I/O Architectures for Virtualization

Mallik Mahalingam

R &D - Networking

VMWORLD 2006

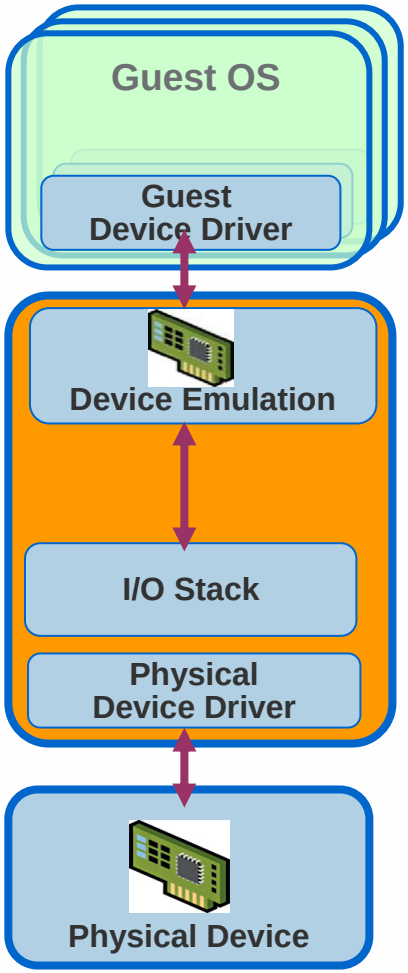
Motivation

- I/O Virtualization is an essential component in the virtualization framework
- Software based I/O Virtualization provides
  - Rich set of features
    - I/O Sharing, Consolidation, Security, Isolation, Mobility
  - Simplified management
    - Transparent Teaming and Failover
- High I/O performance is desirable for enterprise class applications
- Can we accelerate I/O performance without compromising virtualization benefits?

Outline

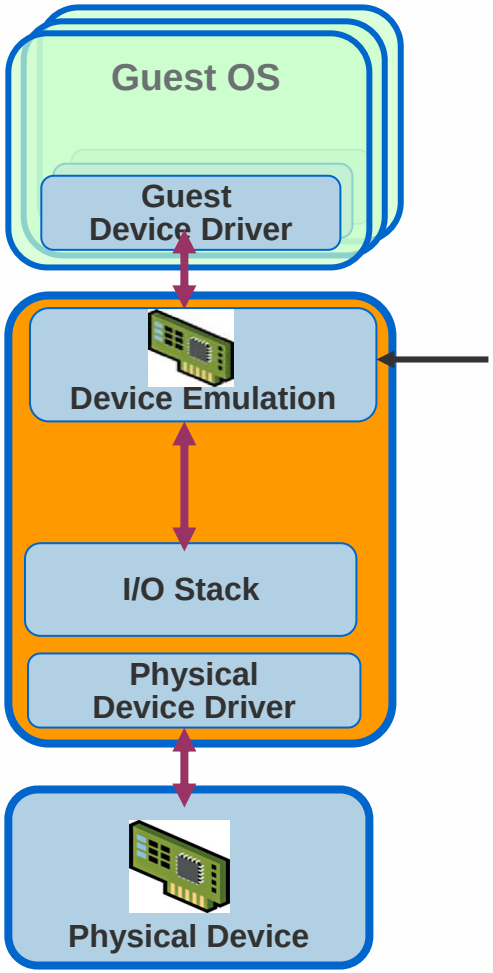
- VMware I/O Virtualization Architecture
- Hardware/Software Assists for Virtualized I/O
- Passthrough I/O
- Conclusion

I/O Virtualization Architecture



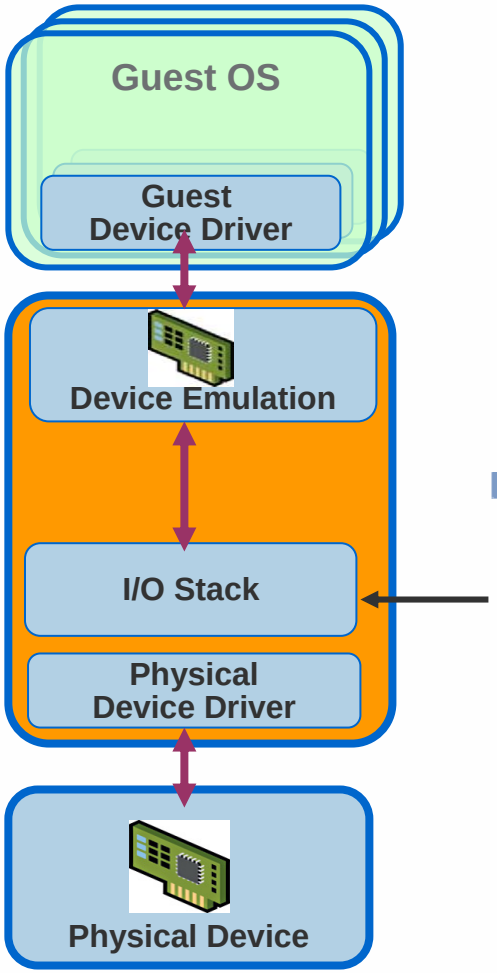
- I/O Virtualization architecture consists of
  - Guest driver
  - Virtual device
  - Communication mechanism between virtual device and virtualization stack
  - Virtualization I/O stack
  - Physical device driver
  - Real device

I/O Virtualization Architecture (contd.)



- Virtual device
  - model a real device
    - *e.g.,* Intel *e1000, LSI mptscsi*
  - model a simple virtualization friendly device
    - *e.g., VMware vmxnet*

I/O Virtualization Architecture (contd.)

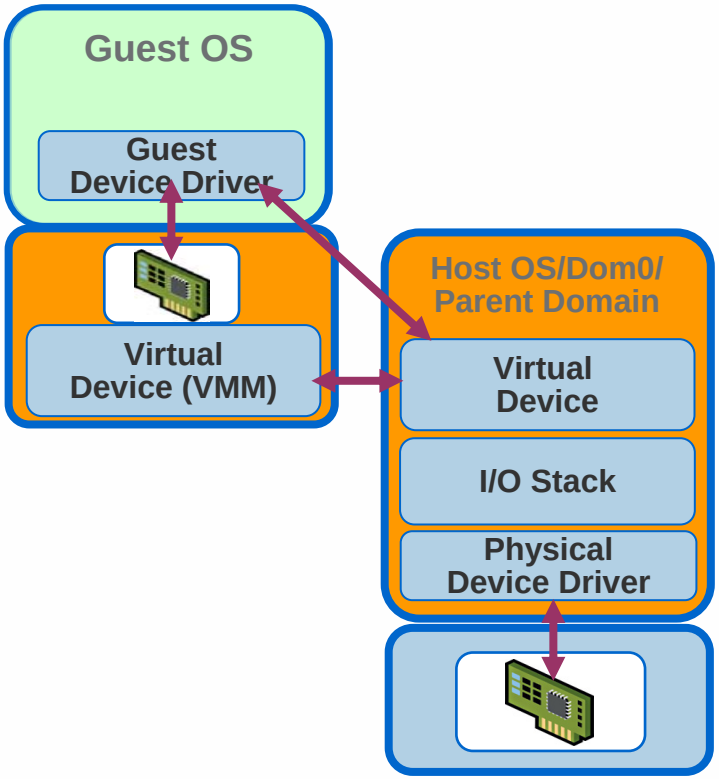


- Virtualization I/O stack
  - translates guest I/O addresses to host addresses
  - handles inter VM communication
  - multiplexes I/O requests from/to the physical device
  - provides enterprise-class I/O features

I/O Virtualization Implementations

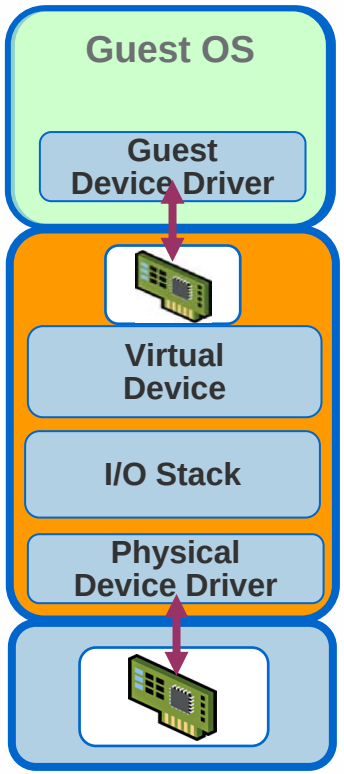
Virtualized I/O

Hosted or Split



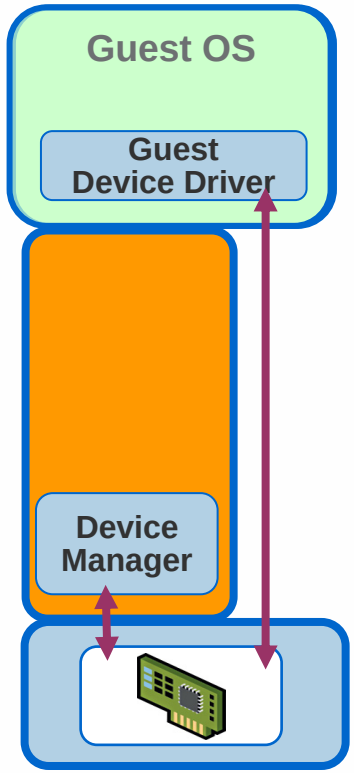
VMware Workstation, VMware Server, VMware ESX Server
Microsoft Viridian & Virtual Server, Xen

Hypervisor Direct



VMware ESX Server
(storage and network)

Passthrough I/O



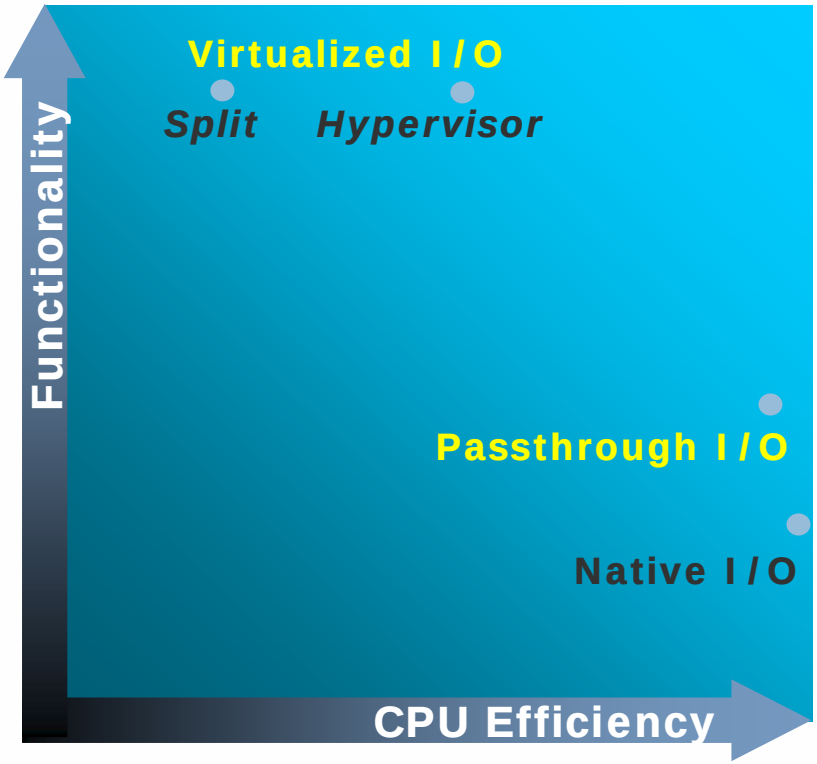
A Future Option

Comparing I/O Virtualization Approaches

- Virtual Device Model
  - Emulation of well-known hardware
  - Virtualization-friendly device for I/O intensive workloads
  - VMware, Microsoft, Xen accommodate both models
- Messaging/Signaling Mechanism
  - VMware: virtual PCI bus and IPC or syscall
  - Xen/Microsoft: XenBus/VMBus, IPC
- Virtualization Stack
  - VMware: runs in ESX VMkernel
  - Xen and Microsoft: Dom0, Parent Domain

Virtualized I/O vs. Passthrough I/O

Tradeoff between functionality and efficiency



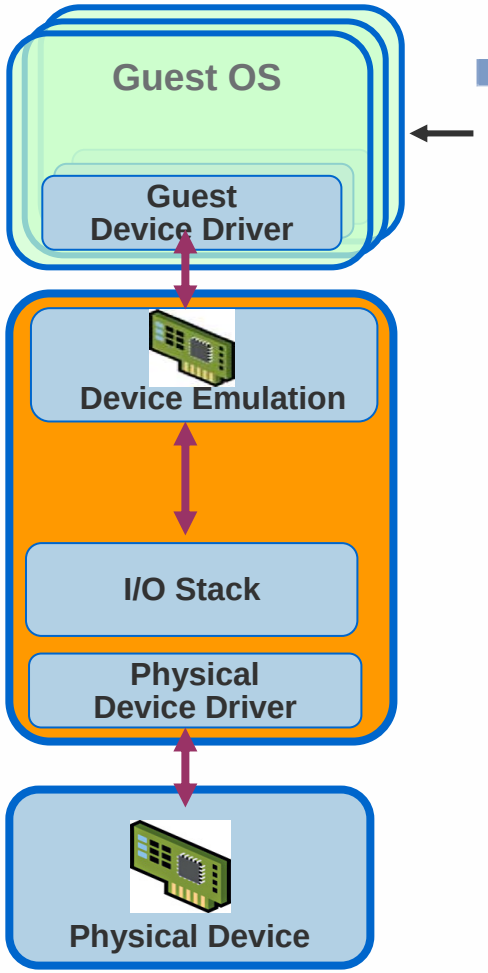
- Virtualized I/O provides rich virtualization functionality
- Passthrough I/O minimizes CPU utilization

What assists can be built for each?

Virtualized I/O

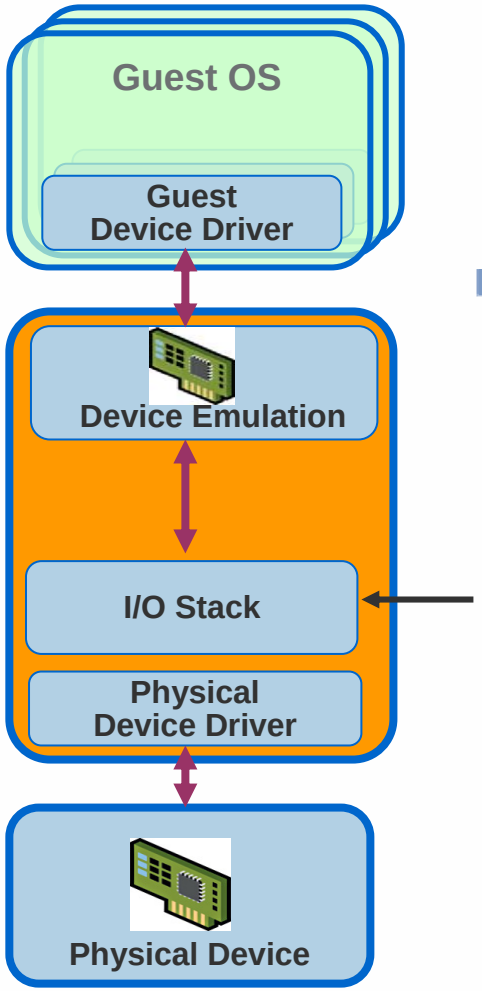
- Fully supports data center virtualization
  - Hardware independence
  - Transparent VMotion
    - Virtual device checkpointing
    - Write protect pre-copied guest memory
  - Memory over-commitment
  - I/O device sharing
  - Advanced features
    - Security, Firewall, Path Failover, etc.
- Challenge is reducing CPU utilization

Improving Virtualized I/O



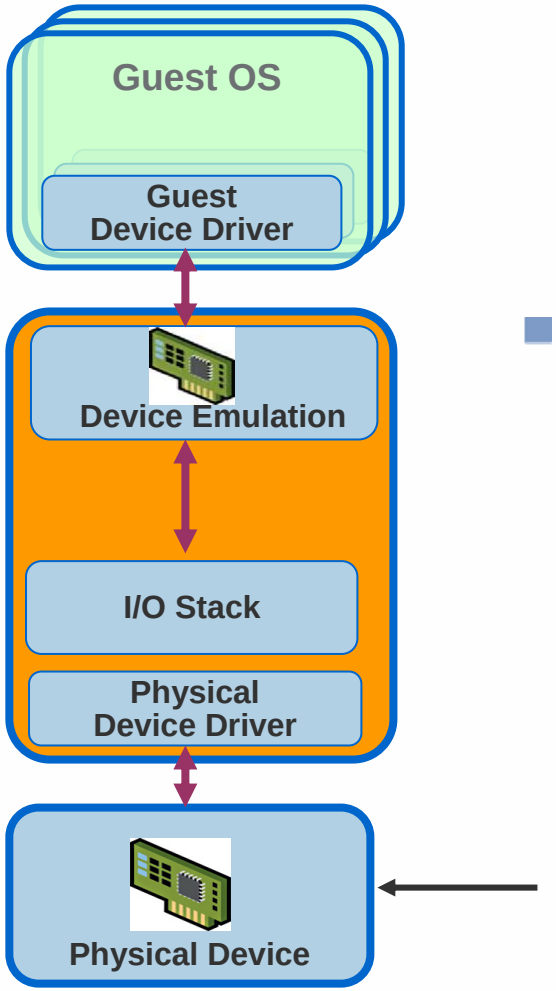
- Accelerate guest code
  - Faster CPU and MMU virtualization
  - Support for improving device virtualization
    - e.g. improving MMIO, virtual interrupt delivery
  - Simplified device models
  - Para-virtualized drivers

Improving Virtualized I/O (contd.)



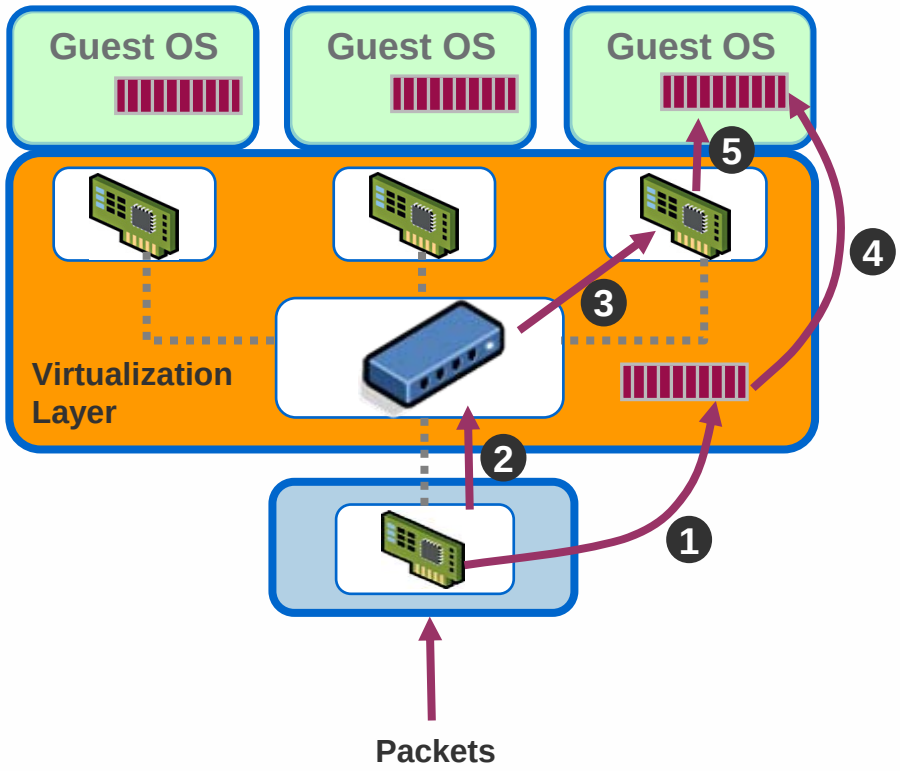
- Accelerate virtualization stack with HW assists
  - Intel I/OAT
    - Dedicated DMA engine for memory-memory copy
  - TCP/IP Offload Engines (TOE)
    - Accelerate VMkernel TCP processing
  - Remote DMA
    - Accelerate VMotion, NFS, iSCSI

Improving Virtualized I/O (contd.)



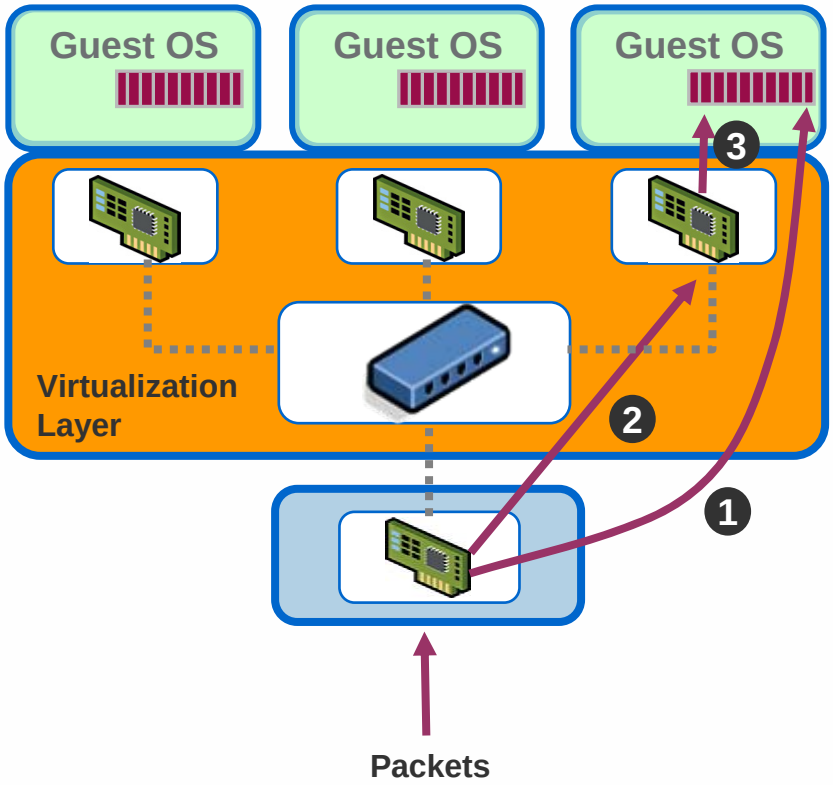
- Physical NIC Acceleration
  - Spread the packet processing to multiple cores
    - Multiple receive queues
    - Configure queue based on MAC addresses
    - Assign a receive queue to each virtual NIC
    - Map receive buffers to guest memory - avoids a copy
    - Interrupt per queue (MSI/MSI-X) – steer to idle or optimal processor core

Network Receive Processing (today)



1. DMA packet into receive buffers owned by VMkernel
2. Raise physical Interrupt
3. Parse packet to find destination VM
4. Copy packet into guest receive queue
5. Raise virtual NIC interrupt

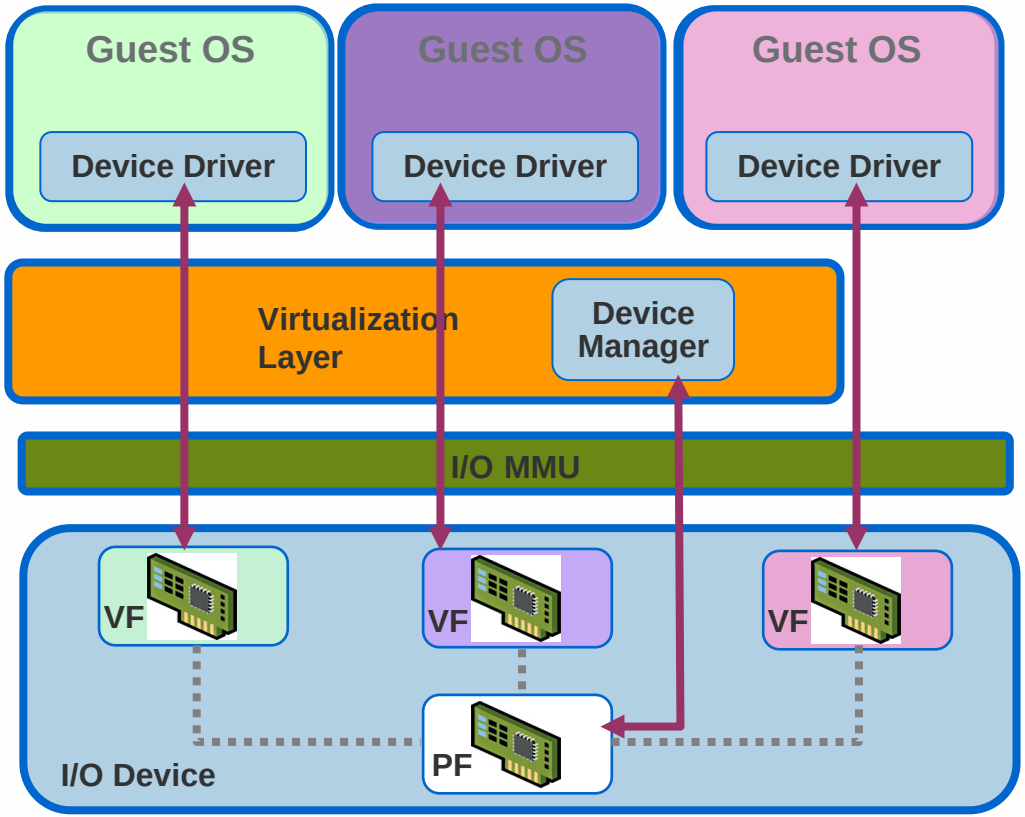
Receive With Physical NIC Acceleration



1. DMA packet into guest receive queue
2. Raise NIC interrupt at guest CPU
3. Raise virtual NIC interrupt

Passthrough I/O

- Guest drives device directly
  - Use case: I/O Appliances, High performance VMs



PF = Physical Function, VF = Virtual Function

Requires :

- I/O MMU for DMA Address Translation and protection (Intel® VT-d, AMD I/O MMU)
- Partitionable I/O device for sharing (PCI-SIG IOV SR/MR specification)

Challenges with Passthrough I/O

- Transparent VMotion
  - No simple way to checkpoint device state
  - Inability to write protect pre-copied guest memory
- Hardware independence
  - Checkpoint is probably limited to same HW vendor/model
  - HW errata/workarounds in the driver
- Memory over-commitment
  - No visibility of DMAs to guest memory
- VM Management
  - IHV driver for different vendors, device types, revisions, guest OSes
- Isolation/Security
  - E.g. Promiscuous Mode, SRC MAC address spoofing

Special VMs/Appliances may not have some of these requirements

Possible Solutions – Guest Software

- Teaming solution - Team real device and virtualized device
  - Mostly limited to networking and storage
  - Failover to emulation (standby) before VMotion
  - Re-enabling Passthrough after VM Migration
    - Fallback to primary in the team (optional driver re-init)
    - Reload new IHV driver (for different HW) & join the team
- IHV-Plugin - Emulation driver provides wrapper for IHV driver to allow performance critical operations
  - Tx, Rx, interrupt proxying, quiesce, restart
  - “Small” IHV driver code and “easy” to manage
  - Dynamic loading of plug-ins (for different HW)

Possible solutions – Ideal HW

- Passthrough performance critical operations only
  - Tx, Rx, Interrupt cause
- Uniform register layout, descriptors, scatter gather elements
- Easy device checkpoint
  - Tx/Rx Rings are maintained in the guest memory
  - Checkpoint HW Producer & Consumer ring Index
- Single driver works with real HW as well as with emulation
  - Similar to I2O but solves different problems

Virtualized & Passthrough I/O - Comparison

	Virtualized I/O (Including Physical NIC Acceleration)	Passthrough I/O	
		Guest SW Approaches	Ideal HW
Support for VMotion	Transparent	Needs Guest SW support	Transparent
Memory Over-commitment	Transparent	Use DMA API	Use DMA API
Hardware Independence	Completely Independent	Issues with VMotion	No backward compatibility
Performance	Limited by CPU	Best case	Best case
VM Management	Easy to Manage	Needs Device specific drivers	Easy to Manage
IHV Differentiation	Limited by Emulation	More opportunities	Limited by the Ideal HW Interface

Conclusions

- Use Virtualized I/O when Abstraction and Mobility is needed
- Virtualized I/O offers rich set of features
  - Performance at the cost of CPU utilization
  - Physical NIC Acceleration offers good scaling by fanning out processing to different CPU cores
- Passthrough I/O gives best performance
  - Compromises some of the virtualization features/benefits
  - Almost all the approaches discussed have limitations
  - Use Passthrough I/O for special scenarios until a more “complete” solution is developed.

Call To Action

- Develop Physical NIC Acceleration Hardware to improve Virtualized I/O performance
- For Passthrough I/O -
  - Develop common HW interface for performance-critical operations
  - Develop I/O SW Interface for
    - Guest DMA Pin/Unpin operations
    - VMotion (Teaming/Guest IHV plug-ins)
  - Develop Hypervisor control functions in HW for security and management

Presentation Download

Please remember to complete your
session evaluation form
and return it to the room monitors
as you exit the session

The presentation for this session can be downloaded at
http://www.vmware.com/vmtn/vmworld/sessions/

Enter the following to download (case-sensitive):

Username: cbv_rep
Password: cbvfor9v9r

Some or all of the features in this document may be representative of feature areas under development. Feature commitments must not be included in contracts, purchase orders, or sales agreements of any kind. Technical feasibility and market demand will affect final delivery.

VMWORLD 2006