OpenNebula KVM SR-IOV Driver

With the recent release of an OFED which supports SR-IOV on Infiniband HCAs it is now possible to use verbs from inside a VM. This VMM driver supports these Infiniband HCAs, and any other SR-IOV network device, in OpenNebula.

Author: David Macleod, CHPC
Version: 0.1

Web Links:

Project Website: http://wiki.chpc.ac.za/acelab:opennebula_sr-iov_vmm_driver

Demonstration Video: http://www.youtube.com/watch?v=wB-Z1o2jGaY

OpenNebula Page: http://opennebula.org/software:ecosystem:sr-iov

Prerequisites and Limitations

1. This driver has been developed to support OpenNebula 4.0 and KVM. The driver should be backwards compatible the OpenNebula 3.x.
2. SR-IOV capable hardware and software is required. Before using this driver SR-IOV must be functional on the VM host.
3. Libvirt must run as root. This is required for the hypervisor to attach the SR-IOV virtual function to the VM.
4. Any OpenNebula functions which rely on the "**virsh save**" command (stop, suspend and migrate) require the virtual functions to be hot plugged. This may cause unpredictable behaviour on the OS running in the VM.
5. A virtual bridge is required. The VM will attach interfaces to this bridge but the OS will not use them. They are only required to pass IP address information into the VM. The bridge does not require any external connectivity, it just needs to exist.
6. VF usage tracking is implemented in the */tmp* file system. During a fatal host error the VF usage tracking might become out of sync with actual VF usage. You will have to manually recover.
7. A modified context script is required to decode the SR-IOV interface information inside the VM. An example is given for Infiniband.
8. The maximum number of VMs with SR-IOV interfaces that the host can support is limited by the number of VFs the root device exposes. Usually around 64.
9. To use this driver with Ethernet SR-IOV devices you will need to modify the VM context script and

take into consideration MAC prefixes.

10. Network isolation will not work for any SR-IOV based network interfaces. The SR-IOV interfaces directly access the hardware bypassing the isolation mechanisms.
11. IPv6 has not been tested.

Testing Environment

- CentOS 6.4
- Mellanox OFED 2.0
- libvirt 0.10.2
- OpenNebula 4.0

Installation

1. Extract **kvm-sriov.tar.gz** to */var/lib/one/remotes/vmm/kvm-sriov*
2. Edit */etc/one/oned.conf* and add:

```
VM_MAD = [
    name       = "kvm_sriov",
    executable = "one_vmm_exec",
    arguments  = "-t 15 -r 0 kvm-sriov",
    default    = "vmm_exec/vmm_exec_kvm.conf",
    type       = "kvm" ]
```

3. Use this guide to extract the bus slot and function addresses for the virtual functions:

https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization_Host_Configuration_and_Guest_Installation_Guide/sect-Virtualization_Host_Configuration_and_Guest_Installation_Guide-SR_IOV-How_SR_IOV_Libvirt_Works.html

4. In the */var/lib/one/remotes/vmm/kvm-sriov/vf_maps* directory create a file with the name of the VF's root device, e.g. “**ib0**”. In the file write the bus, slot and function for each VF you want the driver to use. The address must be in hexadecimal. Each line in the file represents a VF, the bus, slot and function addresses are separated by a single space character “ “. There must be **no additional text**, spaces or lines in the table.

   Example, four virtual functions:

/var/lib/one/remotes/vmm/kvm-sriov/vf_maps/ib0:

```
0x07 0x00 0x01
0x07 0x00 0x02
0x07 0x00 0x03
0x07 0x00 0x04
```

5. Edit */var/lib/one/remotes/vmm/kvm-sriov/kvmrc* and set the "**DUMMY_BRIDGE**" and "**DUMMY_MAC_PREFIX**". The "**DUMMY_MAC_PREFIX**" must be different to the prefix which OpenNebula creates. The prefix will be used by the contextualisation scripts to differentiate SR-IOV and libvirt network interfaces.
6. On VM hosts using this driver edit */etc/libvirt/qemu.conf* and change the user and group to "**root**" then restart libvirtd.
7. On the head node restart OpenNebula.
8. Create a virtual network in OpenNebula. The "**Bridge**" field must contain "**sriov_**" prepended to the name of the file that contains the VF mapings, e.g. "**sriov_ib0**"
9. Update the contextualisation scripts for the VMs with the code provided.
10. Create hosts to use the "**kvm_sriov**" driver.

Appendix

Libvirt "save" command

The "**save**" command saves the state of the VM's memory to the host's disk. This operation fails when a VM contains an SR-IOV device because it is passed through the hypervisor and not defined by it. As a result the device's memory state cannot be read, causing the save operation to fail.

To work around this issue the kvm-sriov driver will hot remove SR-IOV interfaces before executing the "**save**" command, and hot attach SR-IOV devices after executing the "**restore**" command.

This driver also supports live migration. Before migrating a VM the driver will attempt to acquire VF locks at the destination. If the migration fails the driver will attempt to reattach the devices at the source.

Dummy Bridge

Mellanox does not support user writeable GUIDs for their HCAs. As a result of this there is no way to pass IP information into the VM from the VF. To work around this the driver attaches an additional libvirt interface and passes the IP information for the SR-IOV interface though on it.

The additional interface is encoded with a different prefix which the context script inside the VM uses to differentiate it from other interfaces. The information is then used to build the IP information for the SR-IOV interface.

VF Usage Tracking

Libvirt does provide a built-in method for tracking VF usage but it does not support the Mellanox OFED 2 drivers. As a result I had to create my own usage tracking mechanism. If you are using this driver with an SR-IOV card which supports VF assignment from a pool then you can use the instructions found below to update the scripts:
http://wiki.libvirt.org/page/Networking#Assignment_from_a_pool_of_SRIOV_VFs_in_a_libvirt_.3Cnetwork.3E_definition

The independent tracking mechanism provided with this driver creates a *vf_interfaces* directory in the host's */tmp* directory. Inside the *vf_interfaces* directory a directory will be created for each SR-IOV root device. Inside the root device's directory files will be created to indicate that a VF is in use. For example, if the first, third and fourth VF are in use you see:

/tmp/vf_interfaces/ib0:

```
0
2
3
```

Ethernet SR-IOV Devices

This driver will always put the libvirt interfaces used for passing IP information at the end of the devices list. This means that if you create a VM with a mix of SR-IOV and libvirt interfaces they will be labeled consecutively and in the same order as displayed in OpenNebula.

Also, you must ensure that Ethernet VFs do not contain either the OpenNebula default MAC prefix or the one specified for identifying SR-IOV devices. To ensure this write a script that checks and sets the MAC addresses of the root device's VFs once the host has booted.

The context script will need to be updated to support the Ethernet devices, however the changes are minimal and simple.

Context Script Modification

Edit the “**00-network**“ and update the **gen_network_configuration()** and **gen_iface_conf()** functions.

```
gen_iface_conf() {
  cat <<EOT
  DEVICE=$DEV
  BOOTPROTO=none
  ONBOOT=yes
  ####################
  # Update this      #
  ####################
  TYPE=$TYPE
  #------------------
  NETMASK=$MASK
  IPADDR=$IP
  EOT
    if [ -n "$GATEWAY" ]; then
      echo "GATEWAY=$GATEWAY"
    fi
    echo ""
}
gen_network_configuration()
{
  IFACES=`get_interfaces`
  for i in $IFACES; do
      MAC=`get_mac $i`
      ###################################
      # Update this                     #
      ###################################
      ib_vf=$(echo "$MAC" | cut -c 1-2)
      ib_pos=$(echo "$MAC" | cut -c 5)
      if  [ "$ib_vf" == "AA" ]; then
        DEV="ib"$ib_pos
        UPCASE_DEV=`upcase $DEV`
        TYPE="Infiniband"
      else
        DEV=`get_dev $i`
        UPCASE_DEV=`upcase $DEV`
        TYPE="Ethernet"
      fi
      #----------------------------------
      IP=$(get_ip)
      NETWORK=$(get_network)
      MASK=$(get_mask)
      GATEWAY=$(get_gateway)
      ###################################
      # Update this                     #
      ###################################
      if  [ "$ib_vf" == "AA" ]; then
        gen_iface_conf > /etc/sysconfig/network-scripts/ifcfg-ib$ib_pos
      else
        gen_iface_conf > /etc/sysconfig/network-scripts/ifcfg-${DEV}
      fi
      #----------------------------------
  done
}
```

Take note of:

1. “$ib_vf” == “AA”. This means that the script is expecting AA as the MAC prefix for SR-IOV devices.
2. gen_iface_conf > */etc/sysconfig/network-scripts/ifcfg-ib$ib_pos*. The ifcfg file is being generated for Infiniband.

License