author	Johannes Thumshirn <johannes.thumshirn@wdc.com>	2025-03-17 16:04:01 +0100
committer	David Sterba <dsterba@suse.com>	2025-04-17 11:56:19 +0200
commit	b0c26f47992672661340dd6ea931240213016609 (patch)	
tree	23ef234aa14045dfa58de36f99a12189147adddb	
parent	7d82240c457fc15abdf7dedf15104cea774b005b (diff)	
download	linux-b0c26f47992672661340dd6ea931240213016609.tar.gz	

btrfs: zoned: return EIO on RAID1 block group write pointer mismatch

```
There was a bug report about a NULL pointer dereference in
__btrfs_add_free_space_zoned() that ultimately happens because a
conversion from the default metadata profile DUP to a RAID1 profile on two
disks.

The stack trace has the following signature:

  BTRFS error (device sdc): zoned: write pointer offset mismatch of zones in raid1 profile
  BUG: kernel NULL pointer dereference, address: 0000000000000058
  #PF: supervisor read access in kernel mode
  #PF: error_code(0x0000) - not-present page
  PGD 0 P4D 0
  Oops: Oops: 0000 [#1] PREEMPT SMP NOPTI
  RIP: 0010:__btrfs_add_free_space_zoned.isra.0+0x61/0x1a0
  RSP: 0018:ffffa236b6f3f6d0 EFLAGS: 00010246
  RAX: 0000000000000000 RBX: ffff96c8132f3400 RCX: 0000000000000001
  RDX: 0000000010000000 RSI: 0000000000000000 RDI: ffff96c8132f3410
  RBP: 0000000010000000 R08: 0000000000000003 R09: 0000000000000000
  R10: 0000000000000000 R11: 00000000ffffffff R12: 0000000000000000
  R13: ffff96c758f65a40 R14: 0000000000000001 R15: 000011aac0000000
  FS: 00007fdab1cb2900(0000) GS:ffff96e60ca00000(0000) knlGS:0000000000000000
  CS: 0010 DS: 0000 ES: 0000 CR0: 0000000080050033
  CR2: 0000000000000058 CR3: 00000001a05ae000 CR4: 0000000000350ef0
  Call Trace:
  <TASK>
  ? __die_body.cold+0x19/0x27
  ? page_fault_oops+0x15c/0x2f0
  ? exc_page_fault+0x7e/0x180
  ? asm_exc_page_fault+0x26/0x30
  ? __btrfs_add_free_space_zoned.isra.0+0x61/0x1a0
  btrfs_add_free_space_async_trimmed+0x34/0x40
  btrfs_add_new_free_space+0x107/0x120
  btrfs_make_block_group+0x104/0x2b0
  btrfs_create_chunk+0x977/0xf20
  btrfs_chunk_alloc+0x174/0x510
  ? srso_return_thunk+0x5/0x5f
  btrfs_inc_block_group_ro+0x1b1/0x230
  btrfs_relocate_block_group+0x9e/0x410
  btrfs_relocate_chunk+0x3f/0x130
  btrfs_balance+0x8ac/0x12b0
  ? srso_return_thunk+0x5/0x5f
  ? srso_return_thunk+0x5/0x5f
  ? __kmalloc_cache_noprof+0x14c/0x3e0
  btrfs_ioctl+0x2686/0x2a80
  ? srso_return_thunk+0x5/0x5f
  ? ioctl_has_perm.constprop.0.isra.0+0xd2/0x120
  __x64_sys_ioctl+0x97/0xc0
  do_syscall_64+0x82/0x160
  ? srso_return_thunk+0x5/0x5f
```

```
  ? __memcg_slab_free_hook+0x11a/0x170
  ? srso_return_thunk+0x5/0x5f
  ? kmem_cache_free+0x3f0/0x450
  ? srso_return_thunk+0x5/0x5f
  ? srso_return_thunk+0x5/0x5f
  ? syscall_exit_to_user_mode+0x10/0x210
  ? srso_return_thunk+0x5/0x5f
  ? do_syscall_64+0x8e/0x160
  ? sysfs_emit+0xaf/0xc0
  ? srso_return_thunk+0x5/0x5f
  ? srso_return_thunk+0x5/0x5f
  ? seq_read_iter+0x207/0x460
  ? srso_return_thunk+0x5/0x5f
  ? vfs_read+0x29c/0x370
  ? srso_return_thunk+0x5/0x5f
  ? srso_return_thunk+0x5/0x5f
  ? syscall_exit_to_user_mode+0x10/0x210
  ? srso_return_thunk+0x5/0x5f
  ? do_syscall_64+0x8e/0x160
  ? srso_return_thunk+0x5/0x5f
  ? exc_page_fault+0x7e/0x180
  entry_SYSCALL_64_after_hwframe+0x76/0x7e
  RIP: 0033:0x7fdab1e0ca6d
  RSP: 002b:00007ffeb2b60c80 EFLAGS: 00000246 ORIG_RAX: 0000000000000010
  RAX: ffffffffffffffda RBX: 0000000000000003 RCX: 00007fdab1e0ca6d
  RDX: 00007ffeb2b60d80 RSI: 00000000c4009420 RDI: 0000000000000003
  RBP: 00007ffeb2b60cd0 R08: 0000000000000000 R09: 0000000000000013
  R10: 0000000000000000 R11: 0000000000000246 R12: 0000000000000000
  R13: 00007ffeb2b6343b R14: 00007ffeb2b60d80 R15: 0000000000000001
  </TASK>
  CR2: 0000000000000058
  ---[ end trace 0000000000000000 ]---
```

The 1st line is the most interesting here:

```
 BTRFS error (device sdc): zoned: write pointer offset mismatch of zones in raid1 profile
```

When a RAID1 block-group is created and a write pointer mismatch between
the disks in the RAID set is detected, btrfs sets the alloc_offset to the
length of the block group marking it as full. Afterwards the code expects
that a balance operation will evacuate the data in this block-group and
repair the problems.

But before this is possible, the new space of this block-group will be
accounted in the free space cache. But in __btrfs_add_free_space_zoned()
it is being checked if it is a initial creation of a block group and if
not a reclaim decision will be made. But the decision if a block-group's
free space accounting is done for an initial creation depends on if the
size of the added free space is the whole length of the block-group and
the allocation offset is 0.

But as btrfs_load_block_group_zone_info() sets the allocation offset to
the zone capacity (i.e. marking the block-group as full) this initial
decision is not met, and the space_info pointer in the 'struct
btrfs_block_group' has not yet been assigned.

Fail creation of the block group and rely on manual user intervention to
re-balance the filesystem.

Afterwards the filesystem can be unmounted, mounted in degraded mode and
the missing device can be removed after a full balance of the filesystem.

Reported-by: 西木野崴基 <yanqiyu01@gmail.com>
Link: https://lore.kernel.org/linux-btrfs/CAB_b4sBhDe3tscz=duVyhc9hNE+gu=B8CrgLO152uMyanR8BEA@mail.gmail.com/
Fixes: b1934cd60695 ("btrfs: zoned: handle broken write pointer on zones")
Reviewed-by: Anand Jain <anand.jain@oracle.com>
Signed-off-by: Johannes Thumshirn <johannes.thumshirn@wdc.com>
Signed-off-by: David Sterba <dsterba@suse.com>
```
```

Diffstat

-rw-r--r-- fs/btrfs/zoned.c 1

1 files changed, 0 insertions, 1 deletions

```
diff --git a/fs/btrfs/zoned.c b/fs/btrfs/zoned.c
index fb8b8b29c169ca..7c502192cd6bef 100644
--- a/fs/btrfs/zoned.c
+++ b/fs/btrfs/zoned.c
@@ -1659,7 +1659,6 @@ int btrfs_load_block_group_zone_info(struct btrfs_block_group *cache, bool new)
                 * stripe.
                 */
                cache->alloc_offset = cache->zone_capacity;
-               ret = 0;
        }

 out:
```