SR-IOV Support

Motivation

Single Root I/O Virtualization (SR-IOV) is a virtualization technology that allows one physical device (e.g. a network adapter) to be used by multiple virtual machines simultaneously without the mediation from the hypervisor. The actual physical device is known as a Physical Function (PF) while the others are known as Virtual Functions (VF). With this feature, the hypervisor can assign one or more VFs to a Virtual Machine so that the guest can then use the device as if it was directly assigned.

PFs are fully featured PCIe functions which can be discovered, managed, and manipulated like any other PCIe device. PFs have full configuration resources, and can be used to configure or control the PCIe device. On the other hand, VF is a lightweight PCIe function that shares one or more physical resources with the Physical Function and with other VFs that are associated with the same PF. Each VF has its own set of RX/TX queues and lightweight PCI functions - just enough to send and receive data by means of direct memory access (DMA) between a VM and the physical device, completely bypassing the hypervisor and sparing CPU costly interrupts.

An SR-IOV enabled Ethernet network adapter will run a hardware L2 classifier to sort incoming packets using the MAC address or VLAN ID and assign them to corresponding VF queues. Each assigned VF communicates with the (vendor-specific) VF driver installed on the VM to exchange descriptors with destination addresses for packet data transfer. The IOMMU translates between I/O Virtual Addresses (IOVA) and physical memory addresses. This allows the virtual guest to program the device with guest physical addresses, which are then translated to host physical addresses by the IOMMU. The actual packet data transfer between the NIC and the VM memory is performed by DMA without interrupting the CPU.

Currently, EVE allows to either directly assign an entire physical network adapter into a VM, or to share a single port with multiple VMs using a software switch based on Linux bridge. While the former option reduces resource utilization efficiency, the latter software/interrupt-based approach cannot keep up with the bandwidth and latency requirements of VNFs.

To summarize, SR-IOV enabled network devices provide a high degree of scalability in virtualized environments as well as improved I/O throughput and reduced CPU utilization. Actually, even in cases where only a single VF is allocated for the physical device and dedicated to a single VM, the extra security features of SR-IOV, such as the MAC/VLAN spoofing protection, make the solution superior to the traditional direct assignment of the entire physical PCI device to a VM.

Limitations

The number of VFs that a particular PF can be divided into is limited. The SR-IOV specification sets 256 as the upper limit. In practice, most SR-IOV enabled NICs allow the creation of up to 64 VFs. For NFV edge use-cases this should be more than enough.

On the face of it, it may appear that hardware-based SR-IOV must always outperform software-based vswitch like Linux bridge or OVS. But in situations where the traffic flows East-West within the same server (between applications on the same edge node), OVS (accelerated by DPDK) will likely win against SR-IOV. This is because with OVS-DPDK, traffic is routed/switched within the server and not going back-and-forth between memory and the NIC. There is no advantage of bringing SR-IOV for east-west communication. Rather SR-IOV can become a bottle-neck: traffic paths are longer with PCI-e bus potentially limiting the bandwidth and NIC resources becoming over-utilized. In this case, it is better to route the traffic within the server using technology like DPDK. There is a good study from Intel which compares SR-IOV with DPDK-backed vhost interfaces in detail.

To conclude, SR-IOV is suitable for accelerating North-South traffic (beats DPDK in most benchmarks), but for service function chaining (SFC) on the same node it should be combined with high-performance software-based solutions such as DPDK, VPP, etc.

Requirements

First of all, SR-IOV must be supported by the NIC itself. There are plenty of high-throughput SR-IOV enabled NICs offered by Intel, Mellanox, Broadcom and other vendors. An SR-IOV capable driver corresponding to the selected NIC must be available and installed on the host as well as on every VM with assigned VF (e.g. `igb/ixgbe` for some of the Intel NICs). Note that there can be a different driver for the physical and for the virtual function (e.g. `i40e` and `i40evf`). The physical driver additionally exposes a management interface for the NIC.

Check that the NIC supports SR-IOV with:

```
lspci -s <NIC_BDF> -vvv | grep -i "Single Root I/O Virtualization"
```

Secondly, even if not involved in the packet data transfer, the hypervisor still needs to be aware of the SR-IOV technology and be able to properly configure PFs and VFs. The support for SR-IOV is mature and quite well documented on both kvm and xen.

On the Linux host / dom0, it is required to enable IOMMU by adding `intel_iommu=on` into the list of kernel command line parameters. It is recommended to also include `iommu=pt` (pass-through) to get the best performance. SR-IOV state information and configuration options are exposed through the sys filesystem and netlink interface.

SR-IOV virtual functions are instantiated when the PF driver is informed (typically from dom0) about the (maximum) number of required VFs:

```
echo <vf-num> > /sys/class/net/<device-name>/device/sriov_numvfs
```

However, once at least one VF is assigned to a domU, the number of VFs cannot be changed anymore. In order to modify the number of VFs, all domUs with assigned VFs have to be purged first.

With KVM/QEMU, VMs run as user-space processes and VFIO is used as a mechanism to allow direct I/O from Linux user space to PCI SR-IOV Virtual Functions.

In Xen, the xen-pciback / xen-pcifront drivers handle direct assignment.

When VFs are created, they are assigned IDs starting from 0 (0, 1, 2 …) and random MAC addresses by the PF driver. But aside from these IDs (which have no effect on networking), all VFs corresponding to the same PF are effectively equivalent until they get statically assigned MAC addresses and/or VLAN IDs (only then it matters which VF is assigned to which domU). MAC address and VLAN have to be configured before the assignment is made and cannot be changed without purging the domain.

Note that it is recommended to avoid random MAC addresses (and configure static ones), or else it will appear to applications running in domUs that NICs are changing with each reboot, which could be inconvenient especially for network-oriented applications, like VNFs.

Also note that there are special netlink calls used to configure MAC and VLAN on a VF (`IFLA_VF_VLAN`, `IFLA_VF_MAC`), different from those used for normal interfaces. VLAN is not created as a sub-interface, but rather configured directly for the VF.

EVE API Additions

SR-IOV support can be added into EVE with only few changes/additions to the API and even to the implementation. The API should allow to configure the maximum number of required VFs. We propose to make this part of the device model. For SR-IOV capable NICs we will introduce new values for `PhyIoType`: `PhyIoNetEthPF` and `PhyIoNetEthVF`, to be used in `PhysicalIO.ptype` and `Adapter.type`, respectively.

```
type PhyIoType int32

const (
  PhyIoType_PhyIoNoop    PhyIoType = 0
  PhyIoType_PhyIoNetEth  PhyIoType = 1
  …
  PhyIoType_PhyIoNetEthPF  PhyIoType = 8
  PhyIoType_PhyIoNetEthVF  PhyIoType = 9
)
```

To pass the number of required VFs, we could reuse `cbattr map` and define a new attribute `“sriov-vf-count”`, expecting integer as a value (which must be a power of two).

A model for SR-IOV capable device could look as follows:

```
{
  ...
  "ioMemberList": [
    {
      "ztype": 8,
      "phylabel": "eth0",
      "usage": 1,
      "assigngrp": "eth0",
      "phyaddrs": {
        "Ifname": "eth0"
      },
      "logicallabel": "eth0",
      "cost": 0,
      "usagePolicy": {},
      “cbattr”: {“sriov-vf-count”: “32”}
    }
  ]
}
```

Note that there is no need to represent each VF as a separate `PhysicalIO`. With the number of VFs up to 256 it would not be very practical anyway.

On the application side, we need to allow to configure the MAC address and the VLAN ID for an assigned VF. Adapter from devcommon.proto will be extended with `EthVF`:

```
message Adapter {
  org.lfedge.eve.common.PhyIoType type = 1;  // “9” for VF
  string name = 2;
  EthVF eth_vf = 3;
}

message EthVF {
  string mac = 1;
  Uint16 vlan_id = 2;
}
```

For VF assignment, `Adapter.type` should be set to `PhyIoNetEthVF` and MAC/VLAN optionally defined under `Adapter.eth_vf`. The underlying NIC of the VF (`PhysicalIO` of type `PhyIoNetEthPF`) should be referenced by `Adapter.name`. For other types of assignments, `eth_vf` should be left empty. Exceeding the number of available VFs will result in error.

For PF assignment (which is also a valid use-case), `Adapter.type` should be set to `PhyIoNetEthPF` and `Adapter.name` should point to `PhysicalIO` of the same type. Trying to assign PF or VF of a NIC not capable of SR-IOV (`PhyIoNetEth`) will result in error.

EVE Implementation Changes

We propose to implement SR-IOV management fully in the domainmgr, i.e. not across multiple zedbox microservices. This is because domainmgr is already the only microservice responsible for configuring direct device assignments. It receives `PhysicalIOAdapterList` from zedagent, converts each entry to `IOBundle`, collects PCI information and MAC addresses, moves devices between pciback and the host and publishes `AssignableAdapters`.

In the case of SR-IOV it would additionally:

1. Create VFs for `PhyIoNetEthPF` if requested through `PhysicalIO.cbattr["sriov-vf-count"]`:
   `echo <vf-num> > /sys/class/net/<device-name>/device/sriov_numvfs`
   It may need to wait for a few seconds for the VFs to be created by the PF driver.
   This could be deferred until there is at least one VF assignment requested.
2. Once VFs are created, domainmgr needs to collect information about them, most importantly their PCI addresses. This can be obtained from the sys filesystem:

   ```
   $ ls -l /sys/class/net/eth0/device/virtfn*
   /sys/class/net/eth0/device/virtfn0 ->../0000:18:02.0
   /sys/class/net/eth0/device/virtfn1 -> ../0000:18:02.1
   /sys/class/net/eth0/device/virtfn2 -> ../0000:18:02.2
   ```

   Note that the PCI address uses BDF notation (bus:device.function). VF ID is used as the function number.
3. The list of VFs would be added to `IOBundle` (bundle itself would be the PF):

   ```
   type IoBundle struct {
     …
     VFs []EthVF
   }

   type EthVF struct {
     Index      uint8
     PciLong    string
     Ifname     string
     UsedByUUID uuid.UUID
     // etc.
   }
   ```

   Note that `IoBundle.KeepInHost` will always be true for SR-IOV PF.
4. domainmgr will manage `IoBundle.VFs`. Most importantly, it will record reservations using `EthVF.UsedbyUUID`, prevent over-assignments (exceeding the number of available VFs), etc.
5. `kvmContext.CreateDomConfig` and `xenContext.CreateDomConfig` will need to change to use the right PCI address for assignment - if the type of the assignment is `PhyIoNetEthVF`, then they will take the PCI address from the VFs list, from the entry with the corresponding `UsedByUUID`.
6. In `doActivateTail()`, domainmgr will additionally configure MAC addresses and VLANs before the domain is started. This can be done using the netlink package for Go. Even though this is a network configuration, it is not related to device connectivity or network instances, thus there is no need to move this to zedrouter or NIM and make things more complicated.

Additionally, we will need to modify `PublishAppInfoToZedCloud` slightly, to properly publish info for VF assignments - namely to publish the PCI and MAC address of the VF, not the underlying PF (as it would be without any changes otherwise).

Note that we do not and cannot collect metrics for directly assigned network adapters.

Lastly, we will need to add a grub option to `grub.cfg`, e.g. `enable_sriov`, which would automatically include `intel_iommu` and add `iommu=pt` into the list of kernel command line parameters.