WHITE PAPER

A Path to Line-Rate-Capable NFV Deployments with Intel® Architecture and the OpenStack* Kilo Release

Leveraging Enhanced Platform Awareness features—such as SR-IOV, NUMA, huge pages, and CPU pinning—facilitates line-rate NFV application deployment.

Adrian Hoban
Przemyslaw Czesnowicz
Sean Mooney
James Chapman
Igor Shaula
Ray Kinsella
Christian Buerger

Executive Summary

Network Functions Virtualization (NFV) provides a way to take fundamental network services (such as load balancing, firewall services, network address translation, caching, and intrusion prevention) and implement them in software on standard high-volume servers (SHVSs), rather than using expensive, complex hardware switches, special-purpose appliances, and routers. Initiated by the European Telecommunications Standards Institute (ETSI) Industry Specification Group (ISG) in 2012, the specification for NFV defines a network infrastructure model that supports virtualized scalable applications—running on a commodity hardware platform—rather than the traditional model that relies on monolithic, vertically integrated, discrete applications.

Intel® architecture-based servers offer capabilities to optimize and accelerate the deployment of virtualized NFV applications. These capabilities include:

- Intel® Virtualization Technology (Intel® VT) for IA-32
- Intel® 64 and Intel® Architecture (Intel VT-x)
- Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d)
- Intel® Virtualization Technology (Intel® VT) for Connectivity (Intel® VT-c).

Additionally, a set of configuration capabilities on Intel architecture-based servers help deliver improved performance and enhanced predictability for NFV applications.

OpenStack*, a leading open-source software suite for creating private and public clouds, can be used to fill the role of the Virtualized Infrastructure Manager in the ETSI-NFV reference architecture. To do this, a number of feature additions are required to deliver on the performance and predictability demands required for NFV applications. Embedding innate knowledge of NFV applications in OpenStack is not necessary. OpenStack extensions are required to offer sufficient tuning capabilities necessary to deploy a high-performance NFV workload.

Lab tests of a reference NFV application, the Intel® Data Plane Performance Demonstrator, showed that by leveraging Enhanced Platform Awareness features, such as Single Root I/O Virtualization (SR-IOV), Non-Uniform Memory Architecture (NUMA), huge pages, and CPU pinning, line-rate NFV application deployment is facilitated.

Table of Contents

New Features and Fixes: OpenStack Kilo

The new features of OpenStack Kilo and the value proposition of each within NFV environments are shown in Table 1.

Table 1. Added OpenStack* Kilo Features and Fixes

FEATURE NAME	DESCRIPTION	INTEL® ARCHITECTURE VALUE	KEY CHARACTERISTIC
CPU pinning	Supports the pinning of virtual machines (VMs) to physical processors.	Improves the performance of applications supporting the Data Plane Development Kit (DPDK); workloads can be assigned (pinned) to specific physical CPU cores.	Delivers higher performance through Enhanced Platform Awareness capabilities.
I/O-based, NUMA-aware scheduling (for PCIe* operations)	Creates an affinity that associates a VM with the same NUMA nodes as the PCI* device passed into the VM.	Delivers optimal performance when passing through NICs and QA devices.	Improves performance.
Code fixes to NUMA topology	Fixes an issue with the consume_ from_instance method generating an exception if an instance has NUMA topology defined.	Stabilizes operations involving NUMA topology.	Improves stability.
Code fixes to NUMA I/O scheduling	Fixes an issue that prevented the scheduling of VMs with NUMA topology and a PCI device.	Supports VM scheduling of PCI devices within NUMA topology.	Improves stability.
Support for VFIO	Through the Devstack plug-in to OpenStack provides support for VFIO. VFIO was added in DPDK 1.7; this epic makes SRT use VFIO instead of UIO.	VFIO was added in DPDK 1.7; this epic makes SRT use VFIO instead of UIO. VFIO provides a more secure user space driver environment that UIO.	Improves the security of the user space driver environment.
PCI pass-through bug fix	Fixes a bug that generated an exception if a host is without PCI devices.	Supports hosts regardless of whether PCI devices are connected.	Improves stability.

Introduction

This paper introduces Network Functions Virtualization and some of the activities related to OpenStack* that are helping to enable deployment of high-performance workloads on industry-standard, high-volume servers for use in telecommunications environments.

This is an update of a previous version of this paper that focused on the Juno release. Content updates reflect advances incorporated into the OpenStack Kilo release.

Network Functions Virtualization

Network Functions Virtualization (NFV) is a European Telecommunications Standard Institute (ETSI) Industry Specification Group (ISG).[1] A number of the world's leading operators met in 2012 to form this specifications group within ETSI. The role of this ISG is to foster collaboration between a wide spectrum of the telecommunications industry to create specifications that could be used to accelerate the research, development, and deployment of telecommunications workloads on industry-standard, high-volume servers.

As a result of this activity, three whitepapers have been published. The first paper[2] is introductory and sets the stage for work to come. The second[3] and third[4] papers offer the network operators perspectives on the industry progress against the stated NFV goals.

After just 10 months of existence, a number of draft specifications were published and have subsequently been updated and are available publicly.[5] The specification on NFV Performance & Portability Best Practices[6] is particularly relevant to the scope of this paper. A number of platform capabilities were identified that should be configured or leveraged correctly in the platform to enable high-performance networking applications.

OpenStack

OpenStack is a leading open-source software suite for creating private and public clouds.[7] The functionality provided can be, for the most part, classified under similarly intentioned names such as Cloud Operating System or Virtualized Infrastructure Manager (VIM). OpenStack is used to manage pools of compute, networking, and storage infrastructure. These infrastructure resources are typically based on industry-standard, high-volume servers and can be partitioned and provisioned for the user in an on-demand style by means of a Command Line Interface (CLI), RESTful API, or a Web interface. OpenStack was released to the open-source community in 2010 and has since grown in popularity with an active community of users and contributors. The code is released under an Apache 2.0 license.

The OpenStack compute service is called Nova*. It is responsible for managing all compute infrastructure in an OpenStack managed cloud. Multiple hypervisor drivers are supported including QEMU*/KVM* (by means of libvirt), Xen*, and VMware vSphere* Hypervisor (VMware ESXi*). Nova contains the scheduling functionality that is used to select which compute host runs a particular workload. It filters all available platforms to a suitable subset of platforms based on the input requirements, and then selects a platform from the subset based on a weighting routine.

The OpenStack Networking service is called Neutron*. Neutron is designed to be a scalable service offering many different plug-in solutions to help with network management for the provider and to offer a self-service interface to the consumer to create their own networks. Several network models are available such as a flat network, Virtual LAN (VLAN), VXLAN, and others. IP Address Management

(IPAM) support includes static IP address allocation, Dynamic Host Configuration Protocol (DHCP), and Floating IP support, the latter allowing for traffic to be dynamically rerouted in the infrastructure to different compute nodes. In addition, some advanced networking services such as Load Balancers (LBs), Firewalls and Virtual Private Networks (VPNs) can be managed.

There are a number of other services that come together to form an OpenStack release. The OpenStack Dashboard service is called Horizon*. It offers users the ability to manage their infrastructure through a web interface. The OpenStack Storage service supports both block storage and object storage. Storage support is available through the Cinder*, Swift*, and Ceph* projects. There is a set of shared services, such as the Image Service (Glance*) that is used for image import, registration, snapshotting, and so on, and the Identity Service (Keystone*) that is used as a common authentication system and for inter-service message authorization.

Broadband Network Gateway

The Broadband Network Gateway (BNG) is a component in service providers' networks that is used to route traffic between the Internet and remote access devices such as Digital Subscriber Line Access Multiplexers (DSLAM), a Cable Modem Termination System (CMTS) or Multi-Service Access Nodes (MSAN). It is sometimes referred to as a Broadband Remote Access Server (BRAS).

The BNG is an enforcement point in the service providers' network for policy-based Quality of Service (QoS) and is a logical termination point of the consumers' link access protocols such as Point-to-Point Protocol over Ethernet (PPPoE), and Point-to-Point Protocol over ATM (PPPoA). It is typically the first IP hop that the user equipment (remote access client device) sees when making a connection to the Internet. The BNG requires high-performance network connectivity combined with low packet latency and low packet delay variation in order to run effectively.

This sample BNG should be considered a protocol conversion application. The BNG reference architecture is shown in Figure 2. The BNG is based on a pipelined software architecture where different parts of the packet processing workload are separated out and allocated to different threads. The workloads in the various pipeline stages were balanced to minimize as much as possible waiting between stages. The cpe 0 and cpe 1 are the network interfaces connected to the Customer Premise Equipment (CPE) side network, and inet 0 and inet 1 are the network interfaces connected to the (Internet) core side. For the uplink path, a CPU-core LB processes the incoming traffic from cpe 0 and distributes it to one of eight Worker Threads (WT). After the chosen WT processing completes, the traffic is directed to an Aggregation Thread (A) that transmits the data to the dedicated egress interface. The downlink data path followed a similar pattern.

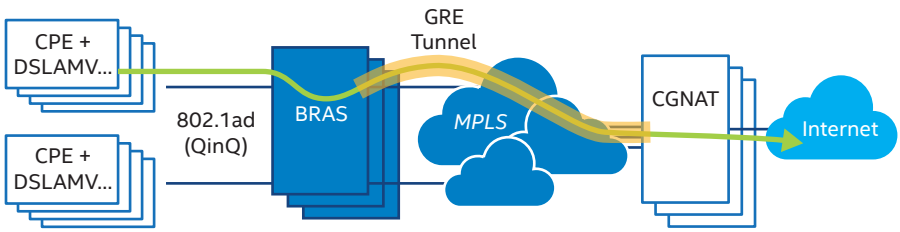

Figure 1. The Broadband Network Gateway deployment in a service provider network.

Intel® Data Plane Performance Demonstrator

A Virtualized Network Function (VNF) was needed to demonstrate how OpenStack could be used to configure high-performance network connectivity to a VNF and enable high-performance behavior of the VNF itself. The Intel® Data Plane Performance Demonstrator (Intel® DPPD)[8] is a Data Plane Development Kit (DPDK) v1.7 based application.[9] The Intel DPPD is a user space application that was designed for benchmarking purposes in order to demonstrate how high-density network traffic similar to real BNG traffic can be handled using DPDK and to study platform performance under such a workload. The Intel DPPD implements many typical BNG functionalities, such as handling properly constructed BNG packets; however, exception path processing has not been implemented. The software is released under a BSD 3-Clause License.[10] The Intel DPPD was deployed in a virtual environment based on QEMU/KVM.

Note: A detailed analysis of packet processing performance related to this sample application is available in the Network Function Virtualization: Virtualized BRAS with Linux* and Intel® Architecture white paper.[11]

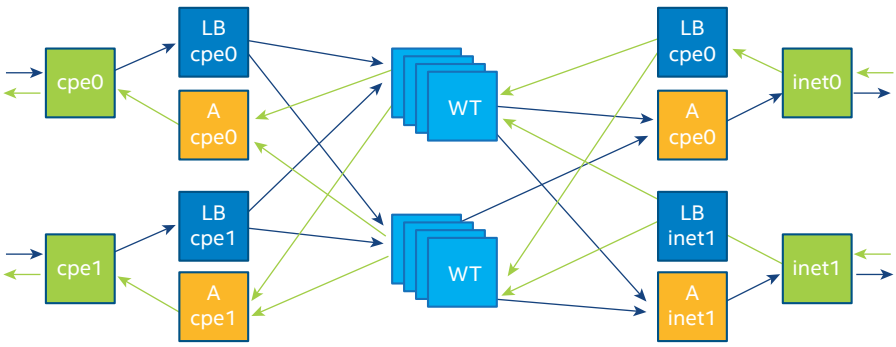

Figure 2. Intel® Data Plane Performance Demonstrators (Broadband Network Gateway) software architecture.

Document Scope

This paper focuses on Network Functions Virtualization and discusses how OpenStack extensions help deploy NFV workloads efficiently on Intel architecture-based processors. Refer to the specific processor product documentation for determining how the processor features described in this paper apply to a chosen processor.

OpenStack Extensions That Benefit NFV

From an NFV perspective, OpenStack does not need to be innately aware of the NFV applications or their functions. However, OpenStack does need the ability to provide a suitably advanced selection of tuning capabilities that enable a service provider to deploy NFV with the necessary performance and efficiency characteristics.

OpenStack features and capabilities are developing rapidly. This section describes some of the features that have a particular applicability to deploying NFV workloads effectively.

CPU Feature Request

Some NFV applications may have been developed to make specific use of certain CPU instructions. For example, a VPN appliance that requires a high-performance cryptography library could be coded to leverage specific instructions such as the Intel® Advanced Encryption Standard New Instructions (Intel® AES-NI).[12] Best practice guidelines would recommend that software developers check for the availability of instruction set extensions by means of the `cpuid` instruction before invoking code that leverages them.

In the OpenStack Icehouse release a change was made to the Nova libvirt driver to expose all of the CPU instruction set extensions to the Nova scheduler. This correlated with a related change in libvirt to make this data available by means of the libvirt API. These changes made it possible to create Nova flavors that contained specific feature requests by adding these to the flavor as extra_specs.

```
nova flavor-key <flavor-
name> set capabilities:cpu_
info:features=<feature name,
e.g. aes>
```

During scheduling, the compute_capabilities_filter in Nova compares requirements on the host CPU as specified by the flavor `extra_specs` with a database of hosts and their respective CPU features.

For CPU feature requests to work, configure libvirt to expose the host CPU features to the guest. The following setting in `/etc/nova/nova.conf` enables this.

```
[libvirt]

cpu_mode=host-model or
host-passthrough
```

PCIe* Pass-Through

The OpenStack Havana release included support for full device pass-through and SR-IOV for non-networking devices and networking devices not managed by Neutron. This included the ability to allocate Physical Functions (PFs) or Virtual Functions (VFs) to the virtual machine (VM) from PCIe cryptographic accelerators such as Intel® QuickAssist Technology. To pass through the full PF device, the PF device driver must not have VFs configured. In the case of networking devices, the networking configuration is the responsibility of the VM. It is not intended for integration with an OpenStack Neutron managed network.

The Nova configuration file, `nova.conf`, on the host needs the alias set up, the white list configured to enable the VF identifiers to be shared with the Nova scheduler, and the `PciPassthroughFilter` enabled.

```
pci_alias={"name":"niantic",
"vendor_id":"8086", "product_
id":"10fd"}

pci_passthrough_whitelist={"
address":"0000:08:00.0"}

scheduler_default_filters
= <default list>, Pci-
PassthroughFilter
```

SR-IOV Extensions

In the OpenStack Juno release, the SR-IOV capability was extended to include support for network devices. By doing so, the highest performing I/O path from physical NIC to the VM was enabled.

The steps required to use the SR-IOV extensions for NICs follow:

- On the host, the device driver for the NICs must be configured to enable the NIC VFs, and the IOMMU needs to be enabled.

```
Edit /etc/default/grub
GRUB_CMDLINE_LINUX="intel_
iommu=on"
```

- A recommended practice is to create a host aggregate for the platforms that have additional tuning for NFV and an aggregate for platforms not tuned to NFV.

```
nova aggregate-create
nfv-aggregate

nova aggregate-set-metadata
nfv-aggregate nfv=true


nova aggregate-create
default-usage

nova aggregate-set-metadata
default-usage nfv=false
```

Update all other flavors to capture the metadata for the `nfv-aggregate` as being disabled.

```
nova flavor-key <flavor-name>
set aggregate_instance_
extra_specs:nfv=false
```

- In the Nova configuration file, `nova.conf`, the white list needs to be configured to enable the VF identifiers to be shared with the Nova scheduler, and the `PciPassthroughFilter` needs to be enabled. Note: The PCI alias is not required when managing the SR-IOV networking device with Neutron.

```
pci_passthrough_whitelist={"
address":"0000:08:00.0","physi
cal_network":"physnetNFV"}

scheduler_default_
filters = <default list>,
PciPassthroughFilter
```

- Neutron requires that the SR-IOV mechanism driver be used and the VF vendor and device IDs be set up.

```
Configure /etc/neutron/
plugins/ml2/ml2_conf.ini
[ml2]
  tenant_network_types = vlan
  type_drivers = vlan
  mechanism_drivers =
  openvswitch,sriovnicswitch

[ml2_type_vlan]
  network_vlan_ranges =
  physnetNFV:50:100

Configure /etc/neutron/
plugins/ml2/ml2_conf_sriov.ini
[ml2_sriov]
  supported_pci_vendor_devs
  = 8086:10fb
  agent_required = False
[sriov_nic]
  physical_device_mappings =
  physnetNFV:eth1
```

- With the Neutron API, the tenant must create a port with a Virtual NIC type (`vnic_type`) of `macvtap` or `direct`. The latter means that the VF will be allocated directly to the VM.

```
neutron net-create —
provider:physical_
network =physnetNFV
-provider:network_type=vlan
NFV-network

neutron subnet-create
NFV-network <CIDR> —name
<Subnet_Name> —allocation-
pool=<start_ip>, end=<end_ip>

neutron port-create NFV-
network --binding:vnic-type
direct
```

- The port ID that is returned from the Neutron port-create command must be added to the Nova boot command line. Note: During the boot process, Nova will check the validity of this port ID with Neutron.

```
nova boot --flavor <flavor-
name> --image <image> --nic
port-id=<from port-create
command> <vm name>
```

NUMA Extensions

Awareness of NUMA topology in the platform was added in the OpenStack Juno release with the Virt driver guest NUMA node placement and topology extension.[13] This feature allows the tenant to specify its desired guest NUMA configuration. The Nova scheduler was extended with the numa_topology_filter to help match guest NUMA topology requests with the available NUMA topologies of the hosts.

```
scheduler_default_
filters = <default list>,
NUMATopologyFilter
```

Tenants can specify their request by means of a Nova flavor-based mechanism. An example of such a command is:

```
nova flavor-key <flavor-name>
set hw:numa_mempolicy=strict
hw:numa_cpus.0=0,1,2,3
hw:numa_cpus.1=4,5,6,7
hw:numa_mem.0=1024 hw:numa_
mem.1=1024
```

Tenants also have the option to specify their guest NUMA topology request by means of an image-property-based mechanism. An example of such a command is:

```
glance image-update image_
id —property hw_numa_
mempolicy=strict —property
hw_numa_cpus.0=0,1,2,3
—property hw_numa_
cpus.1=4,5,6,7 —property
hw_numa_mem.0=1024 —property
hw_numa_mem.1=1024
```

These commands result in OpenStack configuring the guest virtual CPUs 0, 1, 2, and 3 to be mapped to socket 0 (also known as cell 0 in libvirt terminology), and virtual CPUs 4, 5, 6, and 7 to be mapped to socket 1.

I/O-Aware NUMA Scheduling

The previous section on OpenStack NUMA configuration shows how to request the guest NUMA topology. This configuration does not take into account the locality of the I/O device providing the data to the processing cores. For example, if an application requires that vCPU cores are allocated to a particular NUMA cell, but the NIC transferring the data is local to another NUMA cell, this will result in reduced application performance.

The OpenStack Kilo release enables I/O-aware NUMA scheduling. To use this capability with PCIe pass-through, the flavor must be updated to request the particular PCIe device.

```
nova flavor-key <flavor-
name> set "pci_
passthrough:alias"="niantic:1"
```

In this example, the `alias` is the same as was configured in the PCIe Pass-Through section, and the number 1 represents the number of VFs that must be allocated.

The `PciPassthroughFilter` finds hosts with the requested PCIe devices. If the I/O device is a network device and the `vnic-type` set during network port create is `direct` or `macvtap`, then the `physical_network` setting that was used during the neutron `net-create` command is also taken into account by the scheduler to identify a suitable set of hosts. Next, the `NUMATopologyFilter` filter selects the NUMA node and once the node is selected it checks to confirm that the NIC is locally attached.

CPU Pinning

The previous section on OpenStack NUMA configuration shows how the guest NUMA topology can be requested. In the example given, the guest vCPUs 0, 1, 2, and 3 are mapped to socket 0 (also known as "NUMA Cell 0"). However, this does not mean that vCPU 0 maps to the physical CPU (pCPU) 0 or that vCPU 1 maps to pCPU 1, and so on.

The host scheduler still has the flexibility to move threads around within that NUMA cell. This impacts NFV applications such as the virtual BNG that requires that the threads are pinning to pCPUs to comply with particular application design constraints, or to deliver on particular latency or predictability metrics.

The Kilo release of OpenStack adds a CPU pinning capability, enabled by specifying the CPU policy as an extra spec. The default is a "shared" CPU policy; this is a behavior also implemented in the Juno release. A "dedicated" CPU policy enables one of the CPU thread policies to be applied.

```
hw:cpu_policy=shared|dedicated
```

Four CPU thread policies are approved for inclusion in OpenStack. Each CPU thread policy relates to the configuration of guest vCPU to host pCPU mapping in the context of a Simultaneous Multi-Threading (SMT) enabled system. An SMT-enabled, Intel® architecture-based platform has two CPU hardware threads that can execute simultaneously on each core execution unit' To the kernel, this appears to be twice as many cores in the system as are actually available.

The CPU Threads Policy "prefer" has been implemented in Kilo as the default behavior. The prefer setting places the guest vCPUs onto the host pCPU siblings.

```
hw:cpu_threads_policy=prefer
```

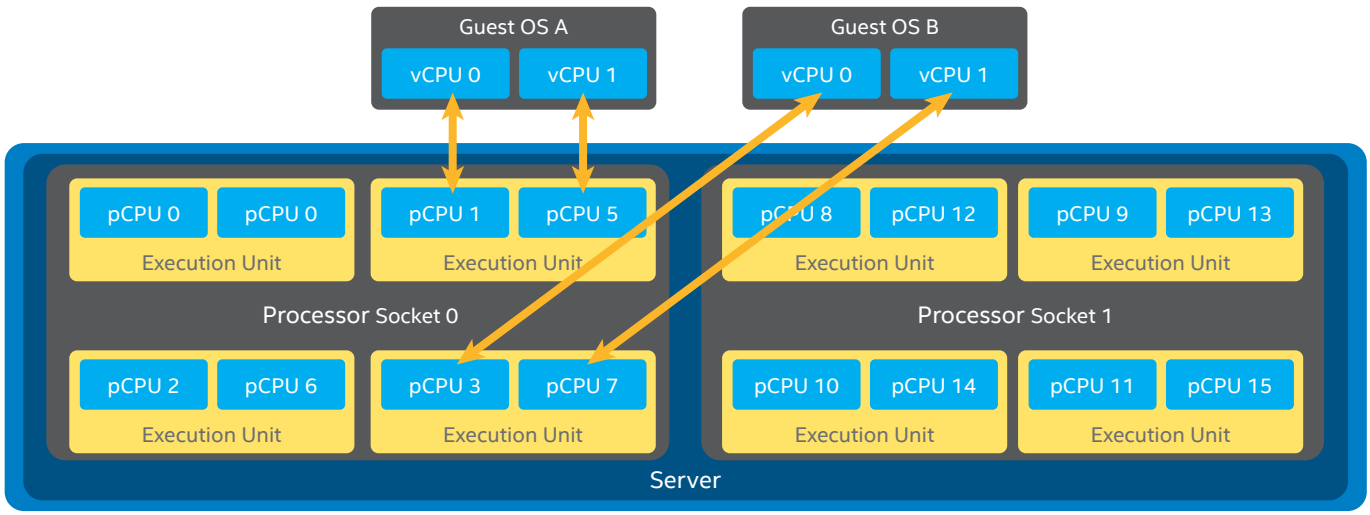

Figure 3. OpenStack* "Prefer" Virtual CPU to Physical CPU Threads Policy Mapping.

When using a dedicated CPU Policy, isolate CPUs from the host operating system (OS) on the platform to prevent the guest and the host from contending for resources on the same cores. To prevent the host from using specific cores, use the isolcpus setting on the grub command line. To avoid any contention on execution units from shared CPU siblings, such as between pCPU 0 and pCPU 4 in Figure 3, isolate sibling CPUs from the host. Based on the CPU numbering shown in Figure 3, if it is necessary to dedicate one entire execution unit from each socket to the host, then a possible isolation strategy would be to allocate pCPUs 0, 4, 8, and 12 to the host and isolate the other pCPUs.

```
Edit /etc/default/grub

GRUB_CMDLINE_
LINUX="isolcpus= 1, 2, 3, 5,
6, 7, 9, 10, 11, 13, 14, 15"
```

Three other settings for the CPU Thread Policy are defined in the approved blueprint, expected to be implemented in the Liberty release. The `"separate"` setting ensures that the vCPUs from the same guest are not placed on physical CPU siblings. The `"isolate"` setting ensures that the vCPUs from the same guest are not allocated to a pCPU that already has a sibling allocated. The `"avoid"` setting ensures guest vCPUs are not allocated to platforms that have SMT enabled.

Huge Pages

The OpenStack Kilo release supports huge page capabilities on hosts. To leverage huge pages, the host OS must be configured to define the huge page size and the number to be created. As shown in the following example, the default huge page setting is 2 MB, with eight 1 GB pages allocated.

```
Edit /etc/default/grub

GRUB_CMDLINE_LINUX="default_
hugepagesz=2MB hugepagesz=1G
hugepages=8"
```

In addition, `libvirt` on the host must be configured to allow the use of Huge Pages. In the `/etc/libvirt/qemu.conf` file, uncomment the `hugetlbfs_mount` line and make a change similar to the following statement:

```
hugetlbfs_mount = "/mnt/
huge"
```

The OpenStack Nova scheduler has been updated to track and allocate these huge pages to guest OSs. The flavor definition should contain the page size that is required for the VM. The following example shows a request for 1 GB page size.

```
nova flavor-key <flavor name>
set hw:mem_page_size=1048576
```

VLAN Trunking API Extension

The OpenStack Kilo release supports a new network property that indicates the requirement for transparent Virtual Local Area Networks (VLANs) (defined as a network configuration that allows VLAN tagged traffic to be passed into the guest). When a transparent VLAN-enabled network is requested, those ML2 drivers flagged as not supporting transparent VLANs or without the attribute will not create the transparent network.

This setting does not provide greater support for transparent networks with the Kilo release but does make it more obvious to the user whether or not transparent VLAN network requests can be fulfilled.

```
neutron net-create —
provider:physical_network
=<physical network name>
-provider:network_type=vlan
—vlan-transparent=true
<tenant network name>
```

Service VM Port Security (Disable) Extension

In the Juno release, Neutron security groups would routinely apply anti-spoofing rules on the VMs. These rules allow traffic to originate and terminate at the VM as expected, but prevent traffic from passing through the VM, as can be required by the Virtual Network Function. The Kilo release adds the ability to disable port security for use cases that require the VM to pass traffic through it.

```
neutron port-create
<network> --port_security_
enabled False
```

Sample NFV Performance Results

The Intel DPPD was instantiated on Intel® Server Board S2600GZ (code-named Grizzly Pass), BIOS version SE56600.86B.02.03.000, with two Intel® Xeon® processor E5-2690 v2 @ 3.0 GHz, with 10 cores (20 threads) each, 64 GB of DDR3-1333 RAM, one Intel® Ethernet Server Adapter X520-SR2, and two physical networks.

The OpenStack Juno release was used to instantiate the VM with the sample BRAS/BNG application on the platform. Fedora* 20 x86_64 was used for both the host OS and the guest OS on the compute nodes.

> **Note**: The demo was not updated to the Kilo environment as the isolate CPU Thread Policy is required for this test and the test code had not been updated to support Neutron-managed SR-IOV connections.

The Intel DPPD application handles network traffic generated by the Intel Packet Generator application running on a separate system. During each test the volume of the traffic generated by the Packet Generator was increased until the BNG application started to report packet loss. The maximum BNG throughput without a packet loss was considered a valid data point.

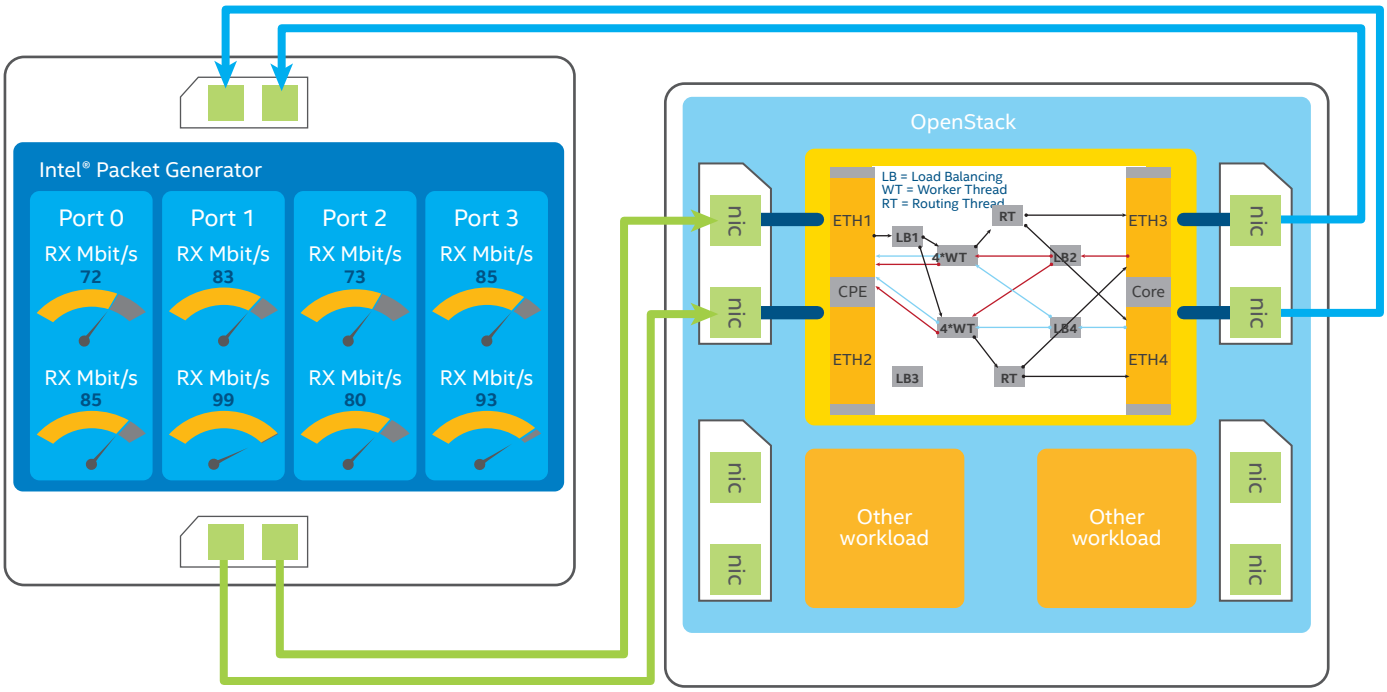

Figure 4. Test setup.

To study the effect of the new functionality we conducted tests of four different configurations:

1. OpenStack Juno without huge pages support.
2. Patched OpenStack enabling huge pages.
3. OpenStack with huge pages and enforced CPU affinity to ensure correct NUMA placement.
4. The same configuration as 3 with manually enforced 1:1 mapping of virtual CPU threads to logical CPU cores.

At the time of writing, the Intel DPPD had not been updated to interoperate with the OpenStack Juno SR-IOV capability so the SR-IOV results are based on setting up the SR-IOV connections outside of Neutron control.

The huge page capability is a feature for OpenStack Kilo so this capability was enabled by leveraging the OpenStack Juno patches on www.01.org.[14]

The Intel DPPD application required that the 16 virtual CPU cores be allocated to the application. For the best performance, all these vCPUs must be from the same socket on the server.

The OpenStack NUMA placement capability was leveraged to help deploy the required guest vCPU topology on the host. The OpenStack Juno functionality does not include support for I/O awareness as part of the NUMA placement decision. This means that the guest topology cannot be specified to include I/O device locality. (For more information, refer to the I/O-Aware NUMA Scheduling section.) To work around this lack of functionality, the test environment was set up and checked to ensure that the platform I/O device was local to the chosen NUMA cell the VM was deployed on.

It is expected that future versions of OpenStack will include support for the isolate CPU Thread Pinning policy that was not supported in OpenStack Juno. For the test setup, the CPU pinning was performed using Linux primitives.

Figure 5 shows the cumulative gains achieved by applying these configuration changes to the environment. The relative improvements are dependent on the order in which the features were enabled, so these relative value data points should not be interpreted as absolutes. Instead the reader should consider the cumulative value of enabling these four features and how this reference NFV workload was enabled to deliver line-rate performance with 256-byte packets on a 10 Gbps Ethernet link with zero packet loss.

Latency and Packet Delay Variance characteristics are other aspects of performance improvement that are not displayed in Figure 5 but are relevant for NFV workloads. Specific measurements were not made during this test, but significantly notable improvements in the predictability and stability of throughput readings were observed in successive test runs, particularly after enabling CPU pinning.

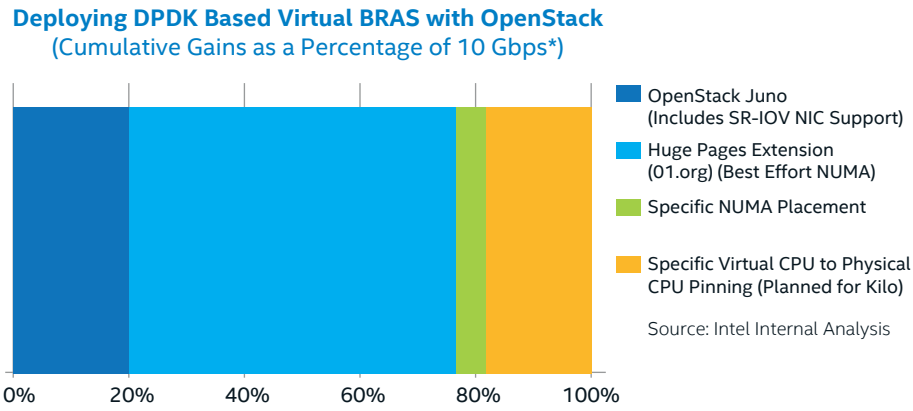

Figure 5. Cumulative performance impact on Intel® Data Plane Performance Demonstrators from platform optimizations.

Some related studies that include latency measurements have been conducted with DPPD. The paper on QoS in Broadband Remote Access Servers with Linux and Intel architecture[16] presents a number of results focusing on latency characteristics. Latency and PDV for Intel DPPD deployed with OpenStack are likely to be areas of analysis in a future paper.

Intel® Technologies for Optimal NFV

To efficiently deploy virtualized, high-performance networking applications on industry-standard, high-volume servers, a number of processor and platform capabilities can be leveraged to significant effect.

Intel® Virtualization Technology for IA-32 and Intel® 64 Processors

Intel VT-x adds extensions to enable high-performance virtualization solutions that help to increase server utilization and reduce costs. The extensions are called Virtual Machine Extensions (VMX). VMX introduces 10 instructions specific for virtualization to the instruction set.

Intel® Virtualization FlexPriority (Intel® VT FlexPriority)

The local Advanced Programmable Interrupt Controller (APIC) has a register for tracking the current running process priority called the Task Priority Register (TPR). The VMM is required to track guest writes to the virtualized TPR to know when is safe to inject interrupts to the VM. This implies that every time the guest writes to the virtualized TPR, usually on task context switch, a VM exit is generated. Intel VT FlexPriority support removes the need for the VMM to track writes to the virtualized TPR register thereby reducing the number of VM exits.

Intel® Virtualization FlexMigration (Intel® VT FlexMigration)

The CPUID instruction is used to report the feature set of a processor. When an application initializes, it typically uses the CPUID instruction to check feature availability. To support live migration—that is, migration with minimal downtime and without restarting the VM—the destination processor must support the same feature set as the source processor. Intel VT FlexMigration enables the VMM to virtualize the CPUID instruction. This puts the VMM in control of what features are exposed to the guest and hence enables the VMM to manage the features exposed in a pool of processors. The processors may have different feature sets, but the VMM can make them look the same to guests by means of the virtualized CPUID instruction.

Virtual Processor Identifiers (VPID)

The MMU has a Translation Look-aside Buffer (TLB) to cache address translations from virtual to physical memory. The TLB was extended with a Virtual Processor ID (VPID) field. This field enables VM-related address translation entries to persist in the TLB between context switches, removing the need for the TLB to be repopulated after a context switch This enables VM context switches to happen without the previously associated TLB flush operation.

Extended Page Tables

Extended Page Tables (EPT) is a feature of the MMU. The EPT feature improves performance by removing the need for the hypervisor to trap VM updates to page tables, a feature known as page table shadowing. EPT supersedes page table shadowing by maintaining a second level of page tables in the VMM that describe guest-physical address to host physical address mappings.

VMX Preemption Timer

The VMX Preemption Timer is an additional platform timer intended to enable the hypervisor to preempt VM execution after a specific amount of time. This removes the need to use another platform timer to facilitate context switches.

Pause Loop Exiting

Pause Loop Exiting (PLE) is a hardware feature that forces a VM exit when the spinlock loop duration expiry event is detected. This enables the VMM to schedule another vCPU and help to reduce the impact of lock holder preemption.

Intel® Virtualization Technology for Directed I/O

Address Translation Services

The Peripheral Component Interconnect Special Interest Group (PCI-SIG) defined the Address Translation Services (ATS) specification as part of the I/O Virtualization (IOV) Specifications. ATS specifies a set of transactions that PCI Express components can use to support I/O Virtualization thereby enabling a PCIe* device to perform a Direct Memory Access (DMA) directly into the VM memory.

Intel VT-d is an implementation in hardware that can perform the address translation needed for DMA transactions in virtual environments. In essence this translates between guest physical addresses and host physical addresses to help ensure that the DMA transaction targets the correct physical page with protection mechanisms to prevent memory pages related to other VMs being affected.

Large Intel VT-d Pages

The Large Intel VT-d Pages feature enables large 2 MB and 1 GB pages in Intel VT-d page tables. Leveraging these huge page table entries in the I/O Memory Management Unit (IOMMU) helps to reduce I/O Translation Look-Aside Buffer (IOTLB) misses which in turn helps to improve networking performance, particularly for small packets. (For more information see the Huge Pages section.)

Interrupt Remapping

Interrupt Remapping enables the routing and isolation of interrupts to CPUs assigned to VMs. The remapping hardware forces the isolation by tracking the interrupt originator ID, and can also cache frequently used remapping structures for performance enhancements.

Intel® Virtualization Technology for Connectivity

Intel VT-c enables improved networking throughput with lower CPU utilization and reduced system latency. This technology is a feature in Intel's range of Ethernet Server Adapters such as the Intel® 82599 10 Gigabit Ethernet controller.

Virtual Machine Device Queues

Virtual Machine Device Queues (VMDQ) is a feature of Intel Ethernet Server Adapters that accelerates network performance by filtering Ethernet frames into VM specific queues based on the VLAN ID and destination MAC address. The VM specific queues can then be affinitized with specific cores improving cache hit rates and overall performance.

PCI-SIG Single Root I/O Virtualization

PCI-SIG SR-IOV allows partitioning of a single Intel® Ethernet Server Adapter port, also known as the Physical Functions (PF). This PF is a full PCIe function that includes the SR-IOV Extended Capability (used to configure and manage the SR-IOV functionality) into multiple VFs. These VFs are lightweight PCIe functions that contain the resources necessary for data movement but minimize the set of configuration resources. They may be allocated to VMs each with their own bandwidth allocation. They offer a high-performance, low-latency data path into the VM.

Figure 6 shows a high-level view of an SR-IOV implementation on an Intel architecture-based platform. It depicts how SR-IOV can be used to bypass the hypervisor to deliver the high-performing, low-latency path into the VM.

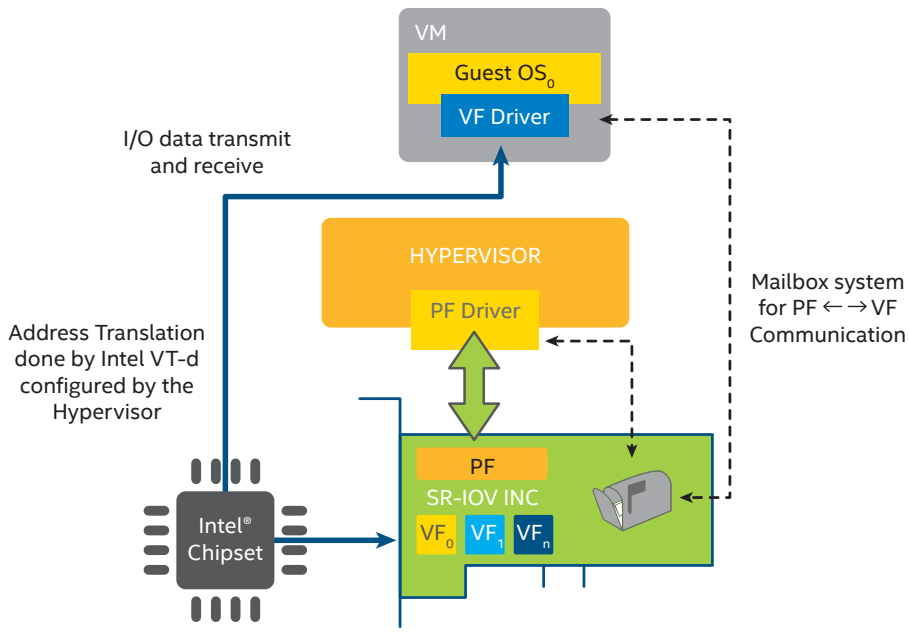

Figure 6. High-level Single Root I/O Virtualization architecture.

Non-Uniform Memory Architecture

Uniform Memory Architecture (UMA) refers to a design where every processor core in the system has the same memory performance characteristics such as latency and throughput, typically as a result of having a shared interconnect to get to the memory.

In many modern multiprocessor systems such as Intel Xeon processors, NUMA design is commonly used. With this model, processor cores have access to locally attached memory with higher performing memory access characteristics. Processor cores also have access to "remote" memory over some shared interconnect that is "locally" attached to another processor core.

A NUMA topology is depicted in Figure 7. The green arrows represent the highest performing memory access paths where a process executing on a CPU core is accessing locally attached memory. The orange arrows represent a suboptimal memory access path from the processor cores to "remote" memory.

The use of a NUMA-based design facilitates very high performance memory characteristics. However to leverage the resources in the platform in the most efficient way, the memory allocation policy and the process/thread affinities with processor cores must be taken into consideration. The memory allocation policy can typically be controlled with OS APIs. The OS can be instructed to allocate memory for a thread that is local to a processor core that it is executing on or is local to a specified core. The latter is particularly useful when a software architecture is used that leverages memory management threads that allocate memory for other worker threads in the system. The section below describes the core affinity consideration.

CPU Pinning

The OSs have a scheduler that is responsible for allocating portions of time on the processor cores to the threads that are running in the system. In a multicore processor, the scheduler generally has the flexibility and capability to move threads between processor cores. It does this to help load balance the workloads across all of the cores in the system. In the context of a NUMA-based system, this capability can mean that a thread that was executing on a processor core and accessing local memory could find itself executing on another core during a subsequent time-slice, accessing the same memory locations, but the memory is now at a remote location relative to the processor core.

CPU pinning is a technique that allows processes/threads to have an affinity configured with one or multiple cores. By configuring a CPU affinity, the scheduler is now restricted to only scheduling the thread to execute on one of the nominated cores. In a NUMA configuration, if specific NUMA memory is requested for a thread, this CPU affinity setting will help to ensure that the memory remains local to the thread. Figure 8 shows a logical view of how threads that can be pinned to specific CPU cores and memory can be allocated local to those cores.

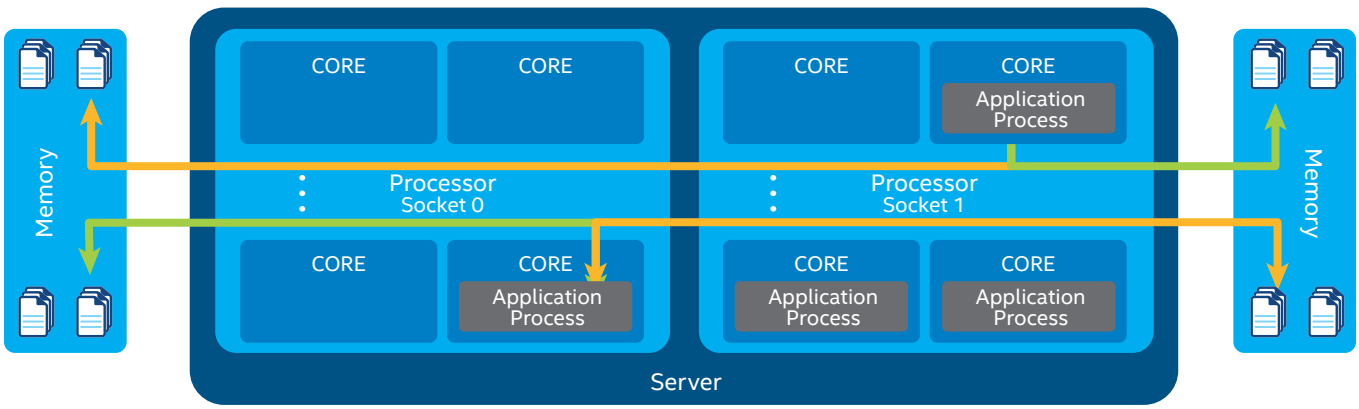

Figure 7. Non-Uniform Memory Architecture.

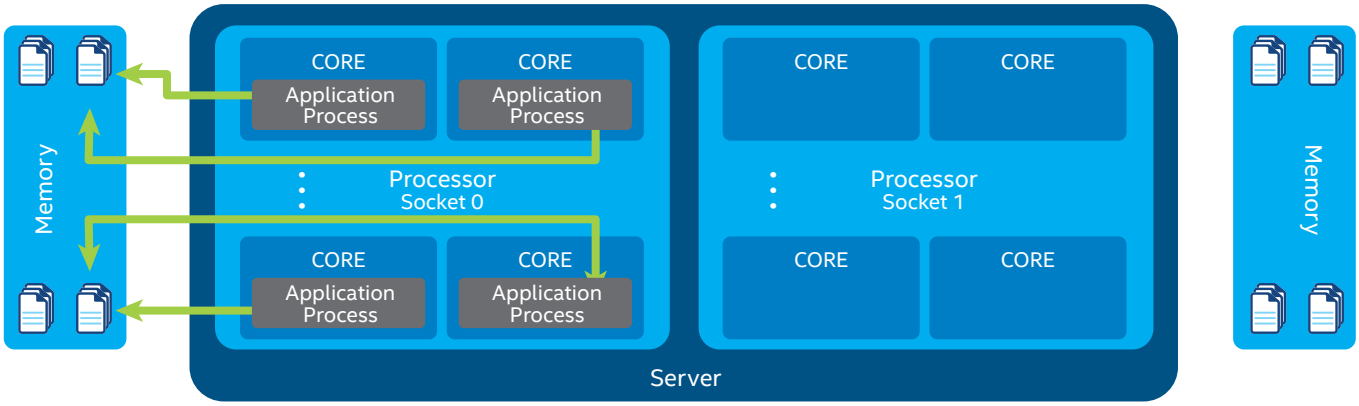

Figure 8. CPU affinity with optimal placement of Non-Uniform Memory Architecture.

Huge Pages

Huge page support in the MMU TLB helps the TLB cache a greater range of memory translations. A subset of steps involved in a memory address translation request is shown in Figure 9. In this example, the use of the small page table entries (4 KB) results in less coverage of the memory address space in the TLB. This can potentially lead to a greater number of TLB misses. A TLB miss causes a memory access to read the page table to get the requested address translation entry. A large number of TLB misses adds a greater number of memory accesses. This can result in suboptimal performance for the application that required the memory address translation.

In Figure 10, huge page table entries are being used. In this case, the 1-GB page table entry sizes result in far greater coverage of the memory space in the TLB, which in turn helps to minimize the burden of TLB cache misses.

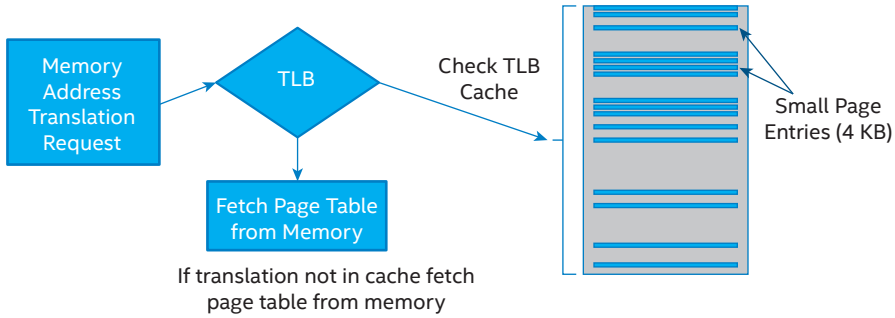

Figure 9. Conceptual view of memory address translation with small page table entries.

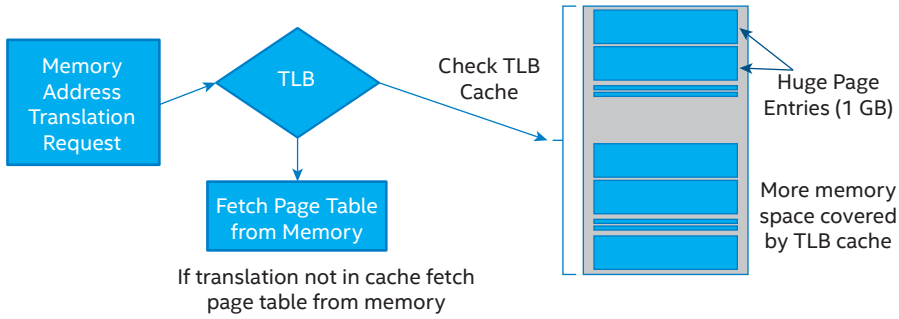

Figure 10. Conceptual view of memory address translation with huge page table entries.

Conclusion

The ETSI-NFV ISG is making significant progress developing specifications to guide network transformation, influence industry standards bodies, and create open-source software stacks.

Intel architecture-based platforms offer a number of capabilities that can provide benefits when deploying virtualized workloads with high performance demands. The Intel VT-x, Intel VT-d, and Intel VT-c capabilities all come together to offer an exceptional experience with virtualized applications. Combining these capabilities with a range of tuning facilities in the platform helps workloads to deliver incredible performance.

OpenStack is a leading open-source software suite for creating and managing private and public clouds infrastructure. For OpenStack to be suitable for deploying high-performance NFV workloads there are a number of additional features that are required to enable an Enhanced Platform Awareness and unleash the performance potential of the NFV applications.

Using the Intel DPPD as a reference NFV application, it was shown that by leveraging SR-IOV connectivity, NUMA-aware quest OS configuration, huge page table configuration, and CPU pinning could help to deliver 10-Gbps line-rate capable performance.

Some of these features have already landed in the OpenStack Juno and Kilo releases. Intel and other members of the OpenStack community are working on actively contributing other NFV-enabling capabilities into subsequent OpenStack releases.

Acronyms

API . . . . . Application Programming Interface

APIC . . . . . Advanced Programmable Interrupt Controller

ATS . . . . Address Translation Services

BNG . . . . Broadband Network Gateway

BRAS . . . . . Broadband Remote Access Server

CLI . . . . . Command Line Interface

CPE . . . Customer Premise Equipment

CPU . . . . . .Central Processing Unit

DHCP . . . . . .Dynamic Host Configuration Protocol

DMA . . . . . Direct Memory Access

DSLAM . . . . . Digital Subscriber Line Access Multiplexer

EPT . . . . . Extended Page Tables

ETSI . . .European Telecommunications Standards Institute

FW . . . . . .Firewall

IOMMU . . . . . Input/Output Memory Management Unit

IOTLB . . . . . .Input/Output Translation Look-Aside Buffer

IP . . . . . .Internet Protocol

IPAM . . . . . Internet Protocol Address Management

ISG . . . . Industry Specification Group

KVM . . . . . Kernel Virtual Machine

LAN . . . . . Local Area Network

LB . . . . . .Load Balancer

MMU . . . . Memory Management Unit

MSAN . . . . Multi-Service Access Nodes

NFV . . . . . .Network Functions Virtualization

NIC . . . . . .Network Interface Card

NUMA . . . . . Non-Uniform Memory Architecture

OPNFV . . . Open Platform for Network Functions Virtualization

OS . . . . . Operating System

PCI-SIG . . . . . Peripheral Component Interconnect Special Interest Group

pCPU . . . . . Physical Central Processing Unit

PLE . . . . . Pause Loop Exiting

PPP . . . . . .Point-to-Point Protocol

PPPoA . . . . . .Point-to-Point Protocol over ATM

PPPoE . . . . . .Point-to-Point Protocol over Ethernet

QEMU . . . . . .Quick EMUlator

QoS . . . . . Quality of Service

SR-IOV . . Single Root I/O Virtualization

RT . . . . . .Routing Thread

SDN . . . Software Defined Networking

SMT . . . Simultaneous Multi-Threading

TLB . . . .Translation Look-aside Buffer

TPR . . . . . Task Priority Register

UMA . . . Uniform Memory Architecture

vCPU . . Virtual Central Processing Unit

VF . . . . . Virtual functions

VIF . . . . . Virtual Interface Function

VIM Virtualized Infrastructure Manager

VLAN . . . . Virtual Local Area Network

VM . . . . . .Virtual Machine

VMDQ . . . . . .Virtual Machine Device Queues

VMM . . . . . Virtual Machine Monitor

VMX . . . . Virtual Machine Extensions

VNF . . . .Virtualized Network Function

VPID . . . . .Virtual Processor Identifiers

VPN . . . . . .Virtual Private Network

VT . . . . . Virtualization Technology

VXLAN . . . . . Virtual Extensible Local Area Network

WT . . . . . Worker Thread

Authors

Adrian Hoban, Przemyslaw Czesnowicz, Sean Mooney, James Chapman, Igor Shaula, Ray Kinsella, and Christian Buerger are software architects and engineers with the Network Platforms Group at Intel Corporation. They focus on Network Functions Virtualization and software-defined networking extensions in open-source components such as OpenStack.

Acknowledgements

This work could not have been completed without the significant effort and innovation by the teams that developed DPDK and Intel DPPD.

Keywords

Orchestration, OpenStack, Network Functions Virtualization, Software-Defined Networking

[1] European Telecommunications Standards Institute Network Functions Virtualization. http://www.etsi.org/technologies-clusters/technologies/nfv

[2] "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action." Introductory white paper. http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[3] "Network Functions Virtualisation: Network Operator Perspectives on Industry Progress." Update white paper. http://portal.etsi.org/NFV/NFV_White_Paper2.pdf

[4] "Network Functions Virtualization – White Paper #3, Network Operator Perspectives on Industry Progress." http://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf

[5] ETSI NFV specifications. http://www.etsi.org/technologies-clusters/technologies/nfv

[6] ETSI GS NFV-PER 001 V1.1.1. "Network Functions Virtualisation (NFV); NFV Performance & Portability Best Practises." http://www.etsi.org/deliver/etsi_gs/NFV-PER/001_099/001/01.01.01_60/gs_NFV-PER-001v010101p.pdf

[7] OpenStack. www.OpenStack.org

[8] Intel® Data Plane Performance Demonstrators. https://01.org/intel-data-plane-performance-demonstrators

[9] Data Plane Development Kit. www.dpdk.org.

[10] BSD 3-Clause License. http://opensource.org/licenses/BSD-3-Clause

[11] "Network Function Virtualization: Virtualized BRAS with Linux* and Intel® Architecture" white paper. http://networkbuilders.intel.com/docs/Network_Builders_RA_vBRAS_Final.pdf

[12] "Using Intel® AES New Instructions and PCLMULQDQ to Significantly Improve IPSec Performance on Linux." http://www.intel.com/content/www/us/en/intelligent-systems/wireless-infrastructure/aes-ipsec-performance-linux-paper.html

[13] Virt driver guest NUMA node placement & topology. https://blueprints.launchpad.net/nova/+spec/virt-driver-numa-placement

[14] Huge Page and I/O NUMA scheduling patches for OpenStack Juno. https://01.org/sites/default/files/page/OpenStack_ovdk.l.0.2-907.zip

[15] "OpenStack* Networking with Intel® Architecture: Getting Started Guide." https://01.org/sites/default/files/page/accelerating_openstack_networking_with_intel_architecture_rev008.pdf

[16] "Network Function Virtualization: Quality of Service in Broadband Remote Access Servers with Linux* and Intel® Architecture." https://networkbuilders.intel.com/docs/Network_Builders_RA_NFV_QoS_Aug2014.pdf

Printed in USA 0715/LB/MESH/PDF ♻ Please Recycle 332169 002US