_req {
  } misc;

  /** page vector */
- struct page *pages[FUSE_MAX_PAGES_PER_REQ];
+ struct page **pages;
+
+ /** inline page vector */
+ struct page *inline_pages[1];

  /** number of pages in vector */
  unsigned num_pages;
@@ -658,9 +661,9 @@ void fuse_ctl_cleanup(void);
 /**
  * Allocate a request
  */
-struct fuse_req *fuse_request_alloc(void);
+struct fuse_req *fuse_request_alloc(int npages);

-struct fuse_req *fuse_request_alloc_nofs(void);
```

```
+struct fuse_req *fuse_request_alloc_nofs(int npages);

 /**
  * Free a request
diff --git a/fs/fuse/inode.c b/fs/fuse/inode.c
index ce0a283..3f399ba 100644
--- a/fs/fuse/inode.c
+++ b/fs/fuse/inode.c
@@ -1027,12 +1027,12 @@ static int fuse_fill_super(struct super_block *sb, void *data, int silent)
  /* only now - we want root dentry with NULL ->d_op */
  sb->s_d_op = &fuse_dentry_operations;

- init_req = fuse_request_alloc();
+ init_req = fuse_request_alloc(0);
  if (!init_req)
   goto err_put_root;

  if (is_bdev) {
-  fc->destroy_req = fuse_request_alloc();
+  fc->destroy_req = fuse_request_alloc(0);
   if (!fc->destroy_req)
    goto err_free_init_req;
  }
```

Subject: [PATCH 2/6] fuse: categorize fuse_get_req()
Posted by Maxim Patlasov on Fri, 07 Sep 2012 17:41:19 GMT
View Forum Message <> Reply to Message

The patch categorizes all fuse_get_req() invocations into three categories:
 - fuse_get_req_nopages() - when caller doesn't care about req->pages
 - fuse_get_req_onepage() - when caller need exactly one page
 - fuse_get_req_multipage() - in other cases.

The decision to split one fuse_get_req() into three different functions was
made because having fuse_get_req(fc, n) scattered over code would be too error
prone. After splitting, it's clear from the first glance when a caller needs
multi-page fuse_req.

The patch doesn't make any logic changes. In multi-page case, it silly
allocates array of FUSE_MAX_PAGES_PER_REQ page pointers. This will be amended
by future patches.

```
 fs/fuse/cuse.c   |    2 +-
 fs/fuse/dev.c    |   10 +++++-----
 fs/fuse/dir.c    |   38 ++++++++++++++++++++++----------------
 fs/fuse/file.c   |   30 ++++++++++++++++++++----------
 fs/fuse/fuse_i.h |   36 ++++++++++++++++++++++++++++++++---
```

```
 fs/fuse/inode.c  |    2 +-
 6 files changed, 74 insertions(+), 44 deletions(-)

diff --git a/fs/fuse/cuse.c b/fs/fuse/cuse.c
index 3426521..672d86a 100644
--- a/fs/fuse/cuse.c
+++ b/fs/fuse/cuse.c
@@ -411,7 +411,7 @@ static int cuse_send_init(struct cuse_conn *cc)

  BUILD_BUG_ON(CUSE_INIT_INFO_MAX > PAGE_SIZE);

- req = fuse_get_req(fc);
+ req = fuse_get_req_onepage(fc);
  if (IS_ERR(req)) {
   rc = PTR_ERR(req);
   goto err;
diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
index c0283a1..6e38af5 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
@@ -117,7 +117,7 @@ static void fuse_req_init_context(struct fuse_req *req)
  req->in.h.pid = current->pid;
 }

-struct fuse_req *fuse_get_req(struct fuse_conn *fc)
+struct fuse_req *fuse_get_req(struct fuse_conn *fc, int npages)
 {
  struct fuse_req *req;
  sigset_t oldset;
@@ -136,7 +136,7 @@ struct fuse_req *fuse_get_req(struct fuse_conn *fc)
  if (!fc->connected)
   goto out;

- req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
+ req = fuse_request_alloc(npages);
  err = -ENOMEM;
  if (!req)
   goto out;
@@ -207,13 +207,13 @@ static void put_reserved_req(struct fuse_conn *fc, struct fuse_req
*req)
  * filesystem should not have it's own file open.  If deadlock is
  * intentional, it can still be broken by "aborting" the filesystem.
  */
-struct fuse_req *fuse_get_req_nofail(struct fuse_conn *fc, struct file *file)
+struct fuse_req *fuse_get_req_nofail_nopages(struct fuse_conn *fc, struct file *file)
 {
  struct fuse_req *req;
```

```
  atomic_inc(&fc->num_waiting);
  wait_event(fc->blocked_waitq, !fc->blocked);
- req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
+ req = fuse_request_alloc(0);
  if (!req)
   req = get_reserved_req(fc, file);

@@ -1563,7 +1563,7 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
  unsigned int offset;
  size_t total_len = 0;

- req = fuse_get_req(fc);
+ req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
  if (IS_ERR(req))
   return PTR_ERR(req);

diff --git a/fs/fuse/dir.c b/fs/fuse/dir.c
index 324bc08..ff7af98 100644
--- a/fs/fuse/dir.c
+++ b/fs/fuse/dir.c
@@ -178,7 +178,7 @@ static int fuse_dentry_revalidate(struct dentry *entry, unsigned int flags)
    return -ECHILD;

   fc = get_fuse_conn(inode);
-  req = fuse_get_req(fc);
+  req = fuse_get_req_nopages(fc);
   if (IS_ERR(req))
    return 0;

@@ -271,7 +271,7 @@ int fuse_lookup_name(struct super_block *sb, u64 nodeid, struct qstr
*name,
  if (name->len > FUSE_NAME_MAX)
   goto out;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  err = PTR_ERR(req);
  if (IS_ERR(req))
   goto out;
@@ -391,7 +391,7 @@ static int fuse_create_open(struct inode *dir, struct dentry *entry,
  if (!forget)
   goto out_err;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  err = PTR_ERR(req);
  if (IS_ERR(req))
   goto out_put_forget_req;
```

```
@@ -592,7 +592,7 @@ static int fuse_mknod(struct inode *dir, struct dentry *entry, umode_t
mode,
 {
  struct fuse_mknod_in inarg;
  struct fuse_conn *fc = get_fuse_conn(dir);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -623,7 +623,7 @@ static int fuse_mkdir(struct inode *dir, struct dentry *entry, umode_t
mode)
 {
  struct fuse_mkdir_in inarg;
  struct fuse_conn *fc = get_fuse_conn(dir);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -647,7 +647,7 @@ static int fuse_symlink(struct inode *dir, struct dentry *entry,
 {
  struct fuse_conn *fc = get_fuse_conn(dir);
  unsigned len = strlen(link) + 1;
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -664,7 +664,7 @@ static int fuse_unlink(struct inode *dir, struct dentry *entry)
 {
  int err;
  struct fuse_conn *fc = get_fuse_conn(dir);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -696,7 +696,7 @@ static int fuse_rmdir(struct inode *dir, struct dentry *entry)
 {
  int err;
  struct fuse_conn *fc = get_fuse_conn(dir);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -723,7 +723,7 @@ static int fuse_rename(struct inode *olddir, struct dentry *oldent,
```

```
  int err;
  struct fuse_rename_in inarg;
  struct fuse_conn *fc = get_fuse_conn(olddir);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);

  if (IS_ERR(req))
   return PTR_ERR(req);
@@ -776,7 +776,7 @@ static int fuse_link(struct dentry *entry, struct inode *newdir,
  struct fuse_link_in inarg;
  struct inode *inode = entry->d_inode;
  struct fuse_conn *fc = get_fuse_conn(inode);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -848,7 +848,7 @@ static int fuse_do_getattr(struct inode *inode, struct kstat *stat,
  struct fuse_req *req;
  u64 attr_version;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1029,7 +1029,7 @@ static int fuse_access(struct inode *inode, int mask)
  if (fc->no_access)
   return 0;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1167,7 +1167,7 @@ static int fuse_readdir(struct file *file, void *dstbuf, filldir_t filldir)
  if (is_bad_inode(inode))
   return -EIO;

- req = fuse_get_req(fc);
+ req = fuse_get_req_onepage(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1197,7 +1197,7 @@ static char *read_link(struct dentry *dentry)
 {
  struct inode *inode = dentry->d_inode;
  struct fuse_conn *fc = get_fuse_conn(inode);
```

```
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  char *link;

  if (IS_ERR(req))
@@ -1410,7 +1410,7 @@ static int fuse_do_setattr(struct dentry *entry, struct iattr *attr,
  if (attr->ia_valid & ATTR_SIZE)
   is_truncate = true;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1518,7 +1518,7 @@ static int fuse_setxattr(struct dentry *entry, const char *name,
  if (fc->no_setxattr)
   return -EOPNOTSUPP;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1557,7 +1557,7 @@ static ssize_t fuse_getxattr(struct dentry *entry, const char *name,
  if (fc->no_getxattr)
   return -EOPNOTSUPP;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1609,7 +1609,7 @@ static ssize_t fuse_listxattr(struct dentry *entry, char *list, size_t size)
  if (fc->no_listxattr)
   return -EOPNOTSUPP;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1654,7 +1654,7 @@ static int fuse_removexattr(struct dentry *entry, const char *name)
  if (fc->no_removexattr)
   return -EOPNOTSUPP;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
```

```
   return PTR_ERR(req);

diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index 7423ea4..c811f3d 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -25,7 +25,7 @@ static int fuse_send_open(struct fuse_conn *fc, u64 nodeid, struct file *file,
  struct fuse_req *req;
  int err;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -368,7 +368,7 @@ static int fuse_flush(struct file *file, fl_owner_t id)
  if (fc->no_flush)
   return 0;

- req = fuse_get_req_nofail(fc, file);
+ req = fuse_get_req_nofail_nopages(fc, file);
  memset(&inarg, 0, sizeof(inarg));
  inarg.fh = ff->fh;
  inarg.lock_owner = fuse_lock_owner_id(fc, id);
@@ -436,7 +436,7 @@ int fuse_fsync_common(struct file *file, loff_t start, loff_t end,

  fuse_sync_writes(inode);

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req)) {
   err = PTR_ERR(req);
   goto out;
@@ -544,7 +544,7 @@ static int fuse_readpage(struct file *file, struct page *page)
   */
  fuse_wait_on_page_writeback(inode, page->index);

- req = fuse_get_req(fc);
+ req = fuse_get_req_onepage(fc);
  err = PTR_ERR(req);
  if (IS_ERR(req))
   goto out;
@@ -657,7 +657,7 @@ static int fuse_readpages_fill(void *_data, struct page *page)
       (req->num_pages + 1) * PAGE_CACHE_SIZE > fc->max_read ||
       req->pages[req->num_pages - 1]->index + 1 != page->index)) {
   fuse_send_readpages(req, data->file);
-  data->req = req = fuse_get_req(fc);
+  data->req = req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
```

```
   if (IS_ERR(req)) {
    unlock_page(page);
    return PTR_ERR(req);
@@ -683,7 +683,7 @@ static int fuse_readpages(struct file *file, struct address_space *mapping,

  data.file = file;
  data.inode = inode;
- data.req = fuse_get_req(fc);
+ data.req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
  err = PTR_ERR(data.req);
  if (IS_ERR(data.req))
   goto out;
@@ -890,7 +890,7 @@ static ssize_t fuse_perform_write(struct file *file,
   struct fuse_req *req;
   ssize_t count;

-  req = fuse_get_req(fc);
+  req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
   if (IS_ERR(req)) {
    err = PTR_ERR(req);
    break;
@@ -1072,7 +1072,7 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
  ssize_t res = 0;
  struct fuse_req *req;

- req = fuse_get_req(fc);
+ req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1108,7 +1108,7 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
    break;
   if (count) {
    fuse_put_request(fc, req);
-   req = fuse_get_req(fc);
+   req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
    if (IS_ERR(req))
     break;
   }
@@ -1470,7 +1470,7 @@ static int fuse_getlk(struct file *file, struct file_lock *fl)
  struct fuse_lk_out outarg;
  int err;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);
```

```
@@ -1505,7 +1505,7 @@ static int fuse_setlk(struct file *file, struct file_lock *fl, int flock)
  if (fl->fl_flags & FL_CLOSE)
   return 0;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1574,7 +1574,7 @@ static sector_t fuse_bmap(struct address_space *mapping, sector_t
block)
  if (!inode->i_sb->s_bdev || fc->no_bmap)
   return 0;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return 0;

@@ -1872,7 +1872,7 @@ long fuse_do_ioctl(struct file *file, unsigned int cmd, unsigned long
arg,
   num_pages++;
  }

- req = fuse_get_req(fc);
+ req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
  if (IS_ERR(req)) {
   err = PTR_ERR(req);
   req = NULL;
@@ -2075,7 +2075,7 @@ unsigned fuse_file_poll(struct file *file, poll_table *wait)
   fuse_register_polled_file(fc, ff);
  }

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return POLLERR;

@@ -2193,7 +2193,7 @@ long fuse_file_fallocate(struct file *file, int mode, loff_t offset,
  if (fc->no_fallocate)
   return -EOPNOTSUPP;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

diff --git a/fs/fuse/fuse_i.h b/fs/fuse/fuse_i.h
```

```
index 5e78840..f41b149 100644
--- a/fs/fuse/fuse_i.h
+++ b/fs/fuse/fuse_i.h
@@ -671,14 +671,44 @@ struct fuse_req *fuse_request_alloc_nofs(int npages);
 void fuse_request_free(struct fuse_req *req);

 /**
- * Get a request, may fail with -ENOMEM
+ * Get a request, may fail with -ENOMEM,
+ * caller should specify # elements in req->pages[] explicitly
  */
-struct fuse_req *fuse_get_req(struct fuse_conn *fc);
+struct fuse_req *fuse_get_req(struct fuse_conn *fc, int npages);
+
+/**
+ * Get a request, may fail with -ENOMEM,
+ * useful for callers who doesn't use req->pages[]
+ */
+static inline struct fuse_req *fuse_get_req_nopages(struct fuse_conn *fc)
+{
+ return fuse_get_req(fc, 0);
+}
+
+/**
+ * Get a request, may fail with -ENOMEM,
+ * useful for callers who need only one page in req->pages[]
+ */
+static inline struct fuse_req *fuse_get_req_onepage(struct fuse_conn *fc)
+{
+ return fuse_get_req(fc, 1);
+}
+
+/**
+ * Get a request, may fail with -ENOMEM,
+ * useful for callers who need many pages in req->pages[]
+ */
+static inline struct fuse_req *fuse_get_req_multipage(struct fuse_conn *fc,
+           int n)
+{
+ return fuse_get_req(fc, n);
+}

 /**
  * Gets a requests for a file operation, always succeeds
  */
-struct fuse_req *fuse_get_req_nofail(struct fuse_conn *fc, struct file *file);
+struct fuse_req *fuse_get_req_nofail_nopages(struct fuse_conn *fc,
+         struct file *file);
```

```
 /**
  * Decrement reference count of a request.  If count goes to zero free
diff --git a/fs/fuse/inode.c b/fs/fuse/inode.c
index 3f399ba..efb6144 100644
--- a/fs/fuse/inode.c
+++ b/fs/fuse/inode.c
@@ -411,7 +411,7 @@ static int fuse_statfs(struct dentry *dentry, struct kstatfs *buf)
   return 0;
  }

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);
```

Subject: [PATCH 3/6] fuse: rework fuse_retrieve()

Posted by Maxim Patlasov on Fri, 07 Sep 2012 17:41:32 GMT

View Forum Message <> Reply to Message

The patch reworks fuse_retrieve() to allocate only so many page pointers as needed. The core part of the patch is the following calculation:

```
 num_pages = (num + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
```

(thanks Miklos for formula). All other changes are moslty shuffling lines.

The patch also contains one-line fix suggested by Miklos: the loop iterating over pages from inode mapping should interpret 'offset' as offset in the very first page only.

```
 fs/fuse/dev.c |   26 ++++++++++++++++++----------
 1 files changed, 16 insertions(+), 10 deletions(-)

diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
index 6e38af5..cf69df3 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
@@ -1562,13 +1562,24 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
  unsigned int num;
  unsigned int offset;
  size_t total_len = 0;
+ int num_pages;

- req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
+ offset = outarg->offset & ~PAGE_CACHE_MASK;
+ file_size = i_size_read(inode);
```

```
+
+ num = outarg->size;
+ if (outarg->offset > file_size)
+  num = 0;
+ else if (outarg->offset + num > file_size)
+  num = file_size - outarg->offset;
+
+ num_pages = (num + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
+ num_pages = min(num_pages, FUSE_MAX_PAGES_PER_REQ);
+
+ req = fuse_get_req_multipage(fc, num_pages);
  if (IS_ERR(req))
   return PTR_ERR(req);

- offset = outarg->offset & ~PAGE_CACHE_MASK;
-
  req->in.h.opcode = FUSE_NOTIFY_REPLY;
  req->in.h.nodeid = outarg->nodeid;
  req->in.numargs = 2;
@@ -1577,14 +1588,8 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
  req->end = fuse_retrieve_end;

  index = outarg->offset >> PAGE_CACHE_SHIFT;
- file_size = i_size_read(inode);
- num = outarg->size;
- if (outarg->offset > file_size)
-  num = 0;
- else if (outarg->offset + num > file_size)
-  num = file_size - outarg->offset;

- while (num && req->num_pages < FUSE_MAX_PAGES_PER_REQ) {
+ while (num && req->num_pages < num_pages) {
   struct page *page;
   unsigned int this_num;

@@ -1596,6 +1601,7 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
   req->pages[req->num_pages] = page;
   req->num_pages++;

+  offset = 0;
   num -= this_num;
   total_len += this_num;
   index++;
```

Subject: [PATCH 4/6] fuse: rework fuse_readpages()
Posted by Maxim Patlasov on Fri, 07 Sep 2012 17:41:40 GMT

The patch uses 'nr_pages' argument of fuse_readpages() as the heuristics for
number of page pointers to allocate.

This can be improved further by taking in consideration fc->max_read and gaps
between page indices, but it's not clear whetheher it's worthy or not.
---
 fs/fuse/file.c | 10 ++++++++--
 1 files changed, 8 insertions(+), 2 deletions(-)

```
diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index c811f3d..9a6dcc6 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -641,6 +641,7 @@ struct fuse_fill_data {
  struct fuse_req *req;
  struct file *file;
  struct inode *inode;
+ unsigned nr_pages;
 };

 static int fuse_readpages_fill(void *_data, struct page *page)
@@ -656,8 +657,10 @@ static int fuse_readpages_fill(void *_data, struct page *page)
       (req->num_pages == FUSE_MAX_PAGES_PER_REQ ||
        (req->num_pages + 1) * PAGE_CACHE_SIZE > fc->max_read ||
        req->pages[req->num_pages - 1]->index + 1 != page->index)) {
+  int nr_alloc = min_t(unsigned, data->nr_pages,
+         FUSE_MAX_PAGES_PER_REQ);
   fuse_send_readpages(req, data->file);
-  data->req = req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
+  data->req = req = fuse_get_req_multipage(fc, nr_alloc);
   if (IS_ERR(req)) {
    unlock_page(page);
    return PTR_ERR(req);
@@ -666,6 +669,7 @@ static int fuse_readpages_fill(void *_data, struct page *page)
  page_cache_get(page);
  req->pages[req->num_pages] = page;
  req->num_pages++;
+ data->nr_pages--;
  return 0;
 }

@@ -676,6 +680,7 @@ static int fuse_readpages(struct file *file, struct address_space *mapping,
  struct fuse_conn *fc = get_fuse_conn(inode);
  struct fuse_fill_data data;
  int err;
+ int nr_alloc = min_t(unsigned, nr_pages, FUSE_MAX_PAGES_PER_REQ);
```

```
  err = -EIO;
  if (is_bad_inode(inode))
@@ -683,7 +688,8 @@ static int fuse_readpages(struct file *file, struct address_space *mapping,

  data.file = file;
  data.inode = inode;
- data.req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
+ data.req = fuse_get_req_multipage(fc, nr_alloc);
+ data.nr_pages = nr_pages;
  err = PTR_ERR(data.req);
  if (IS_ERR(data.req))
   goto out;
```

Subject: [PATCH 5/6] fuse: rework fuse_perform_write()
Posted by Maxim Patlasov on Fri, 07 Sep 2012 17:41:49 GMT
View Forum Message <> Reply to Message

The patch allocates as many page pointers in fuse_req as needed to cover
interval [pos .. pos+len-1]. FUSE_WR_PAGES macro is introduced to hide this
cumbersome arithmetics.
---
 fs/fuse/file.c | 15 ++++++++++----
 1 files changed, 11 insertions(+), 4 deletions(-)

```
diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index 9a6dcc6..84cc83c 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -815,7 +815,8 @@ static size_t fuse_send_write_pages(struct fuse_req *req, struct file *file,

 static ssize_t fuse_fill_write_pages(struct fuse_req *req,
          struct address_space *mapping,
-         struct iov_iter *ii, loff_t pos)
+         struct iov_iter *ii, loff_t pos,
+         unsigned nr_pages)
 {
  struct fuse_conn *fc = get_fuse_conn(mapping->host);
  unsigned offset = pos & (PAGE_CACHE_SIZE - 1);
@@ -875,11 +876,16 @@ static ssize_t fuse_fill_write_pages(struct fuse_req *req,
   if (!fc->big_writes)
    break;
  } while (iov_iter_count(ii) && count < fc->max_write &&
-   req->num_pages < FUSE_MAX_PAGES_PER_REQ && offset == 0);
+   req->num_pages < nr_pages && offset == 0);

  return count > 0 ? count : err;
 }
```

```
+#define FUSE_WR_PAGES(pos, len) min_t(unsigned,          \
+         ((pos + len - 1) >> PAGE_CACHE_SHIFT) - \
+         (pos >> PAGE_CACHE_SHIFT) + 1,       \
+         FUSE_MAX_PAGES_PER_REQ)
+
 static ssize_t fuse_perform_write(struct file *file,
       struct address_space *mapping,
       struct iov_iter *ii, loff_t pos)
@@ -895,14 +901,15 @@ static ssize_t fuse_perform_write(struct file *file,
  do {
   struct fuse_req *req;
   ssize_t count;
+  unsigned nr_pages = FUSE_WR_PAGES(pos, iov_iter_count(ii));

-  req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
+  req = fuse_get_req_multipage(fc, nr_pages);
   if (IS_ERR(req)) {
    err = PTR_ERR(req);
    break;
   }

-  count = fuse_fill_write_pages(req, mapping, ii, pos);
+  count = fuse_fill_write_pages(req, mapping, ii, pos, nr_pages);
   if (count <= 0) {
    err = count;
   } else {
```

Subject: [PATCH 6/6] fuse: rework fuse_do_ioctl()
Posted by Maxim Patlasov on Fri, 07 Sep 2012 17:41:59 GMT
View Forum Message <> Reply to Message

fuse_do_ioctl() already calculates the number of pages it's going to use. It is
stored in 'num_pages' variable. So the patch simply uses it for allocating
fuse_req.
---
 fs/fuse/file.c | 2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

```
diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index 84cc83c..abb070b 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -1885,7 +1885,7 @@ long fuse_do_ioctl(struct file *file, unsigned int cmd, unsigned long
arg,
   num_pages++;
  }
```

```
- req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
+ req = fuse_get_req_multipage(fc, num_pages);
  if (IS_ERR(req)) {
   err = PTR_ERR(req);
   req = NULL;
```

Subject: Re: [PATCH 0/6] fuse: allocate req->pages[] dynamically
Posted by Maxim Patlasov on Mon, 10 Sep 2012 10:18:41 GMT
View Forum Message <> Reply to Message

Hi,

I noticed that 'Signed-off-by' is missed in my patches. Very sorry for
inconvenience. I'll be more careful next time.

Thanks,
Maxim

> Hi,
>
> Currently, any fuse request always includes inline pages[] array of
> FUSE_MAX_PAGES_PER_REQ elements. This is the waste of memory because
> in many cases smaller size would suffice.
> ...

Subject: Re: [PATCH 0/6] fuse: allocate req->pages[] dynamically
Posted by Maxim Patlasov on Wed, 12 Sep 2012 16:07:18 GMT
View Forum Message <> Reply to Message

Hi Miklos,

So far as no objections appeared, I'll go ahead and replace fuse
req->page with req->pagevec. It will point to an array of structs:

```
struct page_vec {
    struct page    *pv_page;
    unsigned int    pv_len;
    unsigned int    pv_offset;
};
```

instead of 'struct page *' as it used to be. It seems to be what you
suggested in one of your comments. Are you OK about it?

Thanks,
Maxim

> Hi,
>
> Currently, any fuse request always includes inline pages[] array of
> FUSE_MAX_PAGES_PER_REQ elements. This is the waste of memory because
> in many cases smaller size would suffice.
>
> The patch-set tries to allocate only as many elements of pages[] array as
> actaully needed. This will be even more useful in the future because of:
>
> 1. Mitsuo's patches making maximum read/write request size tunable.
> 2. My patches optimizing scatter-gather direct IO. To make them simplier I'll
> need to substitute array of 'struct page *' with array of 'struct bio_vec'.
> It would make memory overhead worse if implemented w/o this patch-set.
>
> Thanks,
> Maxim
>

Subject: Re: [PATCH 1/6] fuse: general infrastructure for pages[] of variable size
Posted by Miklos Szeredi on Wed, 12 Sep 2012 16:09:01 GMT
View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

> The patch removes inline array of FUSE_MAX_PAGES_PER_REQ page pointers from
> fuse_req. Instead of that, req->pages may now point either to small inline
> array or to an array allocated dynamically.
>
> This essentially means that all callers of fuse_request_alloc[_nofs] should
> pass the number of pages needed explicitly.
>
> The patch doesn't make any logic changes.

See comments inline.

> ---
> fs/fuse/dev.c | 40 ++----------
> fs/fuse/file.c | 4 ++--
> fs/fuse/fuse_i.h | 9 ++++++---
> fs/fuse/inode.c | 4 ++--
> 4 files changed, 40 insertions(+), 17 deletions(-)
>

```
> diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
> index 7df2b5e..c0283a1 100644
> --- a/fs/fuse/dev.c
> +++ b/fs/fuse/dev.c
> @@ -36,32 +36,52 @@ static struct fuse_conn *fuse_get_conn(struct file *file)
>
>  static void fuse_request_init(struct fuse_req *req)
>  {
> + struct page **pages = req->pages;
> +
```

Rather than this, make fuse_request_init() take 'pages' as a second
argument.

```
>   memset(req, 0, sizeof(*req));
>   INIT_LIST_HEAD(&req->list);
>   INIT_LIST_HEAD(&req->intr_entry);
>   init_waitqueue_head(&req->waitq);
>   atomic_set(&req->count, 1);
> +
> + req->pages = pages;
>  }
>
> -struct fuse_req *fuse_request_alloc(void)
> +static struct fuse_req *__fuse_request_alloc(int npages, gfp_t flags)
>  {
> - struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, GFP_KERNEL);
> - if (req)
> + struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, flags);
> + if (req) {
> +  if (npages <= 1)
```

Instead of '1' use a constant (e.g. FUSE_REQ_INLINE_PAGES)

```
> +   req->pages = req->inline_pages;
> +  else
> +   req->pages = kmalloc(sizeof(struct page *) * npages,
> +          flags);
> +
> +  if (!req->pages) {
> +   kmem_cache_free(fuse_req_cachep, req);
> +   return NULL;
> +  }
> +
>    fuse_request_init(req);
> + }
>   return req;
>  }
```

> +
> +struct fuse_req *fuse_request_alloc(int npages)
> +{
> + return __fuse_request_alloc(npages, GFP_KERNEL);
> +}
> EXPORT_SYMBOL_GPL(fuse_request_alloc);
>
> -struct fuse_req *fuse_request_alloc_nofs(void)
> +struct fuse_req *fuse_request_alloc_nofs(int npages)
> {
> - struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, GFP_NOFS);
> - if (req)
> - fuse_request_init(req);
> - return req;
> + return __fuse_request_alloc(npages, GFP_NOFS);
> }
>
> void fuse_request_free(struct fuse_req *req)
> {
> + if (req->pages != req->inline_pages)
> + kfree(req->pages);
>   kmem_cache_free(fuse_req_cachep, req);
> }
>
> @@ -116,7 +136,7 @@ struct fuse_req *fuse_get_req(struct fuse_conn *fc)
>   if (!fc->connected)
>   goto out;
>
> - req = fuse_request_alloc();
> + req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
>   err = -ENOMEM;
>   if (!req)
>   goto out;
> @@ -193,7 +213,7 @@ struct fuse_req *fuse_get_req_nofail(struct fuse_conn *fc, struct file
*file)
>
>   atomic_inc(&fc->num_waiting);
>   wait_event(fc->blocked_waitq, !fc->blocked);
> - req = fuse_request_alloc();
> + req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
>   if (!req)
>   req = get_reserved_req(fc, file);
>
> diff --git a/fs/fuse/file.c b/fs/fuse/file.c
> index aba15f1..7423ea4 100644
> --- a/fs/fuse/file.c
> +++ b/fs/fuse/file.c
> @@ -57,7 +57,7 @@ struct fuse_file *fuse_file_alloc(struct fuse_conn *fc)

>   return NULL;
>
>   ff->fc = fc;
> - ff->reserved_req = fuse_request_alloc();
> + ff->reserved_req = fuse_request_alloc(0);
>   if (unlikely(!ff->reserved_req)) {
>   kfree(ff);
>   return NULL;
> @@ -1272,7 +1272,7 @@ static int fuse_writepage_locked(struct page *page)
>
>   set_page_writeback(page);
>
> - req = fuse_request_alloc_nofs();
> + req = fuse_request_alloc_nofs(1);
>   if (!req)
>   goto err;
>
> diff --git a/fs/fuse/fuse_i.h b/fs/fuse/fuse_i.h
> index e24dd74..5e78840 100644
> --- a/fs/fuse/fuse_i.h
> +++ b/fs/fuse/fuse_i.h
> @@ -291,7 +291,10 @@ struct fuse_req {
>   } misc;
>
>   /** page vector */
> - struct page *pages[FUSE_MAX_PAGES_PER_REQ];
> + struct page **pages;
> +
> + /** inline page vector */
> + struct page *inline_pages[1];
>

To defend against programming errors, I think it would be wise to add

 /** size of the 'pages' array */
 unsigned max_pages;

so that when storing pages in the array we can check whether we are
overflowing the array.

>   /** number of pages in vector */
>   unsigned num_pages;
> @@ -658,9 +661,9 @@ void fuse_ctl_cleanup(void);
> /**
>   * Allocate a request
>   */
> -struct fuse_req *fuse_request_alloc(void);
> +struct fuse_req *fuse_request_alloc(int npages);

>
> -struct fuse_req *fuse_request_alloc_nofs(void);
> +struct fuse_req *fuse_request_alloc_nofs(int npages);
>
> /**
>   * Free a request
> diff --git a/fs/fuse/inode.c b/fs/fuse/inode.c
> index ce0a283..3f399ba 100644
> --- a/fs/fuse/inode.c
> +++ b/fs/fuse/inode.c
> @@ -1027,12 +1027,12 @@ static int fuse_fill_super(struct super_block *sb, void *data, int
silent)
>   /* only now - we want root dentry with NULL ->d_op */
>   sb->s_d_op = &fuse_dentry_operations;
>
> - init_req = fuse_request_alloc();
> + init_req = fuse_request_alloc(0);
>   if (!init_req)
>   goto err_put_root;
>
>   if (is_bdev) {
> - fc->destroy_req = fuse_request_alloc();
> + fc->destroy_req = fuse_request_alloc(0);
>   if (!fc->destroy_req)
>   goto err_free_init_req;
>   }

Subject: Re: [PATCH 2/6] fuse: categorize fuse_get_req()
Posted by Miklos Szeredi on Wed, 12 Sep 2012 16:20:15 GMT
View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

> The patch categorizes all fuse_get_req() invocations into three categories:
> - fuse_get_req_nopages() - when caller doesn't care about req->pages
> - fuse_get_req_onepage() - when caller need exactly one page
> - fuse_get_req_multipage() - in other cases.

I think you are overcomplicating this.

Just have two functions: fuse_get_req_nopages() and fuse_get_req().

I don't think the _onepage variant deserves a separate helper.

Thanks,
Miklos

Subject: Re: [PATCH 3/6] fuse: rework fuse_retrieve()
Posted by Miklos Szeredi on Wed, 12 Sep 2012 16:23:55 GMT
View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

> The patch reworks fuse_retrieve() to allocate only so many page pointers
> as needed. The core part of the patch is the following calculation:
>
> num_pages = (num + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
>
> (thanks Miklos for formula). All other changes are moslty shuffling lines.
>
> The patch also contains one-line fix suggested by Miklos: the loop iterating
> over pages from inode mapping should interpret 'offset' as offset in the very
> first page only.

That fix is already in -linus, so let's leave it out of this patch.

Thanks,
Miklos

> ---
> fs/fuse/dev.c | 26 ++++++++++++++++++++----------
> 1 files changed, 16 insertions(+), 10 deletions(-)
>
> diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
> index 6e38af5..cf69df3 100644
> --- a/fs/fuse/dev.c
> +++ b/fs/fuse/dev.c
> @@ -1562,13 +1562,24 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
>   unsigned int num;
>   unsigned int offset;
>   size_t total_len = 0;
> + int num_pages;
>
> - req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
> + offset = outarg->offset & ~PAGE_CACHE_MASK;
> + file_size = i_size_read(inode);
> +
> + num = outarg->size;
> + if (outarg->offset > file_size)
> + num = 0;
> + else if (outarg->offset + num > file_size)
> + num = file_size - outarg->offset;
> +
> + num_pages = (num + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
> + num_pages = min(num_pages, FUSE_MAX_PAGES_PER_REQ);
> +

> + req = fuse_get_req_multipage(fc, num_pages);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> - offset = outarg->offset & ~PAGE_CACHE_MASK;
> -
>   req->in.h.opcode = FUSE_NOTIFY_REPLY;
>   req->in.h.nodeid = outarg->nodeid;
>   req->in.numargs = 2;
> @@ -1577,14 +1588,8 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
>   req->end = fuse_retrieve_end;
>
>   index = outarg->offset >> PAGE_CACHE_SHIFT;
> - file_size = i_size_read(inode);
> - num = outarg->size;
> - if (outarg->offset > file_size)
> - num = 0;
> - else if (outarg->offset + num > file_size)
> - num = file_size - outarg->offset;
>
> - while (num && req->num_pages < FUSE_MAX_PAGES_PER_REQ) {
> + while (num && req->num_pages < num_pages) {
>   struct page *page;
>   unsigned int this_num;
>
> @@ -1596,6 +1601,7 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
>   req->pages[req->num_pages] = page;
>   req->num_pages++;
>
> + offset = 0;
>   num -= this_num;
>   total_len += this_num;
>   index++;

Subject: Re: [PATCH 4/6] fuse: rework fuse_readpages()
Posted by Miklos Szeredi on Wed, 12 Sep 2012 16:41:40 GMT
View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

> The patch uses 'nr_pages' argument of fuse_readpages() as the heuristics for
> number of page pointers to allocate.
>
> This can be improved further by taking in consideration fc->max_read and gaps
> between page indices, but it's not clear whetheher it's worthy or not.
> ---
> fs/fuse/file.c | 10 ++++++++--

> 1 files changed, 8 insertions(+), 2 deletions(-)
>
> diff --git a/fs/fuse/file.c b/fs/fuse/file.c
> index c811f3d..9a6dcc6 100644
> --- a/fs/fuse/file.c
> +++ b/fs/fuse/file.c
> @@ -641,6 +641,7 @@ struct fuse_fill_data {
>   struct fuse_req *req;
>   struct file *file;
>   struct inode *inode;
> + unsigned nr_pages;
> };
>
> static int fuse_readpages_fill(void *_data, struct page *page)
> @@ -656,8 +657,10 @@ static int fuse_readpages_fill(void *_data, struct page *page)
>   (req->num_pages == FUSE_MAX_PAGES_PER_REQ ||
>   (req->num_pages + 1) * PAGE_CACHE_SIZE > fc->max_read ||
>   req->pages[req->num_pages - 1]->index + 1 != page->index)) {
> + int nr_alloc = min_t(unsigned, data->nr_pages,
> +   FUSE_MAX_PAGES_PER_REQ);
>   fuse_send_readpages(req, data->file);
> - data->req = req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
> + data->req = req = fuse_get_req_multipage(fc, nr_alloc);
>   if (IS_ERR(req)) {
>   unlock_page(page);
>   return PTR_ERR(req);
> @@ -666,6 +669,7 @@ static int fuse_readpages_fill(void *_data, struct page *page)
>   page_cache_get(page);

Okay, this is where things get hairy and where we should do something like:

 if (WARN_ON(req->num_pages >= req->max_pages))
  return -EIO;

>   req->pages[req->num_pages] = page;
>   req->num_pages++;
> + data->nr_pages--;
>   return 0;
> }
>
> @@ -676,6 +680,7 @@ static int fuse_readpages(struct file *file, struct address_space
*mapping,
>   struct fuse_conn *fc = get_fuse_conn(inode);
>   struct fuse_fill_data data;
>   int err;
> + int nr_alloc = min_t(unsigned, nr_pages, FUSE_MAX_PAGES_PER_REQ);
>

>   err = -EIO;
>   if (is_bad_inode(inode))
> @@ -683,7 +688,8 @@ static int fuse_readpages(struct file *file, struct address_space
*mapping,
>
>   data.file = file;
>   data.inode = inode;
> - data.req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
> + data.req = fuse_get_req_multipage(fc, nr_alloc);
> + data.nr_pages = nr_pages;
>   err = PTR_ERR(data.req);
>   if (IS_ERR(data.req))
>   goto out;

Subject: Re: [PATCH 5/6] fuse: rework fuse_perform_write()
Posted by Miklos Szeredi on Wed, 12 Sep 2012 16:43:54 GMT
View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

> The patch allocates as many page pointers in fuse_req as needed to cover
> interval [pos .. pos+len-1]. FUSE_WR_PAGES macro is introduced to hide this
> cumbersome arithmetics.

Please don't ever use a macro for something which you can use an inline
function.

> ---
> fs/fuse/file.c | 15 ++++++++++----
> 1 files changed, 11 insertions(+), 4 deletions(-)
>
> diff --git a/fs/fuse/file.c b/fs/fuse/file.c
> index 9a6dcc6..84cc83c 100644
> --- a/fs/fuse/file.c
> +++ b/fs/fuse/file.c
> @@ -815,7 +815,8 @@ static size_t fuse_send_write_pages(struct fuse_req *req, struct file
*file,
>
> static ssize_t fuse_fill_write_pages(struct fuse_req *req,
>   struct address_space *mapping,
> -   struct iov_iter *ii, loff_t pos)
> +   struct iov_iter *ii, loff_t pos,
> +   unsigned nr_pages)
> {
>   struct fuse_conn *fc = get_fuse_conn(mapping->host);
>   unsigned offset = pos & (PAGE_CACHE_SIZE - 1);
> @@ -875,11 +876,16 @@ static ssize_t fuse_fill_write_pages(struct fuse_req *req,

```
>    if (!fc->big_writes)
>     break;
>   } while (iov_iter_count(ii) && count < fc->max_write &&
> -   req->num_pages < FUSE_MAX_PAGES_PER_REQ && offset == 0);
> +   req->num_pages < nr_pages && offset == 0);
>
>   return count > 0 ? count : err;
>  }
>
> +#define FUSE_WR_PAGES(pos, len) min_t(unsigned,          \
> +          ((pos + len - 1) >> PAGE_CACHE_SHIFT) - \
> +          (pos >> PAGE_CACHE_SHIFT) + 1,       \
> +          FUSE_MAX_PAGES_PER_REQ)
> +
>  static ssize_t fuse_perform_write(struct file *file,
>         struct address_space *mapping,
>         struct iov_iter *ii, loff_t pos)
> @@ -895,14 +901,15 @@ static ssize_t fuse_perform_write(struct file *file,
>   do {
>    struct fuse_req *req;
>    ssize_t count;
> +  unsigned nr_pages = FUSE_WR_PAGES(pos, iov_iter_count(ii));
>
> -  req = fuse_get_req_multipage(fc, FUSE_MAX_PAGES_PER_REQ);
> +  req = fuse_get_req_multipage(fc, nr_pages);
>    if (IS_ERR(req)) {
>     err = PTR_ERR(req);
>     break;
>    }
>
> -  count = fuse_fill_write_pages(req, mapping, ii, pos);
> +  count = fuse_fill_write_pages(req, mapping, ii, pos, nr_pages);
>    if (count <= 0) {
>     err = count;
>    } else {
```

Subject: Re: [PATCH 0/6] fuse: allocate req->pages[] dynamically
Posted by Miklos Szeredi on Wed, 12 Sep 2012 16:49:01 GMT
View Forum Message <> Reply to Message

"Maxim V. Patlasov" <mpatlasov@parallels.com> writes:

> Hi Miklos,
>
> So far as no objections appeared, I'll go ahead and replace fuse req->page with
> req->pagevec. It will point to an array of structs:
>

> struct page_vec {
>   struct page   *pv_page;
>   unsigned int   pv_len;
>   unsigned int   pv_offset;
> };
>
> instead of 'struct page *' as it used to be. It seems to be what you suggested
> in one of your comments. Are you OK about it?

Yes, that's exactly what I was thinking.

Thanks,
Miklos

Subject: Re: [PATCH 0/6] fuse: allocate req->pages[] dynamically
Posted by Maxim Patlasov on Fri, 14 Sep 2012 13:18:17 GMT
View Forum Message <> Reply to Message

Hi Miklos,

> "Maxim V. Patlasov" <mpatlasov@parallels.com> writes:
>
>> Hi Miklos,
>>
>> So far as no objections appeared, I'll go ahead and replace fuse req->page with
>> req->pagevec. It will point to an array of structs:
>>
>> struct page_vec {
>>   struct page   *pv_page;
>>   unsigned int   pv_len;
>>   unsigned int   pv_offset;
>> };
>>
>> instead of 'struct page *' as it used to be. It seems to be what you suggested
>> in one of your comments. Are you OK about it?
> Yes, that's exactly what I was thinking.

I've encountered a problem while trying to follow this approach.
fuse_get_user_pages() passes 'req->pages' to get_user_pages_fast().
get_user_pages_fast() and friends are not ready to get a pointer to
array of page_vec-s from fuse. I can see five ways to solve the problem:

1. Re-work get_user_pages_fast() and friends adding ability to fill
page_vec array. Too much work. Very ugly. I strongly dislike this way.

2. Allocate a temporary array of page pointers in fuse_get_user_pages()

to use as argument to get_user_pages_fast(). Ugly and may have
performance impact. I dislike this way too.

3. Call get_user_pages_fast() for each page (i.e. pass npages == 1 to
it). Easy to implement but may have performance impact. I'd refrain from it.

4. Keep req->pages 'as is', but add req->page_descs pointing to an array
of <offset, len> structures. Looks clumsy, straightforward, but quite
doable.

5. Use a hack in fuse_get_user_pages(): temporarily cast req->pagevecs
to 'struct page **pages', pass it get_user_pages_fast(), then transform
the content of req->pagevecs[] to have page pointers stored in proper
places (like 'for (i=...) pagevecs[i].pv_page = pages[i];').

What do you think?

Btw, thanks a lot for careful review of patch-set. I agree with your
comments. Next version will have those findings fixed.

Thanks,
Maxim

Subject: Re: [PATCH 0/6] fuse: allocate req->pages[] dynamically
Posted by Miklos Szeredi on Fri, 14 Sep 2012 14:39:44 GMT
View Forum Message <> Reply to Message

"Maxim V. Patlasov" <mpatlasov@parallels.com> writes:

> Hi Miklos,
>

>> "Maxim V. Patlasov" <mpatlasov@parallels.com> writes:
>>
>>> Hi Miklos,
>>>
>>> So far as no objections appeared, I'll go ahead and replace fuse req->page with
>>> req->pagevec. It will point to an array of structs:
>>>
>>> struct page_vec {
>>>   struct page   *pv_page;
>>>   unsigned int   pv_len;
>>>   unsigned int   pv_offset;
>>> };
>>>
>>> instead of 'struct page *' as it used to be. It seems to be what you suggested
>>> in one of your comments. Are you OK about it?

>> Yes, that's exactly what I was thinking.
>
> I've encountered a problem while trying to follow this
> approach. fuse_get_user_pages() passes 'req->pages' to
> get_user_pages_fast(). get_user_pages_fast() and friends are not ready to get a
> pointer to array of page_vec-s from fuse. I can see five ways to solve the
> problem:
>
> 1. Re-work get_user_pages_fast() and friends adding ability to fill page_vec
> array. Too much work. Very ugly. I strongly dislike this way.
>
> 2. Allocate a temporary array of page pointers in fuse_get_user_pages() to use
> as argument to get_user_pages_fast(). Ugly and may have performance impact. I
> dislike this way too.
>
> 3. Call get_user_pages_fast() for each page (i.e. pass npages == 1 to it). Easy
> to implement but may have performance impact. I'd refrain from it.
>
> 4. Keep req->pages 'as is', but add req->page_descs pointing to an array of
> <offset, len> structures. Looks clumsy, straightforward, but quite
> doable.
>
> 5. Use a hack in fuse_get_user_pages(): temporarily cast req->pagevecs to
> struct page **pages', pass it get_user_pages_fast(), then transform the content
> of req->pagevecs[] to have page pointers stored in proper places (like 'for
> (i=...) pagevecs[i].pv_page = pages[i];').
>
> What do you think?

I'd go for number 4.

Thanks,
Miklos
