Performance analysis of DPDK-based applications

Adel Belkhiri        Michel Dagenais

January 8, 2021

Polytechnique Montréal
DORSAL Laboratory
Agenda

Introduction

Investigation and use cases

- Classification libraries (lpm, hash, acl, array, etc.)
- Pipeline library
- Eventdev library

Conclusion and future work
Linux Kernel bypass

- Linux kernel network stack is too slow
  - Overhead of traditional system calls (read, write, etc.)
  - Interrupts and NAPI (New API)
  - Huge skb_buff data structure
  - Packets are processed individually

- Few solutions exist ...
  - Kernel bypass tools (DPDK, PF_RING, Netmap, etc.)
  - XDP (eXpress Data Path)
DPDK - Data Plane Development Kit

- Set of data plane libraries and NICs polling-mode drivers for offloading packet processing from the kernel to userspace processes
- Many optimizations to accelerate packet processing (huge pages, lock-less queues, batch processing, etc.)

What is the problem?

- Kernel-bypassing techniques prevent the applications from using traditional management and monitoring tools.

How do we intend to solve it?

- Propose a tracing-based debugging tool for DPDK applications
  - Static instrumentation of DPDK libraries
  - Development of trace analyses capable of uncovering packet processing bottlenecks and latencies.
Work Environment

• **Software Setup:**
  - DPDK (version 19.05)
  - LTTng (version 2.10)
  - Trace Compass framework

• **Experimental Setup (for use cases)**
  - Hardware: 8 CPU cores + 32 GB memory
  - Ubuntu 18.04 (Kernel version 4.14.0)
  - Trex Traffic Generator (version 2.81)
  - Stress-ng (version 0.09.25)
Classification tables (1)

- DPDK provides many classification libraries, such as:
  - LPM (Longest Prefix Match)
  - ACL
  - HASH
  - ARRAY
  - Classifier
Classification tables (2)

Metrics:
* Per-flow lookup rate
* Table hit/miss percentage
Pipeline Library

- The DPDK Packet Framework: a set of DPDK libraries (*librte_port*, *librte_table*, and *librte_pipeline*) defining a standard methodology for building complex packet processing applications

- Each pipeline module is constructed of three primitive entities: input ports, output ports, and tables
**Use Case 1: Pipeline Library**

- **Analyzed Application**: `./dpdk-ip-pipeline`

  - 1 main thread + 4 threads (1 thread RX, 2 threads for the super-pipeline, 1 thread TX)
Use Case 1: Pipeline Library

First execution (without stress)  Second execution (with stress)

Figure: View to show the Inter-pipeline packet rate
Use Case 1: Pipeline Library

Super-pipeline Architecture

- Pipeline 0
  - net_vhost0/RX0: 2000053 / 2000053
  - net_vhost1/RX0: 47 / 47
- Pipeline 1
  - net_vhost0/RX0: 2000100 / 2000100
  - net_vhost1/RX0: 1990328 / 1990328

Graph showing network traffic over time.
EventDev Library (1)

- EventDev framework provides applications with automatic multi-core scaling, dynamic load balancing, pipelining, synchronization services, etc. via the usage of the event driven programming model.

- Event Device:
  - **Ports**
  - **Queues** (Atomic, Ordered, and Parallel)
  - **Events** (packets, time events, crypto)

Source: https://doc.dpdk.org/guides/prog_guide/eventdev.html
EventDev Library (2)

- Data flow within an event device
Use Case 2: EventDev Library

- **Analyzed Application**: `/dpdk-test-eventdev`
  - Port 2: producer
  - Port 0, Port 1: Worker threads
  - Pipeline composed of two stages (queues)

```
Analyzed Application: ./dpdk-test-eventdev

-- Port 2: producer
-- Port 0, Port 1: Worker threads
-- Pipeline composed of two stages (queues)
```
Use Case 2: EventDev Library

- **Experiment #1**: Ordered queue + Atomic queue

**Figure**: Occupancy of consumer ring buffer (size = 16)

High throughput decrease due to the stressing of worker0
Use Case 2: EventDev Library

- **Experiment #2**: Parallel queue + Parallel queue

  **Execution #1**

  **Execution #2**

  Low throughput decrease due to the stressing of worker0
Use Case 2: EventDev Library

Figure: Per-stage Port Load View

Figure: Port Rings Occupancy View
Conclusion

• Many successful projects depend on DPDK such as OVS-DPDK, FD.io VPP, and TREX.

=> Need for efficient debugging and monitoring tools

Future Work

• Leverage our analyses to study the performance of “real” DPDK applications

• Estimate and analyze the overhead of tracing
Questions?
adel.belkhiri@polymtl.ca