CMA allocations fail due to pinned MOVABLE pages

Juan Yescas & Kalesh Singh

Google

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

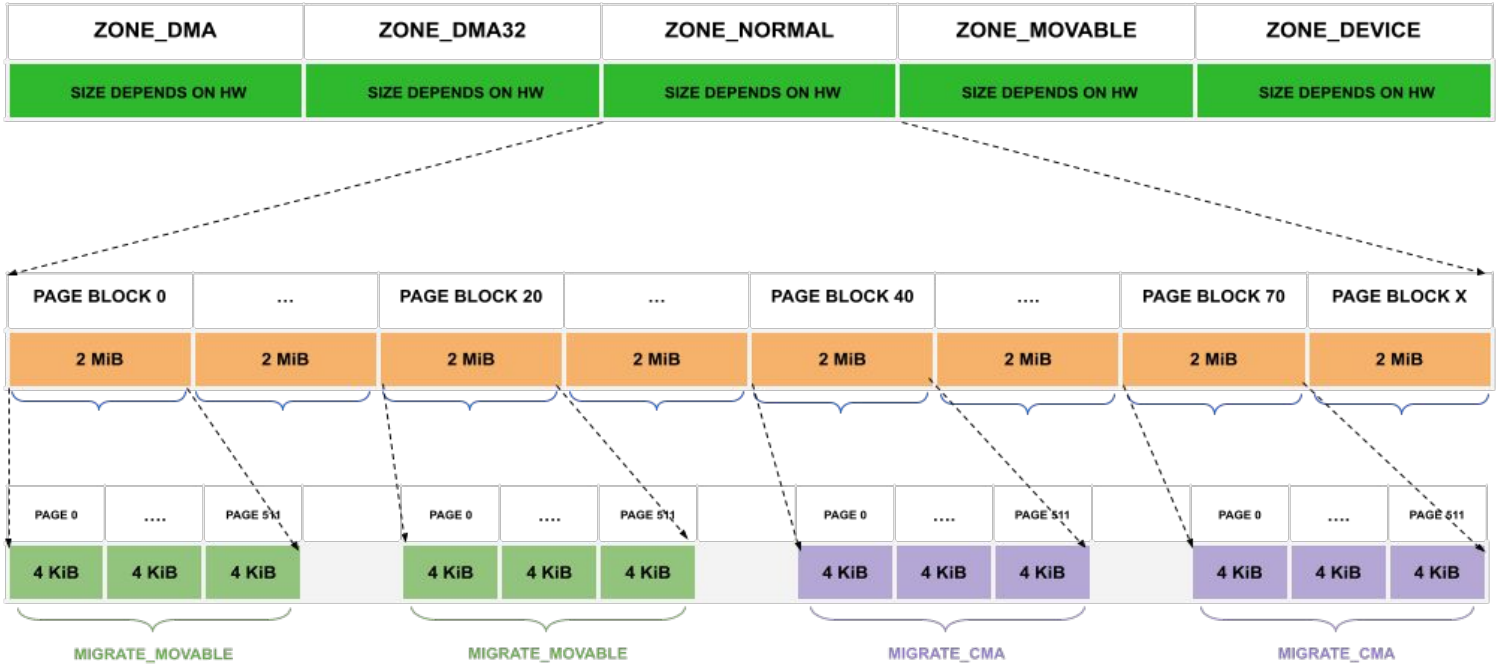
CMA Usage

- Android kernels are configured with up to 32 CMA regions (`CONFIG_CMA_AREAS=32`).

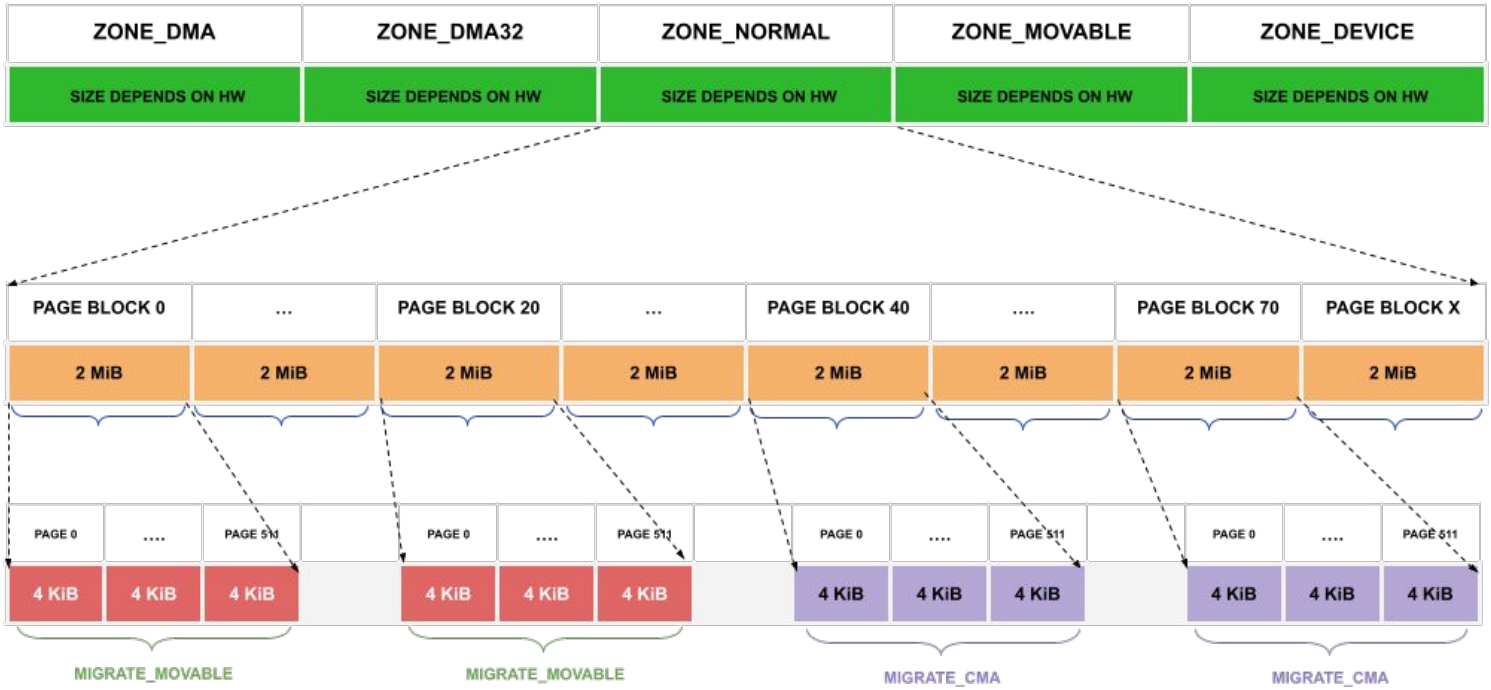
- **CMA regions (2GB+ / 20% of RAM)**

Problem

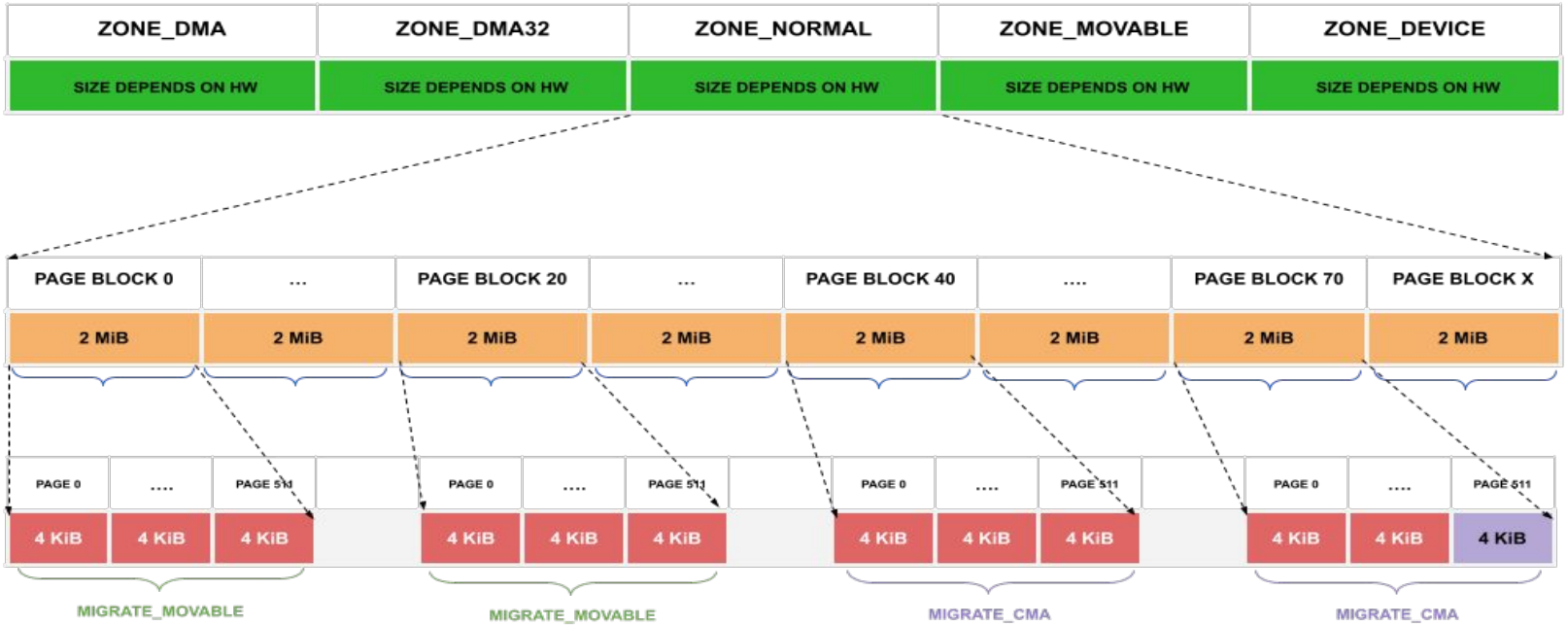
Act 1: Enough free MOVABLE and CMA memory



Act 2: Start running out of MOVABLE pages (e.g. O_DIRECT)



Act 3: Use CMA pages as fallback for MOVABLE pages (e.g. O_DIRECT)



Act 4: Camera app requests 254 MiB CMA memory

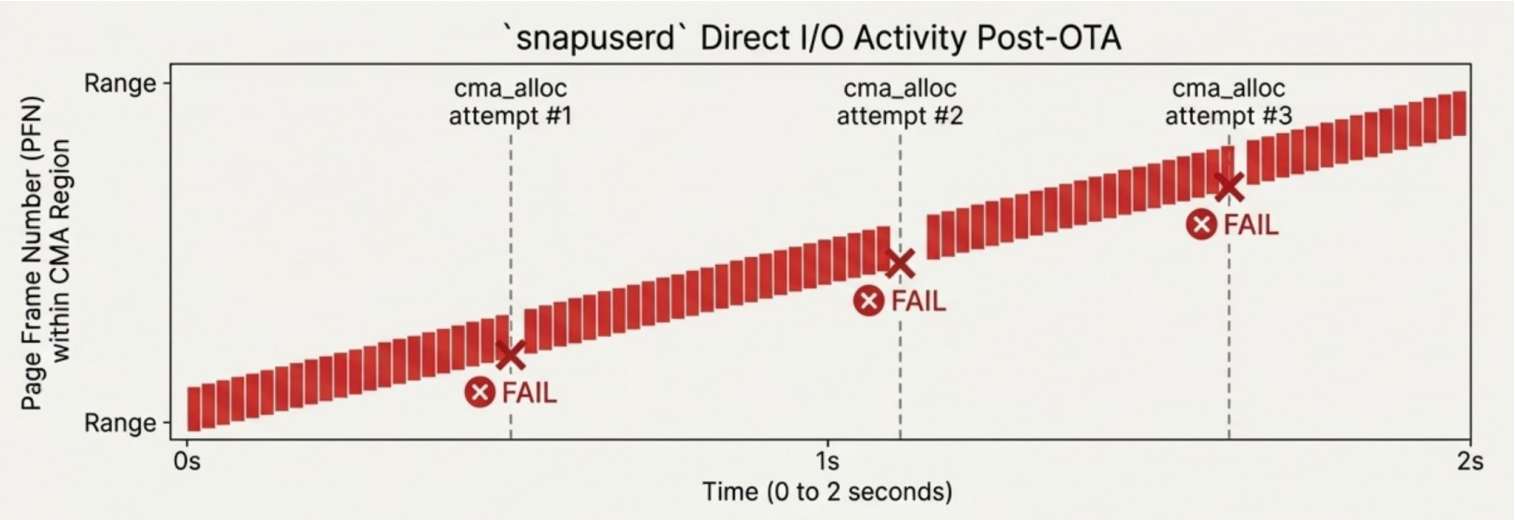
- Not enough CMA memory
- CMA allocation fails
- Page migration fails because many pages have been short pinned due `O_DIRECT`.

Concrete example

OTA updates

1. OTA requested big chunks of `MOVABLE` memory in a short period of time with `O_DIRECT` flag
2. OTA allocations start coming from CMA
3. OTA consumes most of the CMA memory
4. Camera requests CMA memory but there is not available
5. Reclaim starts but it is not able to free enough CMA memory
6. Allocation fails

Rolling Blockade

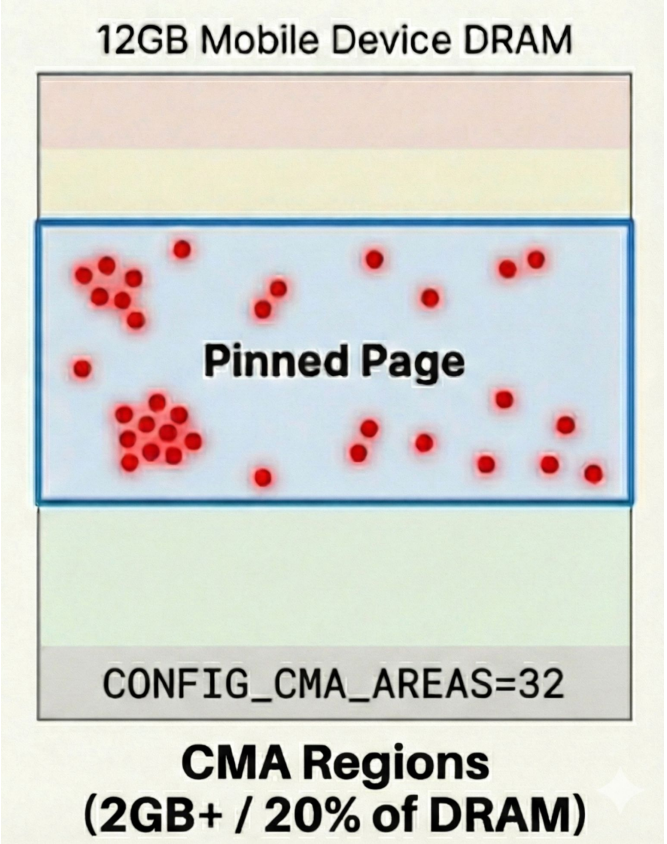


- On first boot after OTA, snapuserd requests a huge amount of direct IO reads which might occasionally disturb CMA allocations.

Rolling Blockade

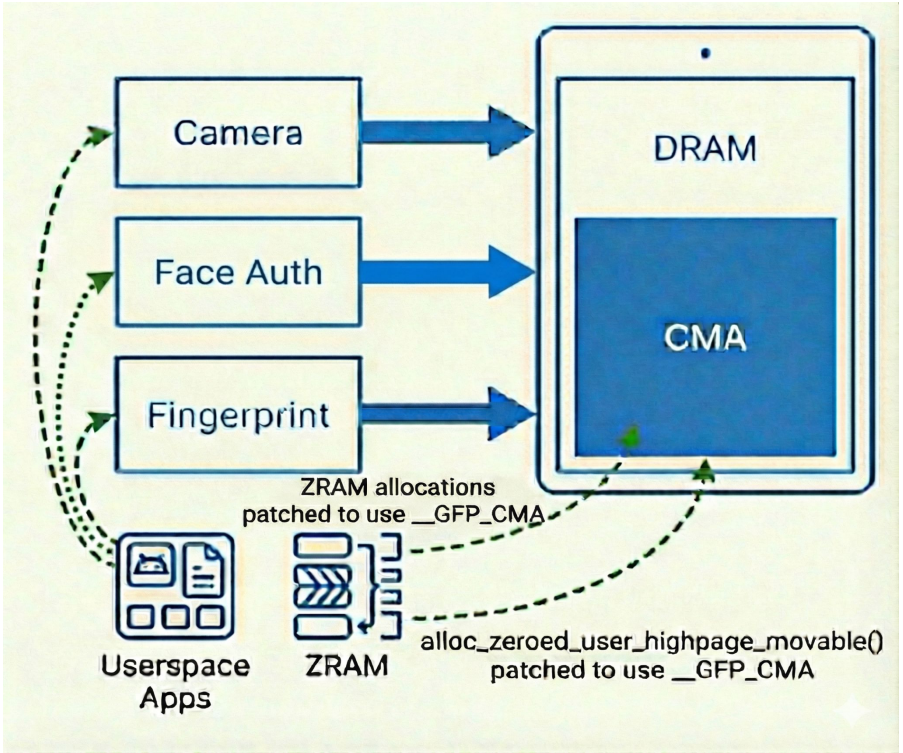
- No single page is pinned for a long time.
- Each pin is a textbook "short-term" pin.
- But the relentless stream of I/O creates a rolling blockade.
- Every time `cma_alloc` tries to find a contiguous block, it hits a different, freshly pinned page.
- Can result in a denial of service for seconds at a time.

Android


12GB Mobile Device DRAM
Pinned Page
CONFIG_CMA_AREAS=32
CMA Regions
(2GB+ / 20% of DRAM)

CMA Utilization

- Lots of free CMA memory but **non-movable allocation fails**.
- Android Kernel Strategy: Increase CMA utilization by redirecting some userspace and ZRAM allocations into CMA.
  - `alloc_zeroed_user_highpage_movable()` patched to use `__GFP_CMA`
  - ZRAM allocations patched to use `__GFP_CMA`
- Improves utilization but creates other conflicts.



Semantics vs Reality

- An RFC attempted to fix this by classifying Direct I/O pages as **`FOLL_LONGTERM`**.

  [RFC PATCH] block, fs: use FOLL_LONGTERM as gup_flags for direct IO

- The rejection from upstream was swift and, architecturally, correct. A short I/O pin is not a long-term pin.

- We agree with the semantics. But the on-device reality is a denial of service.

Current Workarounds

- Retry **`cma_range_alloc()` `5 times`** with a timeout of **`100ms`**

  https://r.android.com/3616623

- Still observe CMA failures in the field.

```
@@ -783,6 +785,8 @@ static int cma_range_alloc(struct cma *cma, struct cma_memrange *cmr,
    unsigned long bitmap_maxno, bitmap_no, bitmap_count;
    int ret = -EBUSY;
    struct page *page = NULL;
+   int num_attempts = 0;
+   int max_retries = 5;

    mask = cma_bitmap_aligned_mask(cma, align);
    offset = cma_bitmap_aligned_offset(cma, cmr, align);
@@ -806,8 +810,28 @@ static int cma_range_alloc(struct cma *cma, struct cma_memrange *cmr,
                bitmap_maxno, start, bitmap_count, mask,
                offset);
        if (bitmap_no >= bitmap_maxno) {
-           spin_unlock_irq(&cma->lock);
-           break;
+           if ((num_attempts < max_retries) && (ret == -EBUSY)) {
+               spin_unlock_irq(&cma->lock);
+
+               if (fatal_signal_pending(current))
+                   break;
+
+               /*
+                * Page may be momentarily pinned by some other
+                * process which has been scheduled out, e.g.
+                * in exit path, during unmap call, or process
+                * fork and so cannot be freed there. Sleep
+                * for 100ms and retry the allocation.
+                */
+               start = 0;
+               ret = -ENOMEM;
+               schedule_timeout_killable(msecs_to_jiffies(100));
+               num_attempts++;
+               continue;
+           } else {
+               spin_unlock_irq(&cma->lock);
+               break;
+           }
        }
        bitmap_set(cmr->bitmap, bitmap_no, bitmap_count);
```

Discussion

Discussion

- **Pin Awareness:** Can `GUP` coordinate with `CMA` migration?
  (Inspired by David Hildenbrand's suggestion: "make `GUP` spin or wait for migration to end")
  https://lore.kernel.org/all/cd013218-7735-4bc1-b6b6-80d1129e2b76@redhat.com/

- **Quarantine:** Can we ban certain allocation types (e.g., FS bdev buffer head, Direct I/O) from falling back to `CMA` without abusing `FOLL_LONGTERM`?

- **Strict Mode:** Do we need a `MIGRATE_CMA_STRICT` type that is never a fallback target for `MIGRATE_MOVABLE`?

Upstream memory debug module ??

```
$ ls /sys/kernel/debug/mm/node-0/
node-0          node-1

$ ls /sys/kernel/debug/mm/node-0/zone-Normal/order-11
migrate-CMA          migrate-Isolate      migrate-Reclaimable
migrate-HighAtomic   migrate-Movable      migrate-Unmovable

$ ls /sys/kernel/debug/mm/node-0/zone-Normal/order-11/migrate-Movable
alloc

$ echo 1 >
/sys/kernel/debug/mm/node-0/zone-Normal/order-11/migrate-Movable/alloc
```

Thanks