get_user_pages() + DMA

Linux Plumbers Conference: 13-15 Nov, 2018

Jan Kara

Mathew Wilcox

Dan Williams

John Hubbard

Goal of this session

Awareness of the problem (and of intrusive patchsets coming soon)

Consensus on the long-term solution

Preferences for steps on getting there

Kernel crash via common use cases

Application pins pages, does DMA on the pages, marks pages dirty, and releases pages.

```
va = mmap(…, MAP_SHARED, file1); // file-backed pages
```

This pattern has been [mis]understood to be OK by many kernel engineers, since 2005 or so

The problem: If the pages are backed by a non-RAM-based filesystem, it’s all just broken.

Crash example from the field

? page_mkclean+0x73/0xa0
? page_referenced_one+0x1a0/0x1a0
__writepage+0x12/0x30
write_cache_pages+0x1ee/0x510
? ext4_writepage+0x590/0x590
ext4_writepages+0x195/0xd30
? swiotlb_map_sg_attrs+0x70/0x150
? find_next_bit+0x15/0x20
? cpumask_next_and+0x2f/0x40
? __enqueue_entity+0x6c/0x70
? enqueue_entity+0x3a7/0xd10
do_writepages+0x1e/0x30
__writeback_single_inode+0x45/0x340
? enqueue_task_fair+0xaa/0x8b0
writeback_sb_inodes+0x262/0x600
__writeback_inodes_wb+0x8c/0xc0
wb_writeback+0x253/0x310
wb_workfn+0x2ea/0x400

Root cause

page buffers get stripped while page is pinned

…but writeback expects page buffers to still be there

→ Root cause: filesystem is unaware that a page can become dirty outside of write_begin/write_end, or outside of page_mkwrite/writepage.

In other words, RDMA to file-backed memory is **not supported** today!

https://lwn.net/Articles/753027/ : "The Trouble with get_user_pages()"

What happened…in more detail (part 1)

va = mmap(…, MAP_SHARED, file1); // file-backed pages

get_user_pages_fast()

page fault

->page_mkwrite()

block_page_mkwrite()

lock_page(page);

attaches buffers to page

makes sure blocks are allocated set_page_dirty(page)

install writeable PTE

unlock_page(page);

What happened…in more detail (part 2)

https://www.spinics.net/lists/linux-mm/msg142700.html (Jan's notes)

kswapd reclaims pages:

shrink_page_list()

trylock_page(page) - this is the page CPU1 has just faulted in

try_to_unmap(page)

pageout(page);

clear_page_dirty_for_io(page);

->writepage()

if (page_has_private(page)) { try_to_release_page(page) // reclaims buffers from the page

__remove_mapping(page) // fails because get_user_pages() still holds page reference

What happened…in more detail (part 3)

set_page_dirty_lock()

put_page()

…kernel crash soon, due to writeback not finding any page buffers

Proposed solution, part 1

http://lkml.kernel.org/r/20181110085041.10071-1-jhubbard@nvidia.com

Part 1: Track (refcount) pages that are get_user_pages()-pinned.

This means new calls: put_user_page(), put_user_pages()

Use page->lru (.next / .prev) fields for flags, refcount: remove pages from LRU, while they are pinned

Proposed solution, part 2

http://lkml.kernel.org/r/20181110085041.10071-1-jhubbard@nvidia.com

Part 2: convert the call sites. There are about 100:

> https://docs.google.com/spreadsheets/d/1qqApTcjt22j-EjFH2NAMqf92QoyQ-v-0f_b4cMpnDt4/edit?usp=sharing
>
> Invoke put_user_page(), put_user_pages(), instead of put_page() or release_pages()
>
> drm, bio and a few other subsystems have complex mix of normal and gup pages

Existing RFC patchset converts Infiniband, as an example

Proposed solution, part 3

Part 3: use the new information

For gup-pinned pages:

- Hold off try_to_unmap()
- Allow writeback while pinned (via bounce buffers)
- Use revocable reservations, in some cases (umount…?)

Performance notes: fio results

fio: https://github.com/axboe/fio

Baseline: 52 – 56 MB/s

With put_user_page*(): 49 – 52 MB/s

Performance notes: batching: put_user_pages()

Send the entire set of pages, where possible: enable future optimizations

SGL (scatter-gather list) support