Discussing page_pool development directions

Jesper Dangaard Brouer

Netconf
Paris, Sep 2023

History

Page Pool (PP) started out as: Memory optimization for XDP

- PP alloc cache for XDP_DROP use-case
- And recycle XDP redirected frames (see `xdp_return_frame`)
- PP pages could not be recycled via netstack (possible today)
- Optimized for 1-page per packet (keeping page refcnt==1)

History

- Developed and proposed at MM-summit 2016
- Merged with mlx5 NIC driver user in 2018

Today

Today: Page Pool have full netstack recycling support

- Uses fields in struct page for pool return path
- Default all SKBs using PP does recycling (`skb_mark_for_recycle()`)
- Recently removed API for non-recycle option (`page_pool_release_page`)

Recent PP developments

Include files restructured

Notice: page_pool include files moved into

- New directory: include/net/page_pool/
- Split into helpers.h and types.h

Work done by: Yunsheng Lin and Alexander “Olek” Lobakin

- Motivated by guidance from Jakub Kicinski

Fragmenting pages - pp_frag_count

Permit a driver to perform fragmenting of the page from within the driver

- Use-case split up by the driver after DMA
- Uses field pp_frag_count in struct page

```
struct page *page = page_pool_alloc_pages();
page_pool_fragment_page(page, DRIVER_PAGECNT_BIAS_MAX);
rx_buf->page = page;
rx_buf->pagecnt_bias = DRIVER_PAGECNT_BIAS_MAX;
/* process a received buffer */
rx_buf->pagecnt_bias--;
/* fully consumed then flush the remaining */
if (page_pool_defrag_page(page, rx_buf->pagecnt_bias))
  continue;
page_pool_put_defragged_page(pool, page, -1, is_napi);
```

Work done by: Yunsheng Lin and Alexander Duyck

Recent proposed changes upstream

Recent proposed: API to hide pp_frag_count

Extending PP with API to hide pp_frag_count handling

- [PATCH net-next v8 0/6] introduce page_pool_alloc() related API
  - By Yunsheng Lin <linyunsheng@huawei.com> V9
- API returns memory as (void) pointer to data
  - and values size and offset via pointers
- Naming is weird as it no-longer deals with struct page

Page Pool evolving into netstack memory layer?

Concerns: PP evol into netstack memory layer

Jesper's concerns

- Specialized use-case gave PP the performance edge
  - Notice: primary lockless RX-cache that gives XDP_DROP performance
- Generalizing PP will naturally lead to reduced performance
  - It will be hard to keep fast as use-cases are added
    - … dead by a 1000 paper cuts

Why not create more memory allocator types?

Alternative to page_pool extending APIs all the time

- Create more allocators types
  - Each specialized to gain performance in their use-case
- See how AF_XDP/xsk have it's own ZC allocator type
  - MEM_TYPE_XSK_BUFF_POOL

XDP layer xdp_return_frame()

- Already handles multiple memory types

Memory providers

Memory providers (by Jakub Kicinski)

- Making it possible to replace “backend” e.g. page-allocator
- e.g. allocate huge-page and split-up
  - to reduce IOTLB misses when using DMA IOMMU

Jakub’s design does fit into Page Pool

- But it can also be used by other allocators types

devmem – Device specific memory

Google (Mina Almasry) devmem proposal (RFC V2)

- Device specific memory for TCP flows
  - memory that CPU cannot read, likely belonging to GPU
- Leveraging Memory providers
- BUT also rather invasive changes to Page Pool APIs
  - Mostly because it deals with memory pointers and not pages

Jesper thinks: Should be new devmem memory allocator type

Open discussion

Open Discussion

End and Thanks

Thanks to recent Page Pool contributors

- Huawei: Yunsheng Lin + Jie Wang
- Meta: Jakub Kicinski + Alexander Duyck
- Intel: Alexander Lobakin
- Red Hat: Lorenzo Bianconi
- Fastly: Joe Damato
- Linaro: Ilias Apalodimas