Nova Scheduler: Optimizing, Configuring and Deploying NFV VNF's on OpenStack

Ian Jolliffe
Director, Architecture
Our software has been deployed in over 2 billion devices; into environments, systems, and applications subject to the highest standards of safety, security and performance.
Deployed Throughout Wired and Wireless Networks

Secure, reliable, low-latency networking for service providers

Titanium Cloud – Private Cloud for Critical Infrastructure
Predictable Performance
Requirements

- Deterministic Throughput Latency
- Predictable CPU allocation
- Predictable Memory Access
- Large number of vCPUs
- Linux as Guest OS
Performance Issues and Solutions
Enhanced Platform Awareness

- Optimize VM placement
  - NUMA
  - Memory requirements
  - NIC support
  - Acceleration engines
  - Hyper-threading

- Configure flavours for best performance

Predictable Performance for Applications
PCI Bus, Network Contention

- **PCI bus contention**
  - Each PCI bus is connected to a single host NUMA node
  - Instances using PCI devices will be affined to the same NUMA node as the device

- **Cross-NUMA traffic should be avoided**
  - vSwitch processes should run on the same NUMA node as the physical NICs they’re using
  - PT or SRIOV will be impacted by crossing NUMA nodes.
  - Instances should run on the same NUMA node as the NIC
    - Planning to submit a blueprint to add this functionality.

- **Network bandwidth contention**
  - All instances connecting to the same host network will use the same host NIC
  - 10G or higher NICs are desirable
Host/Guest Network Bandwidth Limitation

- Emulated NICs (e1000, rtl8139, etc.) are slow
- Paravirtualized NICs are faster if the guest supports it (virtio)
- PCI-passthrough/SR-IOV is faster yet
  - PCI device “passed through” into the guest (this can be complex)
    - `pci_alias={"vendor_id": "8086", "product_id": "1528", "name": "my_PF"}` in nova.conf
    - "pci_passthrough:alias=my_PF:1" in flavor extra-spec
  - Need suitable device driver in the guest
  - Still have overhead of virtual interrupts
  - Can be tricky to configure everything initially.
- A guest that leverages DPDK is fastest
  - Poll-mode driver, works best with dedicated core(s) on host and in guest.
  - Leveraging a poll-mode driver has the guest in a tight loop – draws more power.
Memory Contention

- **Host TLB contention**
  - The TLB caches the virtual page to physical page mapping
    - Shared by all processes on a host CPU core
  - Dedicated CPUs means less contention between guests -- still have contention within guest
  - Use hugepages to minimize contention – fewer mappings, more likely to stay within the cache

- **Memory contention (defaults to 1.5x overcommit)**
  - Use hugepages – this disables overcommit completely
    - Set in flavor extra spec or image properties: “`hw:mem_page_size=<large|2048|1048576>`”
    - Requires hugepages to be reserved on host, ideally at boot time

- **Memory bandwidth contention**
  - Each guest NUMA node is mapped to a unique host NUMA node
  - Spread the VM across NUMA nodes to spread memory access
    - Impact: may limit viable hosts for the scheduler
Host CPU Contention Between Guests

- By default, nova allows 16x overcommit of CPU
  - Each host CPU can run up to 16 vCPU threads

- For better performance, reduce overcommit
  - Per-compute-node
    - Set “cpu_allocation_ratio = <value>” in nova.conf
  - Per-host-aggregate
    - Run “nova aggregate-set-metadata <aggregate> cpu_allocation_ratio=<value>”

- For best performance, ensure each host CPU is “owned” by a single guest
  - Disables overcommit, also reduces host scheduling overhead
  - Set in flavor extra-spec or image properties: “hw:cpu_policy = dedicated”
  - Can’t mix instances with dedicated/shared CPUs on one compute node
    - Use host aggregates to manage, with flavor key matching host aggregate metadata key
Physical Core Contention Between Guests

- Application performance is maximized when you have a single guest running on each host physical core
  - Different guests running on hyperthread siblings are likely to impact performance

- In most cases application performance is maximized with a single guest vCPU per host physical core
  - Set in flavor extra-spec or image properties: “hw:cpu_thread_policy = isolate”
  - This “reserves” any other siblings (if any) of the same host core and keeps them idle

- In some cases application performance benefits from running on hyperthread siblings (due to shared cache or increased pipeline efficiency, for example)
  - Set in flavor extra-spec or image properties: “hw:cpu_thread_policy = require”
  - Highly recommend actually testing to see if hyperthreading is beneficial
Benchmarking
Test Topology
Performance Comparison Baseline Setup

- The basic test involves two instances with
  - 2 vCPUs, dedicated, with the “isolate” CPU thread policy
  - The image used was based on CentOS 7

- In all cases vSwitch is running on 2 host CPUs on host NUMA node 0.
- On one VM iperf was run as "taskset -c 1 iperf -s", and on the other it was run as "taskset -c 1 iperf -c <ip_addr> -t 3000 -i 10"
Performance Comparison – Virtual Interface Model

- Changed from “e1000” to “virtio” vif_model
  - Result was 4X faster

Avoid emulated hardware if possible!
Performance Comparison – Memory Backing Page Size

- Changed page size from 4KiB to 2MiB
  - Result was 4.3X faster

- Changed page size from 2MiB to 1GiB
  - No significant performance change

2MiB page size is the sweet spot
Performance Comparison – Cross-NUMA Traffic

- Move both VMs to same host NUMA node as vswitch
  - Result was 1.9X faster

Minimize cross-NUMA traffic!
Performance Comparison – Hyperthread Contention

- vCPU0 from both guests on HT siblings of same host core, vCPU1 from both guests on HT siblings of another host core
  - Reduced performance by 37%

Only one Pipeline
OpenStack can deliver excellent performance

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Performance Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU isolation</td>
<td>16X</td>
</tr>
<tr>
<td>Huge pages 2MiB</td>
<td>4X</td>
</tr>
<tr>
<td>NUMA awareness</td>
<td>2X</td>
</tr>
<tr>
<td>Virtio interface</td>
<td>4X</td>
</tr>
</tbody>
</table>
Summary

- NFV applications have very stringent performance requirements

- There are many options for configuring OpenStack
  - Know your options and benchmarks
  - Use data to drive your decisions

- Predictable performance will speed adoption of NFV