Identifying and reducing virtualization overhead

by François Philippe Ossim Belias

Polytechnique Montréal

DORSAL Laboratory

Research objectives

- Provide practitioners with analyses, tools, or approaches to identify the causes of the “additional overhead” introduced by virtualization (VM overhead).
- Enhance existing performance analysis tools, such as Trace Compass, LTTng, or related tools.

Methodology overview

1. **Find a workload that generate overhead**
2. **Trace that workload**
3. Identify overhead scenarios
4. Find key tracepoints
5. Develop algorithms to detect and measure overhead

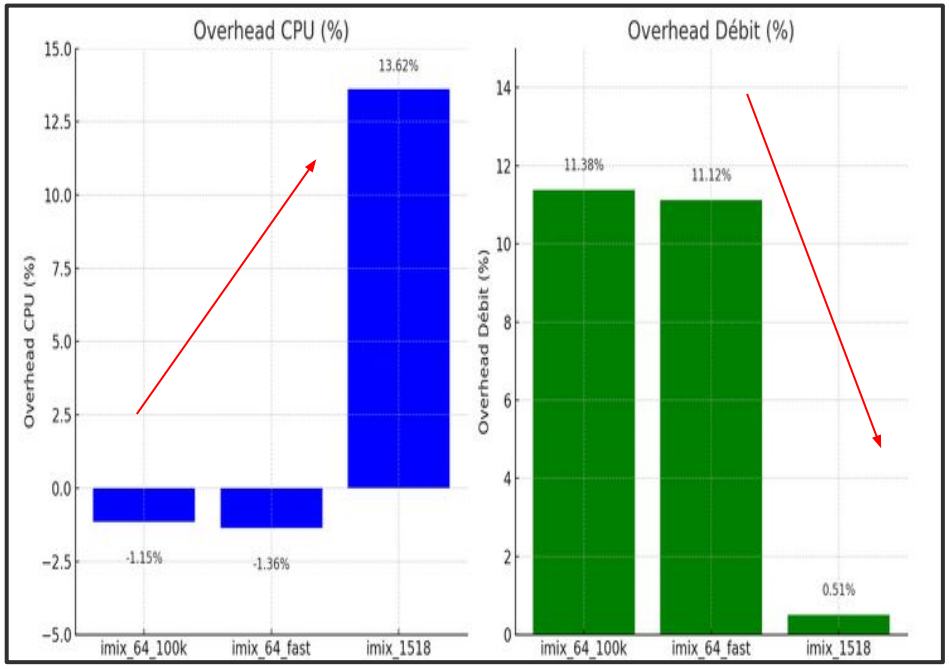
Step 1: Find a workload that generate overhead

- **Tools used**
  - TREX for traffic generation
  - DPDK test-pmd to process the packets and write informations about the packets on the disk
- For each workload we measure the throughput and cpu utilization
- **Tests configurations**
  - Without CPU pinning
  - With CPU pinning

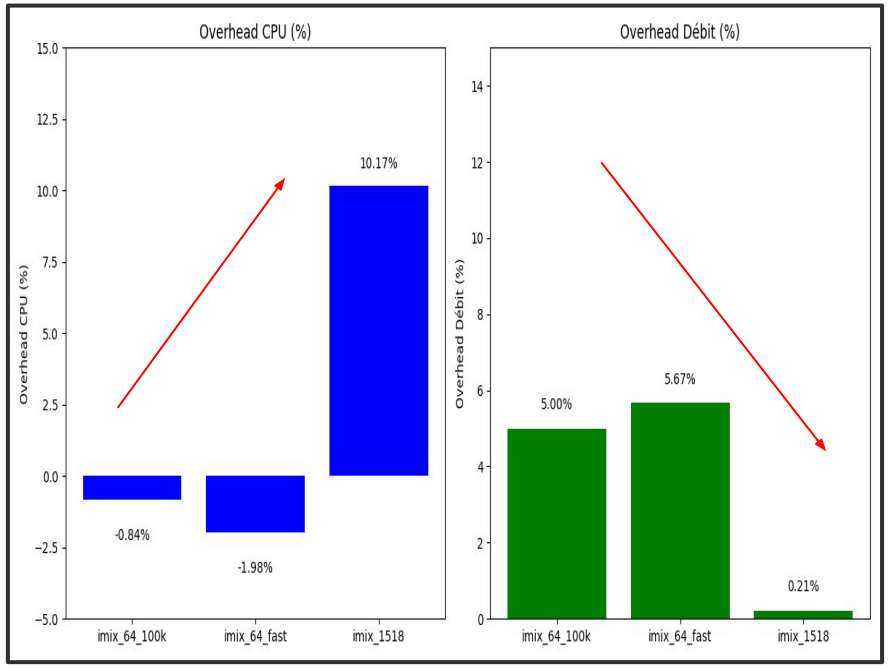
Results of the benchmarks

- Throughput overhead decreases with packet size
- CPU overhead increases with packet size
- CPU pinning increase the performance but the tendency is the same

Results of the benchmarks

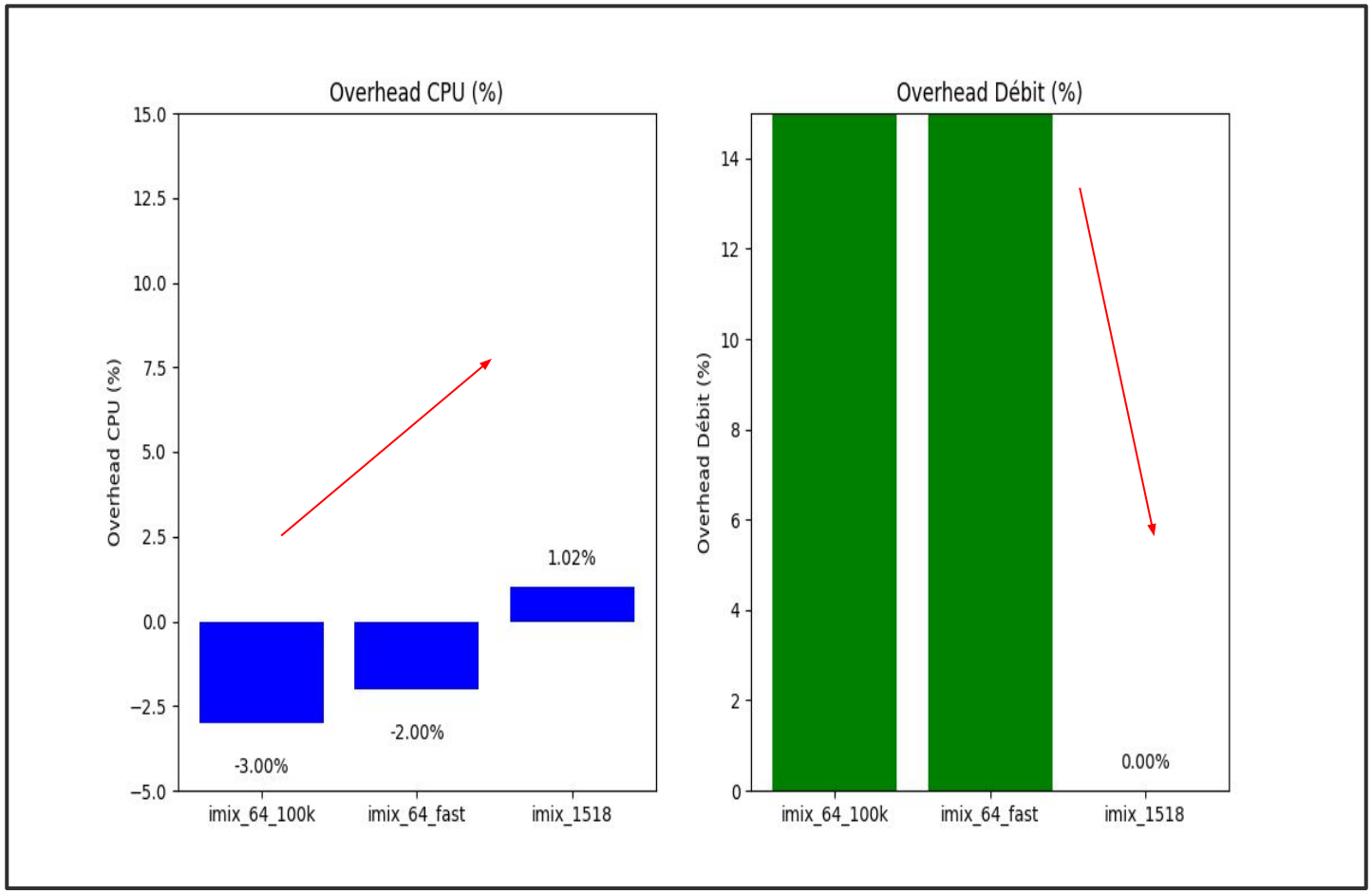


CPU and throughput overhead without cpu pinning



CPU and throughput overhead with cpu pinning

Results of the benchmarks



Throughput and CPU overhead for writing in *w mode* for UDP traffic of varying sizes

Step 2: Trace the workload

- Tools used:
  - Lttng
  - Trace compass
- Trace from the host and trace from the guest and synchronize the two traces

Step 3&4: Identify potential overhead scenarios and find key tracepoints

Scenario	Description	Potential source of overhead	Tracepoints that can help analysis
VM → Host → VM	Direct VM-to-host switch	KVM exit context (exit_reason)	`kvm_exit`, `kvm_entry`
VM → QEMU → Host → VM	Transition through QEMU for emulation	QEMU latency, I/O processing	`kvm_exit`, `kvm_userspace_exit`, `kvm_entry`, `kvm_emulate_insn`
VM → Interrupt → Host → VM	Hardware or software interrupt	Frequent interrupt handling	`irq_handler_entry`, `kvm_exit`, `irq_handler_exit`

Step 5: Develop algorithms to detect and measure overhead

- **Identify Exit and Entry Patterns:** Measure duration between kvm_entry and kvm_exit for each vcpu *(in progress)*
- Distribution of exit reason for each vcpu *(in progress)*

Step 5: Develop algorithms to detect and measure overhead

```
Algorithm VM_Overhead_Analyzer( Input: Event Trace, Ouput: Overhead Report)
entry_time = dict()  // key = vcpu, value = timestamp of the entry
exit_time = dict()  // key = vcpu, value = timestamp of the exit

Procedure VM_OVERHEAD_ANALYZER(event)
        if event.type == kvm_entry:
                vcpu = event.cpu_id
                entry_time[vcpu] = event.timestamp

                if  exit_time[vcpu] exits:
                        idle_duration = entry_time[vcpu] - exit_time[vcpu]
                        update_total_idle_time(vcpu, idle_duration)
        else if event.type == kvm_exit:
                vcpu = event.vcpu_id
                exit_reason = event.exit_reason
                exit_time[vcpu] = event.timestamp

                run_duration = exit_time[vcpu] - entr_time[vcpu]
                update_total_run_time(vcpu, run_duration)
                Increment_exit_reason_distribution(exit_reason)

        Generate_report()
end procedure
```

Next steps

- Find more scenarios
- Implement the algorithms
- Create graphical views

Thank you

Questions ?

Thoughts ?

Comments ?