Bottleneck Analysis of DPDK-based Applications

Adel Belkhiri  Michel Dagenais

June 4, 2021

Polytechnique Montréal
DORSAL Laboratory
Introduction

Investigation and use cases

- How to pinpoint a performance bottleneck?
- Use cases

Conclusion
DPDK - Data Plane Development Kit

- Set of libraries and polling-mode drivers which can be leveraged to implement userspace dataplanes

- Many optimizations to accelerate packet processing (CPU affinity, huge pages, lock-less queues, batch processing, etc.)

Source: https://telcocloudbridge.com/wp-content/uploads/2019/05/image-1.png
Why DPDK-based apps might be bottlenecked? (1)

- A **network bottleneck** is a computing or networking resource that may limit the data flow in the network under some circumstances - obvious or unseen.

- **Bottleneck analysis** is a type analysis that aims at identifying which part of the system is causing the congestion.
Why DPDK-based apps might be bottlenecked? (2)

**Reasons:**

1) Mis-allocation of resources (Example: Traffic consumer threads are slower producer threads)
Why DPDK-based apps might be bottlenecked? (3)

2) Contention for shared resources (Example: Contention for accessing LLC)

3) Buggy design or implementation (Example: usage of an inadequate scheduling mechanism – Elastic Flow Distributor library vs Eventdev library)
Performance Analysis Framework for DPDK-based Applications

1) **Data collection** : static instrumentation (lttng-ust, rte_trace library ?)

2) **Bottleneck Analyses** (Trace Compass)
   - Flow classification libraries (Hash, ACL, LPM, etc.)
   - Vhost-user library
   - Pipeline library
   - Eventdev library (SW and DSW schedulers)
   - ...

Bottleneck Analysis (1)
Bottleneck Analysis (2)

Subset of computed performance metrics:

- Per-flow and per-NIC Packet rate, Enqueue/Dequeue rate, drop rate
- Occupancy of application buffers (NIC RX/TX queue, Software Queues, etc.)
- Latency of Software Queues
- Effective RX spins metric:

\[
\% \text{ Effective RX Spins} = \frac{\text{NB successful calls to } X_{\text{dequeue\_burst}}()} \times 100}{\text{Total number of calls}}
\]
Use Case 1: DPDK Packet Framework

• Bottleneck analysis of the *Internet Protocol (IP) pipeline* application
  ○ Super-pipeline composed of three pipelines, each executed by a thread mapped to a single CPU core.
    ▶ **Pipeline_A** : Receiving and filtering packets
    ▶ **Pipeline_B** : Encrypting packets belonging to specific flows before forwarding them to the next stage.
    ▶ **Pipeline_C** : Transmitting packets to the external network
  ○ The three pipelines are interconnected via two software queues: SWQ0 and SWQ1

**Problem !!** The rate of outbound traffic is lower than that of inbound traffic
Use Case 1: DPDK Packet Framework

Fig. 1: Diagram generated by our tool illustrating the super-pipeline architecture.

Tracepoint:
librte_pipeline:rte_pipeline_create

Tracepoints:
librte_port_sink:rte_port_sink_create
librte_pipeline:rte_pipeline_port_out_create

https://github.com/mermaid-js/mermaid
Use Case 1: DPDK Packet Framework

**Fig. 1**: Super-pipeline transmission rate is below reception rate

**Fig. 2**: Generated traffic did not overflow the RX/TX buffers of vhost-user NICs
Use Case 1: DPDK Packet Framework

Fig. 1: High latency of the software queue SWQ0

Fig. 2: So often, SWQ0 reaches its full capacity and causes packets to be dropped
Use Case 2: EventDev Library

- Bottleneck analysis of the *Eventdev pipeline sample application*
  - Application: 1 RX thread, 1 TX thread, and 4 worker threads
  - Pipeline with 2 stages: 2 atomic queues

*Fig. 1*: Architecture of the event device used in our application

*Problem!!* The rate of outbound traffic is lower than that of inbound traffic
Use Case 2: EventDev Library

Fig. 1: The RX buffer of the first vhost-user NIC is overflowed

Fig. 2: Considerable fluctuation characterizes the dequeue rate of Worker1
Use Case 2: EventDev Library

Fig. 1: The Effective RX Spins metric shows that the first stage workers were not overloaded.

Fig. 2: The same metric shows that the second stage workers were overloaded.
Use Case 2: EventDev Library

Fig. 1: *(zoomed view)* Second stage workers were dequeuing packets in turn and not in parallel !!

Fig. 2: The six flows processed in the first stage were merged in a single elephant flow in the second stage
• Tracing is an efficient technique to monitor the performance of DPDK-based applications and pinpoint their bottlenecks

• Data collection for less than 2% overhead
Questions?
adel.belkhiri@polymtl.ca