Performance analysis of DPDK-based applications

Adel Belkhiri   Michel Dagenais

May 15, 2020

Polytechnique Montréal
Laboratoire DORSAL

Agenda

Introduction

- Linux Kernel bypassing
- What is DPDK ?
- Motivations and goals

Investigations and preliminary results

Use cases

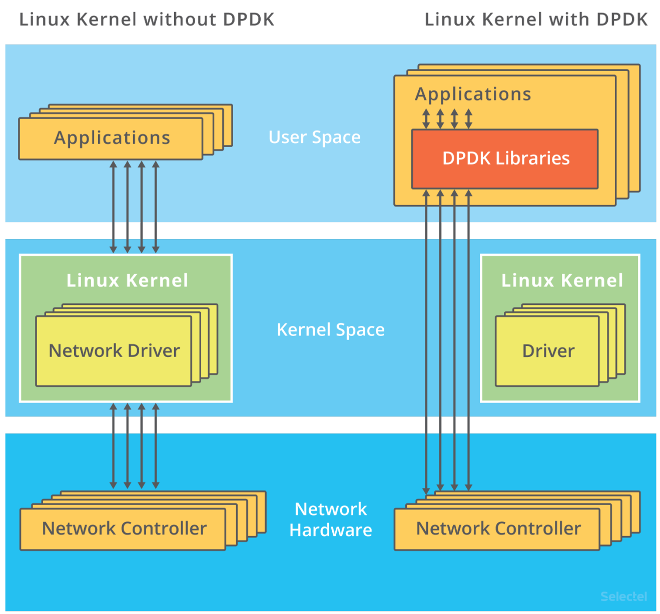
Conclusion and future work

Linux Kernel bypass

- NICs are getting faster and faster : 10Gbps, 100Gbps, etc.
- Linux kernel network stack prevents packets from being processed quickly.
  - Costly context switches and system calls (read, write, etc.)
  - Huge *skb_buff* data structure
  - Interrupts and NAPI (New API)
  - Lack of batching
- Several network stack bypass solutions : PF_RING/DNA, DPDK, PacketShader, OpenOnload, RDMA/IBverbs etc.

What is DPDK ?

- Intel DPDK (Data Plane Development Kit).
- Open source networking framework written in C, supporting a wide range of NICs and processors.
- Higher levels of packet processing throughput via Kernel bypassing.
  - ✔ Processor affinity
  - ✔ Huge pages
  - ✔ Lockless ring buffers
  - ✔ Poll Mode Driver
  - ✔ Batch processing of packets (*burst*)



Source : *https://blog.selectel.com /introduction-dpdk-architecture-principles/*

Motivation and Goals

Motivation

- Incapacity of existing tools to monitor NICs that are managed by DPDK-based applications.

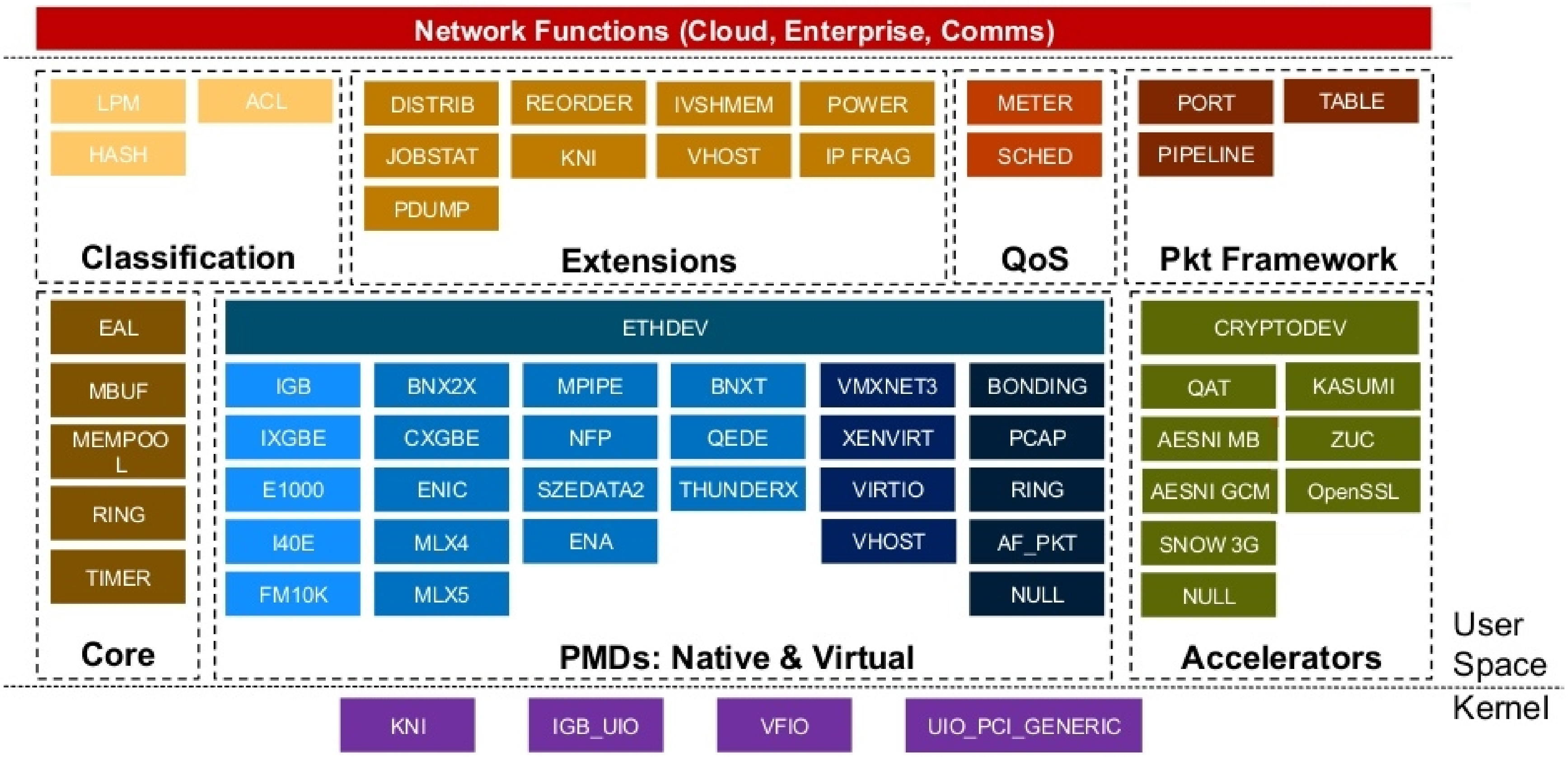
Goals

- Leverage tracing techniques to analyze the performance of DPDK-based applications.
- Shed light on the potential causes of packet processing latencies.
- Analyze the cost of tracing and its impact on frame processing performance

Work Environment

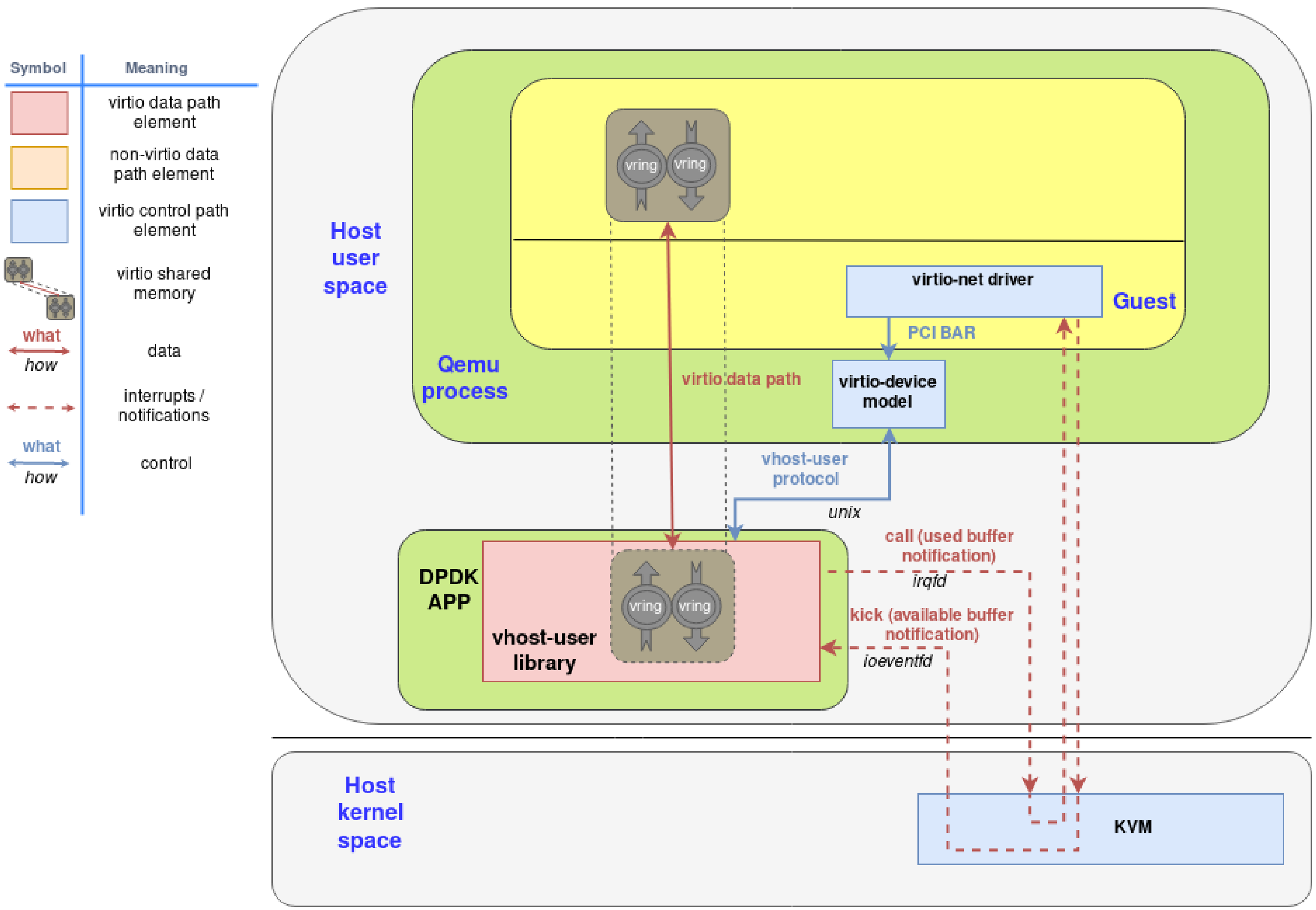
- **Software :**
  - DPDK (version 19.05)
- **Data Collection :**
  - LTTng (version 2.10)
  - Userspace tracing / static instrumentation
- **Performance Analyses :**
  - Trace Compass framework

DPDK Architecture



Source : *www.dpdk.org*

Vhost-user library (1)



Source :
https://www.redhat.com/en/blog/journey-vhost-users-realm

Vhost-user library (2)

- How to identify which entity (Host or Guest) is responsible for a TX/RX performance degradation ?
- How to measure the rate of enqueuing/dequeuing Mbuff to/from each queue ?

Use Case

- **Experiment setup :**
  - Run dpdk-testpmd in the host.

```
$ sudo dpdk-testpmd -l 0,1  --socket-mem=1000 -n 1   \
       -- vdev="net_vhost0,iface=/tmp/vhost-user1"    \
       --vdev="net_vhost1,iface=/tmp/vhost-user2"  --  ...
```

  - Configure the guest to connect to the created virtual devices.

```
<interface type='vhostuser'>
  <mac address='56:48:4f:53:54:01'/>
  <source type='unix' path='/tmp/vhost-user1' mode='client'/>
  <model type='virtio'/>
  <driver name='vhost' rx_queue_size='256' tx_queue_size='256'/>
  ...
</interface>
```

Use Case

Normal execution

Slowing down the guest *(eat-cpu)*

Figure : Host is sending packets to the guest.
(UP) Rate of MBuff enqueuing. ***(DOWN)*** Percentage of TX queue occupancy.

Use Case

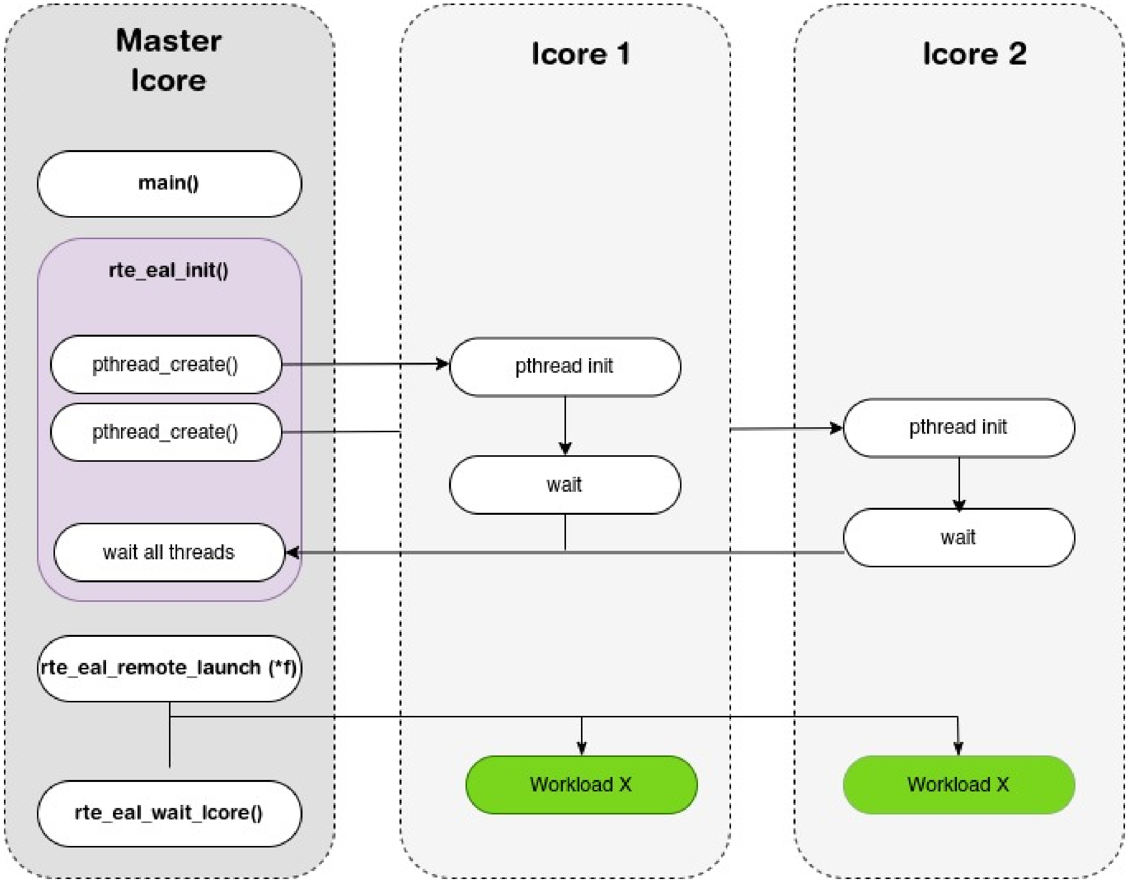
Figure : Guest is sending packets to the host.
(UP) Rate of MBuff dequeuing. ***(DOWN)*** Percentage of RX queue occupancy.

Use Case

Zoom into the previous figure

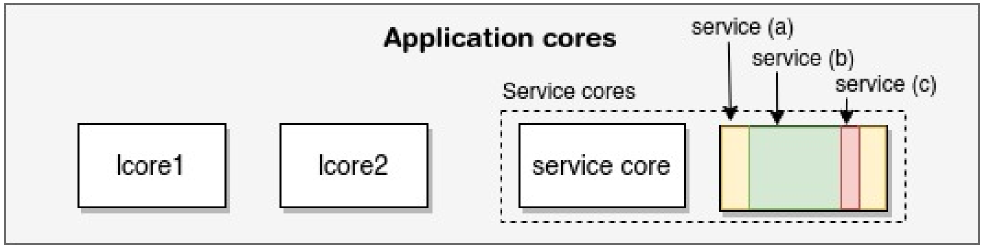
Logical Cores

- The term “lcore” refers to an EAL pthread pinned to a CPU core. “EAL pthreads” are created by EAL to execute the tasks issued via *remote_launch* functions.



Service Cores

- DPDK has support for a new dynamic way of executing workloads on DPDK lcores.
- **Service**
  - Runnable work item
  - Runs an iteration of work then returns
- **Service Core**
  - Dedicated core to running services. These services are scheduled in a simple round-robin run-to-completion.



▶ If there are many services running on a core this could potentially lead to high waiting times for some of the services.

Use Case

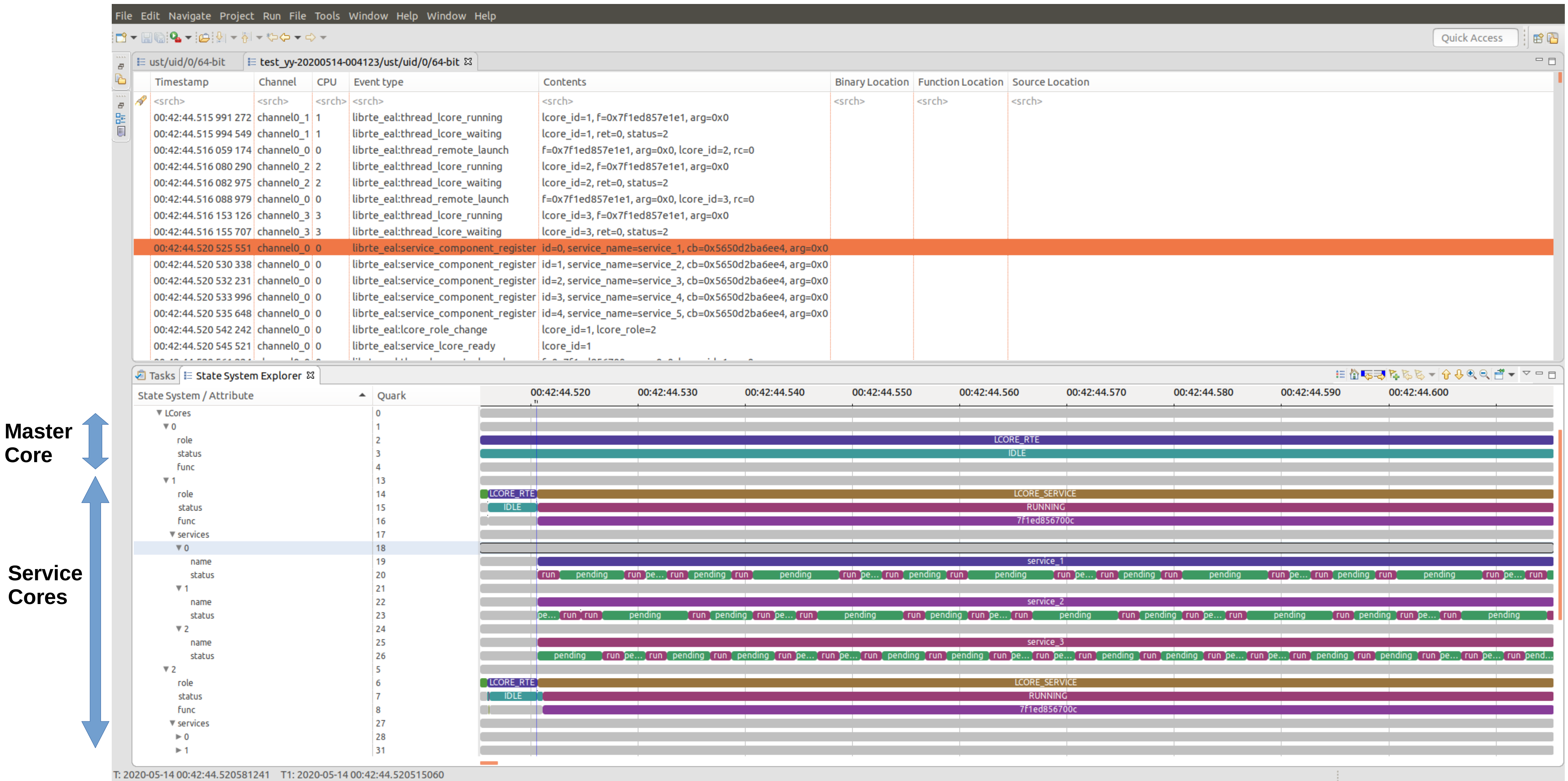
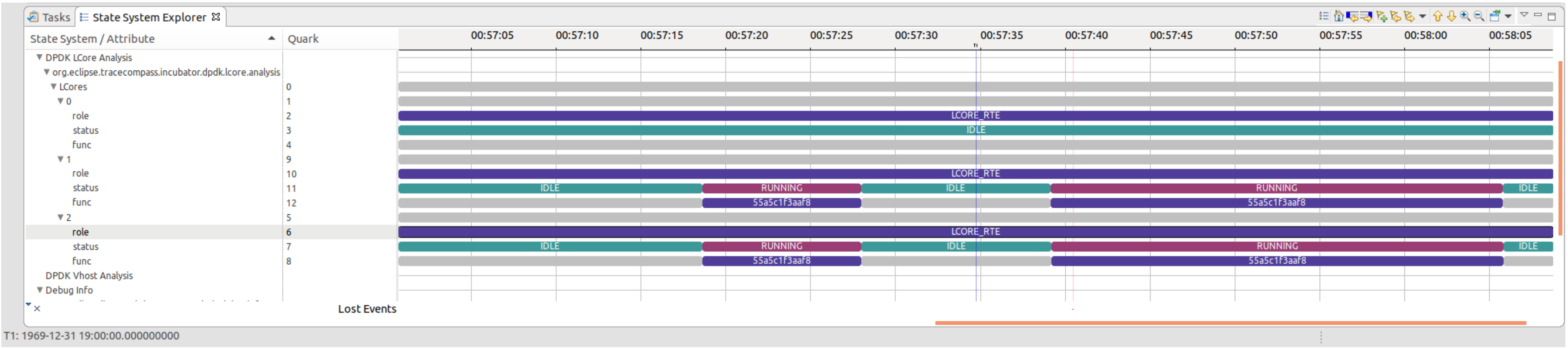


Figure : Execution of "dpdk-service_cores" sample application and illustration of the distribution of service executions across "service cores".

Use Case

- Execution of "dpdk-testpmd" with a master core and two lcores.



Conclusion

- DPDK is one of the most important open-source Linux projects [1] and many successful projects depend on it : OVS-DPDK, FD.io VPP, PfSense, TREX, etc.

- Tracing is an efficient technique to extract low-level performance data and solve many performance issues : multi-core synchronization issues, latency measurements, etc.

- A Native DPDK CTF trace support has been added to release 20.05 [2].
  - No dependency on any third-party library.
  - Ability to trace on Windows platforms.

Future Work

- Continue the instrumentation of the most popular DPDK libraries (eventDev, LPM, ACL, …)
- Refine the instrumentation in the DPDK packet processing datapath to identify possible improvements.
- Develop more comprehensive analyses.

Questions?

adel.belkhiri@polymtl.ca

Reference :

[1] https://www.linuxfoundation.org/projects/

[2] https://doc.dpdk.org/guides/prog_guide/trace_lib.html

[3] https://blog.selectel.com/introduction-dpdk-architecture-principles/

[4] www.dpdk.org

[5] https://www.redhat.com/en/blog/journey-vhost-users-realm