Advanced Computer Networks
263-3501-00

Network I/O Virtualization

Patrick Stuedi
Spring Semester 2014
Outline

- Last week:
  - Software Defined Networking
  - OpenFlow

- Today:
  - Network I/O Virtualization
  - Paravirtualization
  - SR-IOV
Data Transfer Non-Virtualized

1) Application: syscall, e.g., socket.write()

2) OS driver: issue PCI commands
   - Set up DMA operation

3) NIC:
   - transmit data
   - raise interrupt when done
Virtualization and Hypervisors

VM1
- Guest Application
- Guest Operation System

VM2
- Guest Application
- Guest Operation System

VM3
- Guest Application
- Guest Operation System

Hypervisor

Hardware
Virtualization and Hypervisors

How does network access work?
Option 1: Full Device Emulation

- Guest OS unaware that it is being virtualized
- Hypervisor emulates device at the lowest level
  - Privileged instructions from guest driver trap into hypervisor
- Advantage: no changes to the guest OS required
- Disadvantage:
  - Inefficient
  - Complex
Option 1: Paravirtualization

- Guest OS aware that it is being virtualized
  - Runs special paravirtual device drivers
- Hypervisor cooperates with guest OS through paravirtual interfaces
- Advantage:
  - Better performance
  - Simple
- Disadvantage:
  - Requires changes to the guest OS
Paravirtualization with VirtIO

- VirtIO: I/O virtualization framework for Linux
  - Framework for developing paravirtual drivers
  - Split driver model: front-end and back-end driver
  - APIs for front-end and back-end to communicate
Example: KVM Hypervisor

- Based on Intel VT-x
  - Additional guest execution mode
- I/O at guest OS trap into KVM (VM Exit)
- KVM schedules QEMU process to emulate I/O operation

Starting new guest = starting QEMU process
QEMU process interacts with KVM through ioctl on /dev/kvm to
- Allocated memory for guest
- Start guest
- ...
1) VirtIO-Net driver adds packet to shared VirtIO memory
2) VirtIO-Net driver causes trap into KVM
3) KVM schedules QEMU VirtIO Back-end
4) VirtIO back-end gets packet from shared VirtIO memory and emulates I/O (via system call)
5) KVM resumes guest
Vhost puts VirtIO emulation code into the kernel
- Instead of performing system calls from userspace (QEMU)
Inter-VM communication

VM1
- Guest Application
- Guest Operation System
- Hypervisor
- NIC

VM2
- Guest Application
- Guest Operation System

VM3
- Guest Application
- Guest Operation System
How does inter-VM communication work?
Switch in Hypervisor
Switched Vhost in KVM

- Advantage: low latency
  (1 software copy)
- Disadvantage: uses host CPU cycles
Switch Externally...

...either in

- **External switch:**
  - Simplifies configuration: all switching controlled/configured by the network
    - Latency = 2xDMA + 2hops

- **NIC**
  - Latency = 2xDMA
Where are we?

- Option 1: Full emulation
  - No changes to guest required
  - Complex
  - Inefficient

- Option 2: Paravirtualization
  - Requires special guest drivers
  - Enhanced performance
Where are we?

- **Option 1: Full emulation**
  - No changes to guest required
  - Complex
  - Inefficient

- **Option 2: Paravirtualization**
  - Requires special guest drivers
  - Enhanced performance

Not good enough! Still requires hypervisor involvement, e.g., interrupt relaying
Where are we?

- **Option 1: Full emulation**
  - No changes to guest required
  - Complex
  - Inefficient

- **Option 2: Paravirtualization**
  - Requires special guest drivers
  - Enhanced performance

- **Option 3: Passthrough**
  - Directly assign NIC to VM
  - No hypervisor involvement: best performance

Not good enough! Still requires hypervisor involvement, e.g., interrupt relaying.
Paravirtual vs Passthrough in KVM

VM1
- Application
- Guest OS
  - VirtIO-Net Driver
- tx
- vhost net
- tap

VM2
- Application
- Guest OS
  - Physical Driver
- NIC exclusively assigned to VM2

Hypervisor
- KVM module

Real NIC
Challenges with Passthrough / Direct Assignment

- VM tied to specific NIC hardware
  - Makes VM migration more difficult

- VM driver issues DMA requests using VM addresses
  - Incorrect: VM physical addresses are host virtual addresses (!)
  - Security concern: addresses may belong to other VM
  - Potential solution: let VM translate it's physical addresses to real DMA addresses
    - Still safety problem: exposes driver details to hypervisor, bugs in driver could result in incorrect translations

- Need a different NIC for each VM
Challenges with Passthrough / Direct Assignment

- VM tied to specific NIC hardware
  - Makes VM migration more difficult

- VM driver issues DMA requests using VM addresses
  - Incorrect: VM physical addresses are host virtual addresses (!)
  - Security concern: addresses may belong to other VM
  - Potential solution: let VM translate its physical addresses to real DMA addresses
    - Still safety problem: exposes driver details to hypervisor, bugs in driver could result in incorrect translations
  - Solution: Use an IOMMU to translate/validate DMA requests from the device

- Need a different NIC for each VM
  - Solution: SR-IOV, emulate multiple NICs at hardware level
Memory Address Terminology

- **Virtual Address**
  - Address in some virtual address space in a process running in the guest OS

- **Physical Address:**
  - Hardware address as seen by the guest OS, i.e., physical address in the virtual machine

- **Machine address:**
  - Real hardware address on the physical machine as seen by the Hypervisor
IOMMU
IOMMU

VMM programs IOMMU with VM-physical to machine address translations

PCIe function (e.g. NIC)
IOMMU

VMM programs IOMMU with VM-physical to machine address translations

IOMMU

Memory controller

Main memory

Guest OS programs NIC with VM-physical address of DMA

PCIe function (e.g. NIC)
IOMMU

- PCIe function (e.g. NIC)
- NIC issues a DMA request to VM physical memory
- Guest OS programs NIC with VM-physical address of DMA
- VMM programs IOMMU with VM-physical to machine address translations
- Memory controller
- Main memory
IOMMU

PCIe function (e.g. NIC)

IOMMU checks and translates to machine (real) address for transfer

VMM programs IOMMU with VM-physical to machine address translations

Guest OS programs NIC with VM-physical address of DMA

NIC issues a DMA request to VM physical memory

Memory controller

Main memory

PCle function (e.g. NIC)
IOMMU

VMM programs IOMMU with VM-physical to machine address translations

IOMMU checks and translates to machine (real) address for transfer

Memory controller

Main memory

Guest OS programs NIC with VM-physical address of DMA

NIC issues a DMA request to VM physical memory

Memory controller accesses memory

PCIe function (e.g. NIC)

Memory controller accesses memory
SR-IOV

- Single-Root I/O Virtualization
- Key idea: dynamically create new “PCI devices”
  - Physical Function (PF): original device, full functionality
  - Virtual Function (VF): extra device, limited functionality
  - VFs created/destroyed via PF registers
- For Networking:
  - Partitions a network card's resources
  - With direct assignment can implement passthrough
SR-IOV in Action

Guest Application

Guest OS

Physical Driver

Guest Application

Guest OS

Physical Driver

Guest Application

Guest OS

Physical Driver

Hypervisor

Physical Driver

IOMMU

PCI

Virtual Function

Virtual Function

Virtual Function

Physical Function

Virtual Ethernet Bridge/Switch

SR-IOV NIC
References

- **I/O Virtualization**, Mendel Rosenblum, ACM Queue, January 2012