OVS-DPDK vHost async data path using DMA-dev

(session3 - 26/04/22)

Agenda

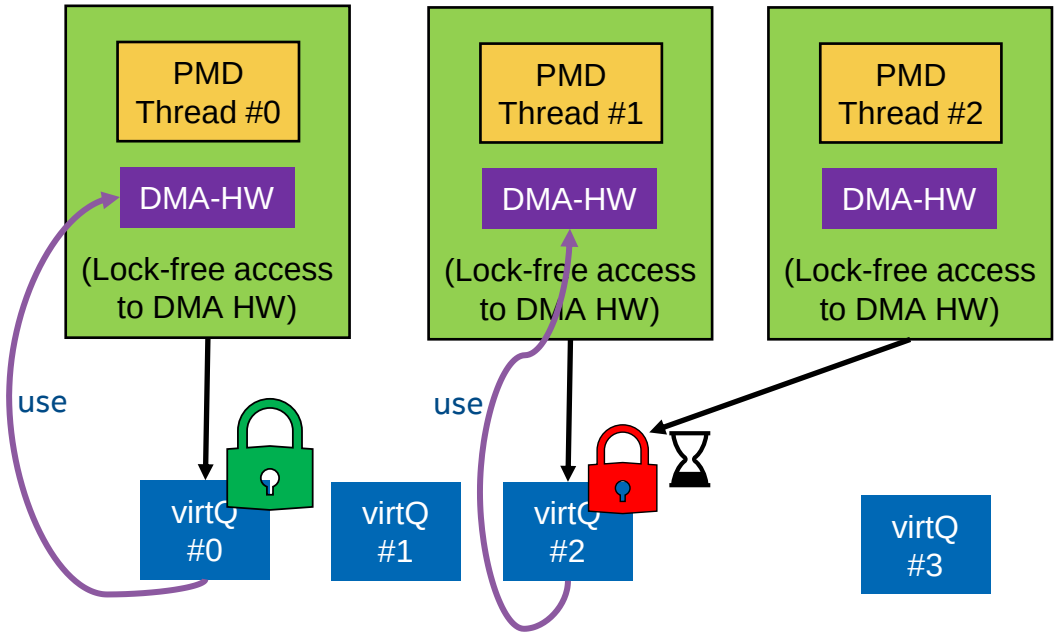
- On previous calls we discussed
  - DMA implementation — "How/where DMA completions are handled" : Defer Work, V3, V4/Lockless?
  - DMA Allocation/Mapping — "Which DMA performs the packet copy" : Static/Dynamic? "map to thread" or "map to vring"?
- In mailing list discussions, a 4th implementation was added
  - "DMA VirtQ Completions"
  - Quick review of design and requirements on next slide
- Discussion "dma-virtq-completions" concept vs others (e.g. Defer Work)
  - Some pros/cons graphics to help discussion

“DMA-VirtQ-Completions” Approach

- Core concept
  - DMA engine sets “packet ready” in virtq : reduces northbound copy latency
    - Does this avoid handling completions in the “application layer” (OVS) totally? (Answer: No, unfortunately not.)
- Technical details
  - DMA completions must still be checked : “RTE_DMA_CAPA_SILENT” is not widely available
    - CPU must check the DMA completions were successful; increase packet rx/tx stats on success?
    - So “completion handling” still required : TX-Completions-under-RX-API, or Defer Work?
  - VHost library “in-order” requirement for packet completions to guest
    - Requires 1:1 mapping DMA-to-virtq : this has bad scaling side-effects! (Graphic to show on next slide)

“Defer Work in OVS for DMA Usage”

1:1 mapping DMA HW to PMD thread (lock-free)
- DMAdev and vhost lib keep pkt order

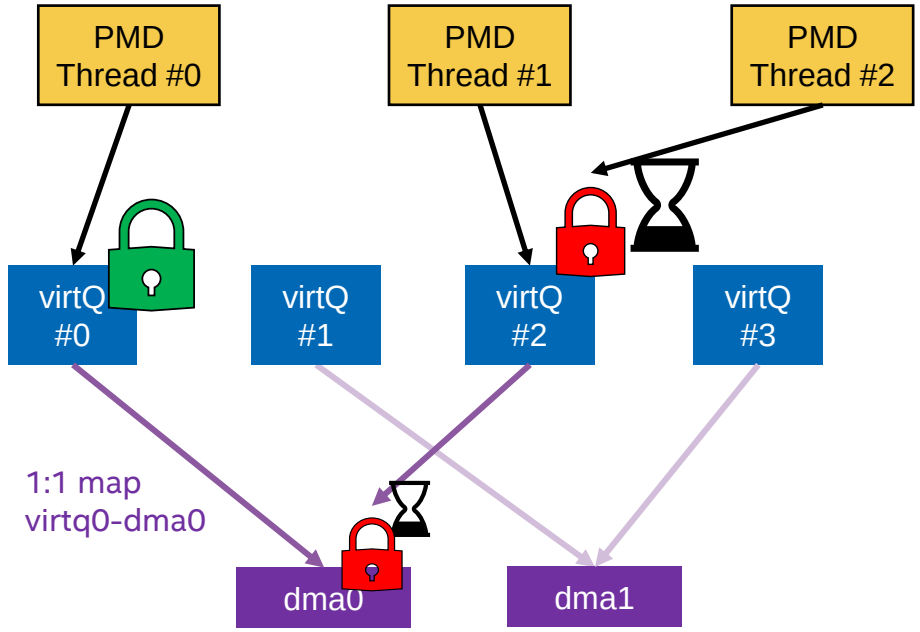


Stable Performance
- Only “true” virtq contention causes lock-waiting
- Adding PMD and DMA scales switch & copy nicely

Take 1x Lock, Make Forward Progress
- Same virtq lock as OVS has today

“DMA-VirtQ-Completions”

DMA HW to do VirtQ desc/idx updates
- Requires 1:1 mapping DMA-virtq for pkt order



Unstable Performance: Traffic Dependant Scaling
- 1:1 virtq-dma mapping causes perf problems

Take VirtQ lock, then WAIT for DMA-lock!!
- Waiting with virtq lock taken is very bad for scaling!
- Causes VirtQ “lock-backlog”
- PMD **#2** blocked until **#0** is done “false” contention

DMA-Mappings Options

- "1:1 DMA-to-PMD-thread" mapping
  - Predictable & stable performance, lock-free access to DMA devices from PMD-thread

- "1:1 DMA-to-VirtQ"
  - Does *NOT* give stable/consistent performance, lock contention *depending-on input traffic*

- "Dynamic (runtime) mapping of DMA devices"
  - Extra overhead in datapath: packet copy requires *dynamic_get()* of *dma_id* to use
  - Unstable performance at runtime from lock contention on DMA HW access?
    - 2x contention due to enq and completion check?
    - Complexity due to thread must handle completions for packets it didn't enq?

Status & Next Steps

- Discussion output?
  - Is consistent performance of OVS desired? yes?
  - Is a clean separation Rx and Tx desired? yes?
  - DMA-HW mapping: is 1:1 DMA to PMD-thread the best candidate? yes?

- Options & Next Steps
  - **Conclude discussion** around exact implementation to develop from here
    - What milestones are required here, how can we make steady progress?
  - **Push code to Github** for ease of use, testing & development
    - On branches as previously discussed on calls & OVS mailing list

(From 2nd call) Dynamic Alloc : Pro/Con Overview

	PMD-thread	VRing
Static Assignment	✓ **Simple access** to DMA HW Direct static mapping: thread to DMA id. ✓ **Lockfree** access to device Direct static mapping: thread to DMA id. ✓ **Scaling** as 1:1 mapping has no contention Adding PMD threads adds more DMA capability too! ✓ **Simple Config:** DMA queue available for thread? Yes; claim it and use it. No; Use CPU memcpy for that PMD thread (like today).	✓ **Simple** access to DMA HW Direct static mapping from VRing to DMA id. ✗ **Locking Required** Multiple vrings map to same DMA id, thread contention. ✗ **Scaling suboptimal** Contention based on traffic, vrings = DMA id, oversubscription! ✓ **Simple Config:** DMA queue available for thread? Yes? Use it. No? CPU copy for this vring.
Dynamic Assignment	? **Simple/Complex** access to device, is it shared? Sharing requires runtime check for DMA id to use? ✗ **Locking required** Sharing == multi-threaded accesses. Drain DMA-completions from 2+ threads? ✗ **Bad Scaling** Active PMD threads map to same DMA id, contention. ✗ **Complex Config** Adding PMD requires other PMD threads to change access from "simple" to "complex" at runtime?	? **Simple/Complex** access to device, is it shared? Sharing requires runtime check for DMA id to use? ✗ **Locking required** Sharing == multi-threaded accesses. Drain completions from 2+ DMA-engines for single virtq? ✗ **Bad Scaling** Active VRings map to same DMA id, contention. ✗ **Complex Config** Adding VRings requires other VRings threads to change access from "simple" to "complex" at runtime?

Agree to merge ***static*** **PMD-thread** mapping now?


intel