vDPA-net Live Migration with Shadow VirtQueue

Eugenio Perez Martin
Sr. Software Engineer <eperezma@redhat.com>

Gautam Dawar
Sr. Manager Software Development <gdawar@amd.com>
Agenda

- SR-IOV
- Live Migration
- Problem: LM with passthrough VF
- Solution: vDPA
- Cross-vendor VM Live Migration Demo
- Shadow virtqueue operation
- Q&A
SR-IOV
VM Live Migration

- **What is “Live” Migration?**
  - Process of moving a VM running on one physical host to another while the guest OS is **running**
  - Useful for load balancing, hardware/software maintenance, etc.

- **How does it happen?**
  - Marking modified RAM pages as **“dirty”**
  - Sending these dirty RAM pages to the destination until a **threshold** is reached
  - Stop guest, **transfer remaining** dirty RAM, device state
  - **Resume execution** on destination
Live Migration: SR-IOV VF Passthrough

- Requires identical NIC HW on both source and destination host
  - Tight coupling between the Guest SW and Host HW
  - Vendor’s VF driver required in the Guest OS
Virtual I/O Device (VIRTIO)

- Virtio is a virtualization specification that describes virtual devices, drivers and how they interact.
- Virtio spec defines how to create a control plane and the data plane between the guest and host.

**Data plane**
- Composed of buffers and rings layouts
- Used for transferring the actual (bulk) data (packets) between host and guest

**Control plane**
- For establishing and terminating the data plane.
- Feature negotiation, vring configuration, etc.
vDPA
vDPA dataplane & control plane
vDPA dataplane & control plane
vDPA dataplane & control plane
Live Migration with vDPA

- Live migration should be transparent to the guest
  - it only sees a virtio-net device, irrespective of actual vendor HW
  - Hypervisor doesn’t require guest’s collaboration
Live Migration - Setup

- Source host (dell750-28) has two NICs
  - **AMD Xilinx SN1022**
  - Physical port running `iperf` server
- Destination host (dell750-23) has single NIC
  - **Mellanox ConnectX 6**
  - These NIC ports are connected via Switch and configured with *same VLAN ID*
Demo

Watch it online here
vDPA: CVQ bypassing QEMU
vDPA: CVQ bypassing QEMU
vDPA: CVQ bypassing QEMU
vDPA: CVQ bypassing QEMU
vDPA: CVQ bypassing QEMU

[Diagram showing the flow of data between the host, Guest, and device, with symbols and meanings for different paths and components.]
vDPA: shadowed CVQ
vDPA: shadowed CVQ
vDPA: shadowed CVQ
vDPA: shadowed CVQ
vDPA: shadowed CVQ
Shadow virtqueue: migrate device state
Shadow virtqueue: migrate device state

```
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>virtio data path element</td>
</tr>
<tr>
<td></td>
<td>non-virtio data path element</td>
</tr>
<tr>
<td></td>
<td>virtio control path element</td>
</tr>
<tr>
<td></td>
<td>virtio shared memory</td>
</tr>
<tr>
<td></td>
<td>data path</td>
</tr>
<tr>
<td></td>
<td>control path</td>
</tr>
</tbody>
</table>
```

```
mac        00:66:77:88:99:aa
#vqs       3
```
Shadow virtqueue: migrate device state

```
mac     00:66:77:88:99:aa
#vqs    3
...
```

```
mac     00:66:77:88:99:aa
#vqs    3
...
```
Shadow virtqueue: migrate device state

Host User space

Source Qemu process

Guest

Data VQs, CVQs

virtio-net driver

vdpa device model

Destination Qemu process

Host User space

Guest

Data VQs, CVQs

mac 00:66:77:88:99:aa

#vqs 3

FEATURES_OK,
DRIVER_OK,
ENABLE CVQ

! ENABLE_DVQ

vendor specific vdpa driver

Device

VF

Vendor CMD

vendor specific driver
Shadow virtqueue: migrate device state

[Diagram showing the migration of device state from one host to another, involving virtio-net driver and vendor-specific VDPA driver.]
Shadow virtqueue: migrate device state

[Diagram of virtqueue migration process]

- **Source Qemu process**: Shadow virtqueue management
- **Destination Qemu process**: Migrated device state

- **Host User space**
  - **Host Kernel space**
    - **Vendor specific vDPA driver**
  - **Device**
    - **VF**

- **Symbol Legend**
  - Virtio data path element
  - Virtio control path element
  - Virtio shared memory
  - Data path
  - Control path

- **Network Configuration**
  - MAC: 00:66:77:88:99:aa
  - #vqs: 3
Shadow virtqueue: migrate device state
Shadow virtqueue: migrate device state
Shadow virtqueue: Dirty memory tracking

- SVQ already uses the QEMU emulated device infrastructure
  - So, dirty memory tracking is “for free” in terms of code changes
  - Improvements to dirty memory tracking are applied to SVQ if they apply to emulated device’s virtio
Shadow virtqueue: Regular operation
Shadow virtqueue: allocating SVQ vring
Shadow virtqueue: filling dirty bitmap
Shadow virtqueue: filling dirty bitmap
Shadow virtqueue: filling dirty bitmap
Shadow virtqueue: configure new vring

qemu:
  set_config(...)  
  set_vring_addr()  
  set_state(DRIVER_OK)
Shadow virtqueue: filling dirty bitmap
Shadow virtqueue: filling dirty bitmap
Shadow virtqueue: Recap

- No changes in the device or the guest to
  - Track device state
  - Track dirty memory
  - Restore device state

- Changes for new features are
  - about the same as adding it to qemu
  - plus, code to send them through CVQ
# Vendor support for vDPA

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Device / Card</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD-Xilinx</td>
<td>Alveo SN1022</td>
</tr>
<tr>
<td>AMD-Pensando</td>
<td>Pensando DSC-200</td>
</tr>
<tr>
<td>Nvidia</td>
<td>Mellanox MT2892 [ConnectX-6 Dx]</td>
</tr>
<tr>
<td>Intel</td>
<td>N3000, C5000X-PL, F2000X-PL</td>
</tr>
<tr>
<td>Solid-run</td>
<td>SolidNET LX2162A DPU</td>
</tr>
<tr>
<td>Alibaba</td>
<td>Alibaba ENI (Elastic Network Interface)</td>
</tr>
</tbody>
</table>
Alveo SN1000 SmartNIC Accelerator Card

- **Industry’s first** SmartNIC offering software-defined hardware acceleration for all function offloads in a single platform

- Supports **custom offloads at line rate**, including customer-built and third-party offloads
  - **Network**: Open vSwitch and virtualization acceleration (Virtio.net)
  - **Security**: IPsec, kTLS and SSL/TLS
  - **Storage**: Virtio.blk, NVMe™ over TCP, Ceph, and compression & crypto services

- Based on the AMD 16nm **UltraScale+™ architecture**
- Powered by the **low-latency** XCU26 FPGA
- **16-core Arm®** processor.

- **P4 Programmability**: Vitis Networking, P4 toolkit from AMD, enables customers to compose custom offloads and tweak existing offloads
vDPA: More information

Thanks!

Questions?
vdpa: dirty page tracking alternatives

- Alternatives proposed
  - Device based
    - Dirty bytemap -> 8x times more memory, bad cache usage, …
    - Dirty ring -> [https://lwn.net/Articles/833206/](https://lwn.net/Articles/833206/)
  - IOMMU based
    - Page Request Interface PRI -> Not available at the moment?
  - Software based (QEMU)
    - failover
    - SVQ
      + Not related to guest’s memory size but host’s memory bandwidth. Automatic throttle for migration case.
      + Device does not need to learn new format (Virtio queue).
      + Re-uses emulated device -> Well tested and maintained.