CMA ALLOCATION FAILURE ANALYSIS

I.MX CAS

GOPISE YUAN

JUN/2022

Issue description:

Sometimes, we may see some CMA allocation errors from kernel log, with different reasons. The most commonly seen one would be:

```
cma     : cma_alloc: alloc failed, req-size: 456 pages, ret: -12
```

"-12" here is "-ENOMEM" which means "no memory".

Of course, some other errors, such as "-16" which means "busy", also exists. But, "busy" will normally only cause a delay in allocation, but not a direct failure.

Basic CMA mechanism:

The CMA actually "shares" memory pages with system memory area.

When it's required to alloc a new block, it will check if there's a "clean" block which can satisfy the alloc. If not, It will try to grab some pages from its coverage range but currently occupied by system memory. This is called "migration". Migration will move existing content from the CMA area to other area and try to make a free continuous block. This costs time and is the major cause for alloc delay.

After running for some time, the CMA will become fragmented due to many small alloc/release requests. Unfortunately, CMA memory normally used as hardware related buffer, especially DMA buffer. So, we can not do "defragment" like the virtual memory do.

Failure case:

What might be un-expected is, we may find there're still many CMA free pages when it complain with NOMEM.

An example case:

When free CMA is 500MB/1000MB, but still got -12 when call alloc.

Why?

Several possible reasons for this kind of failure.

Reason #1: System memory low

- When the system memory is also very low, if a CMA request trigger a migration process from system to CMA, the migration will fail and subsequently cause CMA alloc failure.
- This normally found on systems with small total memory and the divide between CMA and system is impropriate (too much CMA).

- BTW, under this kind of situation (System memory low and CMA requires more), the swap might be triggered, and, if there's no swap configured in system, the "kswapd" task might be abnormal and consume a lot of CPU resources.
- We can check the system memory status as well as CMA free pages to determine if we fall into this case.

Reason #2: Fragmentation

- When CMA memory is heavily fragmented. Under this case, there may be lot of free memory in system area and also enough free pages in CMA area. But checking the bitmap of the CMA, there might be no such big blocks to satisfy the request. All free blocks are smaller and all are separated (fragmented).
- This happens when alloc a big (e.g. 1024 pages) block under a fragmented case.
- We can check the free page bitmap in CMA to determine if the failure caused by this.

Reason #3: Alignment

- The alignment of the request can not be satisfied. Under a fragmented case, we may still find some continuous blocks which might be bigger than requested but the alloc may still fail.
- This may be caused by alignment of the request.
- For example (log truncated):

```
cma     : cma_alloc(cma ffff0000091a4168, count 456, align 8)
cma     : cma_alloc(...cdz_0608_02 cma alloc ...)
bitmap_find_next_zero_area_off: size=286720, start=286209, nr=456, align_mask=0xff, align_offset=0, sindex=286465, index=286720, end=287176
cma     : cma_alloc: alloc failed, req-size: 456 pages, ret: -12
cma     : cma_alloc: mask=255, offset=0, bitmap_maxno=286720, bitmap_count=456
cma     : number of available pages: 1@10415+2@10422 ...
          : +168@51544 ... +255@58369+511@58881+255@59649+ ...
          : +255@94721 ... +255@99841+255@100353+255@10086 ...
          : +255@133377 ... +255@138241+255@138753+255@13926 ...
cma     : cma_alloc(): returned          (null)
```

Reason #3: Alignment Cont'd

- As we can find from the log, the request is 456 pages with alignment of 8 bits. Most of the free pages in the free block list are 255 sized, which can't satisfy the requirement. But we can still find some 511 page sized block, such as "511@58881".
- The alloc still fails because:

511@58881 (511 pages at offset 58881, that means 58881 ~ **59396**)

⬇

58881 -> 0xFF mask (256 align) -> **59136 (aligned start offset)**

⬇

59136 + 456 (request) = **59592 > 59396 (range exceed the tail of this free area!)**

Basically, alignment requirement will shift the start address/offset if the free block is not aligned with the requested alignment. This eventually cause a "smaller" free block.

Debug method:

- Enable the CMA debug configure:

  CONFIG_CMA_DEBUG=y

  CONFIG_CMA_DEBUGFS=y
- After this, there will be detail print for "available pages" which is generated from the CMA bitmap.
- There's a sysfs interface to check the bitmap directly or do manual alloc/free test, under this path:

  <debugfs>/cma/cma-0

SECURE CONNECTIONS
FOR A SMARTER WORLD