Building High-Performance NFV Solutions Using Containers

Jun Nakajima

Contributors:
Sainath Grandhi, Yunhong Jiang, Krishna Murthy, Guangrong Xiao
Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.

Intel may make changes to specifications and product descriptions at any time, without notice.

All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2015 Intel Corporation.
Classical Network Appliance Approach

ETSIs Vision
European Telecommunications Standards Institute

Software
IT
Virtualization
Standard High Volume Machines

Network Virtualisation Approach
Architecture Framework

KVM, Xen, Containers

Virtual Network Function
Why Containers for NFV?

- Instant booting
  - Very quick deployment
  - May be useful for scaling out NFV apps (W. I. P.)
- Low latency
  - Transitions
- Low overhead
  - No virtualization overhead
- Established provisioning and management tools
Challenges when building high-performance NFV solutions
Enhancements for NFV Hypervisor

1. Exclusive allocation of whole CPU cores to VMs
2. Direct I/O (e.g. SR-IOV)
3. Inter-VM Communication (direct-memory mapped)
4. vSwitch implementation as a high performance VM
5. Fast Live Migration

From ETSI
“Network Functions Virtualization (NFV): Infrastructure; Hypervisor Domain”
Network Configuration for SR-IOV
Overview of SR-IOV* NIC

Automatically loaded

PF: Physical Function
VF: Virtual Function

#: Set by Software

*: Defined as Part of PCI-SIG

VXLAN  Geneve  NVGRE
Using SR-IOV NIC for VNF (Virtual Network Function) Containers

Pros:
- Low latency using user-mode driver
- H/W offloads
- HW-based QoS
- High-performance

Cons:
- Limited bandwidth inside NIC
- Limited # of VFs
- NIC-vendor specific issues
- No live migration

Covered later
Not so important for High-Performance NFV
Setups for Containers

1. Assign VF(s) to each container
2. Set up MAC address (if needed)
3. Set VLAN tag (if needed)
4. Set up IP address(es)
5. Set up routing as needed
Setting Up SR-IOV VFs

1. Set # of VFs*
   • Use `ip` command to know about PF/VFs

```
# echo 8 > /sys/bus/pci/devices/0000:01:00.0/sriov_numvfs
```

```
# ip link show

3: enp1s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether a0:36:9f:34:01:2c brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 1 MAC 1e:3a:20:05:98:d2, spoof checking on, link-state auto
    vf 2 MAC 9e:d0:a3:85:57:45, spoof checking on, link-state auto
    vf 3 MAC 12:11:27:16:2f:e4, spoof checking on, link-state auto
    vf 4 MAC 7e:32:01:9d:8b:be, spoof checking on, link-state auto
    vf 5 MAC 5e:97:21:2c:bd:19, spoof checking on, link-state auto
    vf 6 MAC 42:1a:c7:a9:39:71, spoof checking on, link-state auto
    vf 7 MAC 92:04:9d:d0:89:23, spoof checking on, link-state auto

8: enp2s16f4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 9e:d0:a3:85:57:45 brd ff:ff:ff:ff:ff:ff

9: enp2s16f6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 12:11:27:16:2f:e4 brd ff:ff:ff:ff:ff:ff
```

*: If you see “not enough MMIO resources for SR-IOV”, try to add “pci=realloc” in boot parameter
Setting Up SR-IOV VFs (Cont.)

2. Create a Docker Container w/o network

```
$ sudo docker run -it --net=none <Image> /bin/bash
root@d6e6f101ee08:/#
```

3. Create netns associated with the container:
   • Find PID
   • Create netns for that

```
$ pid=$(docker inspect -f '{{.State.Pid}}' d6e6f101ee08)
```

```
$ sudo mkdir /var/run/netns
$ ln -s /proc/$pid/ns/net /var/run/netns/$pid
```

4. Assign a VF to netns
   • Use the Ethernet interface for VF

```
$ vf=enp2s16f6
$ sudo ip link set $vf netns $pid
```

5. Set IP address
   • $ipaddr specifies IP address you need to set

```
$ sudo ip netns exec $pid ip addr add $ipaddr dev $vf
$ sudo ip netns exec $pid ip link set $vf up
```
Tools

- Pipework
  - [https://github.com/jpetazzo/pipework](https://github.com/jpetazzo/pipework)
  - Create a virtual interface with a macvlan bridge
- Pipework forked
  - [https://github.com/Rakurai/pipework](https://github.com/Rakurai/pipework)
  - SR-IOV VF support
Troubleshooting and NIC-Specific Issues

- Spoof checking
  - MAC anti-spoofing: Some NICs allow one to disable, but some don’t.
- Driver-specific
  - Rate limits setting
- Inter-Container communication

$ sudo ip link show
...
  4: enp1s0f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
     link/ether a0:36:9f:34:01:2e brd ff:ff:ff:ff:ff:ff
     vf 0 MAC 7e:ae:4a:f6:cc:67, vlan 42, spoof checking off, link-state auto
     vf 1 MAC fe:aa:34:63:37:b4, vlan 42, spoof checking off, link-state auto
     vf 2 MAC 86:c5:96:1c:aa:b0, vlan 1, spoof checking off, link-state auto
...
**Performance and Bandwidth**

**Aggregated throughput should reach or exceed** max line rate (e.g. 10GbE):
- Includes VF-to-VF communication

**Inter-Container communication using VFs:**
- **Aggregated throughput** = $\alpha \times $ (Max Line Rate)
- $\alpha$: Depends on packet sizes, NIC, platform, etc. e.g. $\approx 1.4^*$
- Use Rate Limiting (Tx)
- Use VLAN tagging to isolate

```
# ip link set enp1s0f0 vf 0 rate 2500
# ip link set enp1s0f0 vf 0 vlan 412
```

*$^*$: Based on iperf (TCP)
DPDK Runs in Docker Container

- Yes, it does*
- Need to expose host resources with privileges elevated:
  - PCI devices, and
  - More to achieve high-performance
- Can expose more attack surfaces

Deterministic Execution and Minimal Latency in Containers
Quick Tryout: Cyclictest

- Run cyclictest* in Container making noise outside
- Measure latency of expected timer notification
- Compare shared CPUs vs. isolated CPUs
- Compare Kernel V4.1 vs. V4.1 + RT patch

15µs or less is required

<table>
<thead>
<tr>
<th>(unit: µs)</th>
<th>Shared CPUs (Min, Avg, Max)</th>
<th>Isolated CPUs (Min, Avg, Max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V 4.1</td>
<td>6, 10, 70</td>
<td>3, 9, 48</td>
</tr>
<tr>
<td>V4.1 + RT</td>
<td>4, 18, 57</td>
<td>2, 8, 16</td>
</tr>
</tbody>
</table>

Measured on Intel® Xeon® CPU E5-4620 v2 @ 2.60GHz (w/ 16 cores) + 420GB memory
Causes of Latencies

Asynchronous Events
- Interrupts, Cache/TLB Misses

Software
- Spin Locks/Loops (in app), OS Scheduling

Hardware
- Power Management, NIC (e.g. SR-IOV VFs)

*: VNF (Virtual Network Function)
Summary of Solutions

- **Exclusive Resource Allocation**
  - Reserve CPUs, Huge Pages

- **Software**
  - Real-Time Configuration
    - Code inspection, testing/measurements

- **Hardware Technologies**
  - Cache Allocation Technology, SR-IOV
    - NIC

- **Real-Time Containers**
  - Virtual Switch (vSwitch)
  - Linux Kernel
  - Hardware

- **Real-Time Containers**

- **Containers**
Cache Allocation Technology

- Last Level Cache partitioning mechanism enabling the separation of an application
- Processes (thus Containers) can be isolated to increase determinism
- Having limited cache is still better than “unlimited cache and noisy neighbors”

CAT is supported on the following 6 SKUs for Intel Xeon processor E5 v3 family: E5-2658 v3, E5-2658A v3, E5-2648L v3, E5-2628L v3, E5-2618L v3, and E5-2608L v3 and Intel(R) Xeon(R) processor D family.
Exclusive Allocation of CPUs

isolcpus:
- Boot-time (Kernel boot parameter)
  - ... default_hugepagesz=1G ... isolcpus=12-15
- Isolation from timers from other CPUs.

Cgroups/cpuset.cpus:
- Run-time
  - Isolate target CPUs (Next Page)
  - Run Container on those CPUs
- Same as isolcpus except the hrtimer issue

$ docker run -ti --cpuset-cpus="12-15" ...
Isolate target CPUs

Workload in “Cpuset A” can be impacted by workload in Root

Solution:
1. Create a directory “subroot” in cpuset root
   • cgroups.*, cpuset.*, tasks, etc. are automatically populated
2. Set cpuset.cpus for subroot
   • Exclude the CPUs for Cpuset A
   • Need to set cpuset.mems prior to that
3. Run Container using Cpuset A
   • Docker will create a cpuset

Get isolated CPUs offline
Get them back to online
Tools: cset

https://code.google.com/p/cpuset/

• Add a patch (if you see the problem below)
  https://code.google.com/p/cpuset/issues/detail?id=10

• Create subroot cgroup

```
# cset set --cpu=`cat /sys/fs/cgroup/cpuset/cpuset.cpus` \ 
  --mem=`cat /sys/fs/cgroup/cpuset/cpuset.mems` --set=subroot
```

• Move all tasks from root to subroot
  • Including kernel threads except for any bound threads
  • Some are not moved (if bound)

```
# cset proc --move --kthread --fromset=root --toset=subroot
```
Huge Pages and NUMA Nodes

Huge pages: 2MB and 1GB pages:
- 1GB pages are optimal
- Host needs to boot with hugepages option
- **Down-side:** no longer swap memory

Make them available to containers:
- Set up on the host, and allow them to mount as volume (e.g. “-v”), or
- Allow them to do `mount -t hugetlbfs` in containers (w/ privileges elevated or cap-add),

Per-Node Huge pages:
- Write access to: `/sys/devices/system/node/node[0-9]*/hugepages.nr_hugepages`
Recap of Solutions

- Reserve system resources
  - CPUs (Isolcpus, cgroups/cpuset.cpus), memory (cgroups/cpuset.mems, huge pages), I/O for direct assignment
- Use realtime-ready software
  - Realtime configuration for the kernel, applications, libraries, etc.
  - Kernel boot parameters
- Disable H/W features that can cause latency
  - Deep C-states, etc. (in BIOS settings)
- Enable H/W features that isolate H/W resources thus lower latency
  - CAT (Cache Allocation Technology)
  - SR-IOV Rate Limiting
Minimal Latency and High-Performance vs. Isolation and Security

- Need to expose host system resources (/sys):
  - PCI devices, huge pages, NUMA nodes, kernel modules, …
- Inter-Container communication:
  - Shared memory (IPC)
    - Use Linux bridge with DPDK (early stage) instead
- Thus, expose more attack surfaces
  - Boot parameter “selinux=0” lowers latency…
- You may need optimized kernels

Options:
- Do those for trustworthy containers,
- Limit use cases,
- Minimize host system resources exposed,
  - Use “--cap-add/drop”
- Use VM-based containers with Docker support
  - Clear Containers, Hyper, …
  - “KVM as The NFV Hypervisor” at KVM Forum 2015
Summary and Next Steps

• NFV is pushing the limits of containers
  • Minimal latency and High-performance vs. Isolation and Security
• Solutions to subset of the NFV-related problems are available
  • There are other problems to be solved
• Continue to work with industry partners
  • E.g. OPNFV