SR-IOV Networking in Xen: Architecture, Design and Implementation

Yaozu Dong, Zhao Yu and Greg Rose

Abstract. SR-IOV capable network devices offer the benefits of direct I/O throughput and reduced CPU utilization while greatly increasing the scalability and sharing capabilities of the device. SR-IOV allows the benefits of the paravirtualized driver's throughput increase and additional CPU usage reductions in HVMs (Hardware Virtual Machines). SR-IOV uses direct I/O assignment of a network device to multiple VMs, maximizing the potential for using the full bandwidth capabilities of the network device, as well as enabling unmodified guest OS based device drivers which will work for different underlying VMMs.

Drawing on our recent experience in developing an SR-IOV capable networking solution for the Xen hypervisor we discuss the system level requirements and techniques for SR-IOV enablement on the platform. We discuss PCI configuration considerations, direct MMIO, interrupt handling and DMA into an HVM using an IOMMU (I/O Memory Management Unit). We then explain the architectural, design and implementation considerations for SR-IOV networking in Xen[1] in which the Physical Function has a driver running in the driver domain that serves as a "master" and each Virtual Function exposed to a guest VM has its own virtual driver.

1 Introduction

I/O is an important part of any computing platform, including virtual machines running on top of a virtual machine monitor (VMM) such as Xen[14][11]. As has been noted many times before it's possible to have all the CPU cycles needed but if there is no data for the CPU to act upon then CPU resources are either wasted on idle cycles, or in the case of software emulation of I/O devices, many CPU cycles are expended on I/O itself, reducing the amount of CPU available for processing of the data presented. For this reason much research and development has been focused on methods for reducing CPU usage for purposes of I/O while increasing the amount of data that can be presented to the VM for processing, i.e. "useful work".

Various strategies and techniques have been employed to this end, especially with respects to network I/O due to the high-bandwidth requirements. Paravirtualization techniques [5] and its enhancement utilizing network multiple queues [12] and Solarflare's SolarStorm technology [1] are used to accelerate the presentation of data to the VM, both reducing CPU overhead and increasing data throughput.

In October of 2007 the PCI-SIG released the SR-IOV specification [2] which detailed how PCIe compliant I/O device HW vendors can share a single I/O device among many VMs. In concert with other virtualization technologies such as Intel® VT-x [3] and VT-d [4] and AMD® AMD-V and the

AMD Integrated Memory Controller [14] it has now become possible to greatly enhance the I/O capabilities of HVMs in a manner that are scalable, fast and highly resource efficient.

Realizing the benefits of PCI-SIG SR-IOV involves integrating and making use of the capabilities of the entire platform and OS. A SR-IOV implementation is dependent upon an array of enabling technologies, all of which are detailed in the following sections. Network device drivers that take advantage of SR-IOV capabilities require specific changes and additional capabilities as well as new communication paths between the physical device and the virtual devices it supports.

2 MSI for direct I/O

Direct I/O access (also known as device pass through) is supported in the Xen driver domain to contain driver failure [6], and is further extended to the user domain for performance when PCIe devices are assigned. Guest MMIO (Memory Mapped IO) of pass through device is directly translated in shadow page table or direct page table to avoid CPU intervention for performance.

One of the biggest challenges in direct I/O is interrupt sharing when working in legacy pin based interrupt signaling mechanism. Hypervisor, at the time physical interrupt fires, must inject a virtual interrupt to all guests whose pass through device shares interrupt. But interrupt sharing imposes more challenges in virtualization system for inter guest isolation both in performance and robustness. A group of guest sharing interrupt is depending on a set of devices, which may malfunction to generate unexpected interrupt storm if the owner guest is compromised, or mislead host to generate malicious interrupt clearance request such as EOI (End of Interrupt).

MSI

We extended Xen direct I/O to support message signaled interrupt (MSI) and its extension (MSI-X) [7] to avoid interrupt sharing. MSI and MSI-X mechanism provide software ability to program interrupt vector for individual MSI or MSI-X interrupt source. Dom0 cooperates with hypervisor to manage the physical interrupt and programs each MSI/MSI-X interrupt vector with a dedicated vector allocated from hypervisor. In the meantime, MSI and MSI-X support is required by SR-IOV devices where INTx is not implemented.

Virtual MSI

A virtual MSI/MSI-X model is introduced in the user level device model to support guest MSI/MSI-X when host has MSI/MSI-X capability. Full hardware MSI and MSI-X capabilities are presented to guest, in the meantime guest per vector mask and unmask operation is directly propagated to host while others are emulated. MSI/MSI-X vectors programmed to device are remapped since host side vectors must come from Xen to maintain host side identical.

3 SR-IOV Network Device Design

Single Root I/O Virtualization and Sharing Specification (SR-IOV) defines extensions to the PCI Express (PCIe) specification suite to enable multiple system images or guests to directly access subset portions of physical I/O resources for performance data movement and to natively share underlying hardware resources. An SR-IOV device presents single or multiple Physical Functions (PFs) which are standard PCIe functions. Each PF can have zero or more Virtual Functions (VFs) which is a

"light-weight" PCIe function that has enough resource for major data movement, as well as a unique requester identifier (RID) to index the IOMMU page table for address translation.

3.1 Architecture

Like a PCIe pass through device, a VF residing in an SR-IOV device can be assigned to a guest with direct I/O access for performance data movement. A Virtual Function Device Driver (VDD) running in the guest, as shown in Figure 1, is VMM agnostic, and thus be able to run on other VMMs as well. The IOMMU, managed by Xen, will remap VF driver fed guest physical address to machine physical address utilizing the VF RID as the IO page table index.

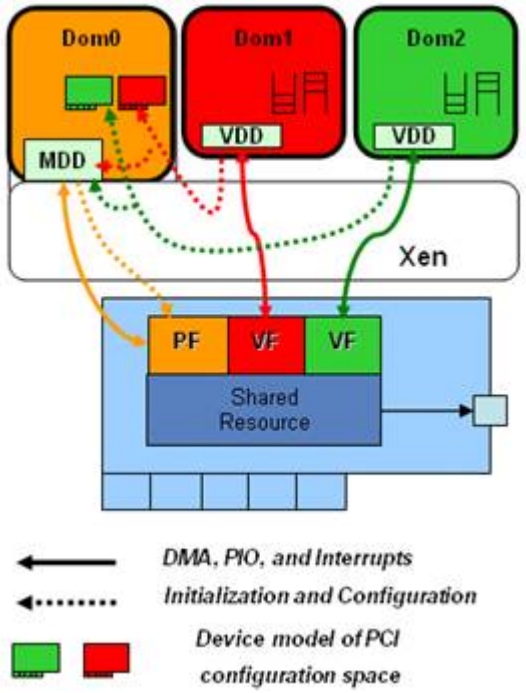


Figure 1: Xen SR-IOV Networking Architecture

VF configuration space access from VDD is trap and emulated by VMM to present guest a standard PCIe device to reuse existing PCI subsystem for discovery, initialization and configuration, which simplifies guest OS and device driver. As shown in Figure 1, user level device mode [11][12] in dom0 emulates guest configuration space access, but some of guest access may be forwarded to HW directly if targets dedicated resource.

PF driver, or Master Device Driver (MDD), running in driver domain (dom0 in Xen) controls underlying shared resources for all associated VFs, i.e. virtualizes non performance critical resources for VF (see 3.3 for details). SR-IOV network device shares physical resources and network bandwidth among VFs. The PF is designed to manage the sharing and coordinate between VFs. MDD in dom0 has the ability to directly access PF run time resources and configuration resources, as well as the ability to manage and configure VFs. Administrator tools such as control panel in dom0 manage PF behavior to set the number of VFs, globally enable or disable VFs, as well as network specific configurations such as MAC address and VLAN setting for each VF.

3.1.1 VF assignment

The VF, appearing as a "light-weight" PCIe function when enabled in hardware, doesn't respond to ordinary PCI bus scan for vendor ID and device ID used in OS to discover PCI function, and thus can't be enumerated by dom0 Linux. Xen SR-IOV networking architecture utilizes Linux PCI hot plug API to dynamically add VFs to dom0 Linux when VFs are enabled and vice versa when VFs are disabled so that existing direct I/O architecture works for VF too.

Xen SR-IOV networking architecture implements SR-IOV management APIs in the dom0 Linux kernel as a generic module rather than in the MDD itself, which simplifies potentially thousands of MDD implementations. In the meantime MDDs must be notified when the configuration is changed so that PF drivers can respond to the event.

3.1.2 Control Interface

PCI sysfs is extended for the PF to provide a user/kernel control interface. An additional "iov" inode is introduced to PF sysfs directory as shown in Table 1. Network SR-IOV devices, such as Intel® 82576 Gigabit Ethernet Controller, require administrator tools to set MAC address for each VF as well as VLAN, but this must be set by PF for security reason. Xen SR-IOV architecture employs "iov/n" (n start from 0) 2nd level inode for per VF configurations. The architecture provides APIs for the MDD to create and query device specific inodes in addition to nodes modification notification for MDD to respond immediately.

inode	Usage
iov	Subdirectory for all iov configurations
iov/enable	1/0 for global VF enable / disable
iov/numvfs	Number of VFs that are available
iov/n	Per VF configuration subdirectory

Table 1: SR-IOV extension inodes in PF sysfs

Kernel / MDD communication APIs are introduced for the MDD to respond to a configuration change. A Virtual Function is a set of physical resources dynamically collected when it is enabled, it must be correctly configured in advance before it can function. All the resources available are initially owned by the Physical Function, but some of them will be transferred to VFs when VFs are enabled such as queue pairs. Pre-notification and post-notification events are introduced so that MDDs can respond to IOV configuration changes before or after the hardware settings really take effect. The same applies to the per VF MAC and VLAN configuration change.

Notification API

- Typedef int (*vf_notification_fn) (struct pci_dev *dev, unsigned int event_mask);

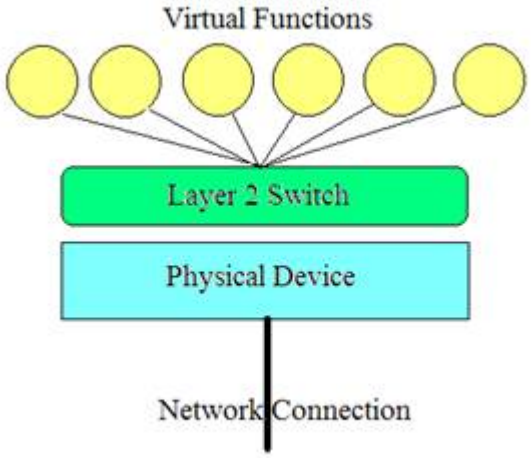
  Bit definition of event_mask:

  - b31: 0/1 means pre/post event
  - b18-16: Event of enable, disable and configuration
  - b7-b0: VFn of event, or totalVFS

3.2 PF Resource Management, Security and Configuration

Enabling SR-IOV features in the network device will inevitably lead to changes in the number and types of resources available to the MDD. Upon loading the MDD there can be no assumption as to whether the SR-IOV capabilities of the device have been enabled or not. The software must be able to inspect its configuration and determine if SR-IOV capabilities are enabled and if they are enabled, how many virtual functions have been enabled. The MDD will also add security and policy enforcement features to monitor VFs for harmful or malicious behavior as well as configuring anti-spoof features in the HW. The ability of the MDD to monitor and control the VFs is one of the more attractive features of the SR-IOV I/O sharing model.

Another consideration for the MDD is that implicit in the sharing of its network resources among several virtual function devices is that at some sort of layer 2 switching will be required to make sure that packets incoming from either the network or from other virtual function devices are properly routed and, in the case of broadcast or multicast packets, replicated to each of the physical and virtual functions as needed.



3.2.1 PF Resource Management

A SR-IOV capable network device such as the Intel® 82576 Gigabit Ethernet Controller [9] is equipped with a set number of resources as part of its HW design. When the SR-IOV capabilities are enabled for the device, resources that were available to the PF are now distributed to the VFs. Depending on the HW design and the number of VFs enabled the PF driver may have all or some

of its resources reserved to the VFs. When the MDD loads it will need to examine the SR-IOV configuration and number of VFs enabled to make a decision as to what resources are available and how they may be best used. As an example, while the PF may have multi-queue capabilities such as RSS, those capabilities may not be available depending on how many VFs are enabled and how many resources they consume.

3.2.2 PF MDD Security Considerations

By the very nature of their architecture and design Virtual Function devices provide a better environment for secure computing. Paravirtualized drivers running in a guest OS with direct access to HW resources to an entire PCI device, without a monitoring entity such as a Physical Function device are inherently less secure. Since there is a HW enforced hierarchy in SR-IOV in which the MDD on the physical function has full visibility of all HW resources of the device while the VDD on the virtual function has visibility only into it's own subset of the device resources, it is possible to create secure policies and safeguards that the MDD will be able to enforce upon the VF device driver. Depending on HW and SW design considerations it would be possible to monitor and enforce policies concerning VF device bandwidth usage, interrupt throttling, congestion control, broadcast/multicast storms and a number of conditions that might be the result of a rogue agent in a guest attempting to use the VF device for malicious purposes.

As a security precaution the MDD must be prepared for the eventuality that the VDD may have been taken over by a rogue agent or that the VDD itself has been replaced by a rogue driver with malicious intent. The rogue agent or driver may attempt to cause an interrupt storm or overwhelm the MDD with what amounts to a denial of service attack by sending a huge volume of spurious or undefined messages. Two primary approaches to handling this type of attack would be to:

a) Drop undefined or repetitious messages and disable the VF when such activity is detected as well as notifying management SW in the VMM that the guest OS to which the VF was assigned has suffered a security breach.

b) Employ heuristics to detect when a malicious agent in the guest OS is attempting to cause an interrupt storm by repeatedly ringing the doorbell.

c) Use the anti-spoof capabilities of the 82576 to prevent MAC and VLAN spoofing, detect which VM is engaging in this behavior and shut it down.

Care in the design and implementation of the MDD should be sufficient to handle such security threats.

3.2.3 VF Configuration

The SR-IOV network PF driver has unique requirements that must be considered during the design and implementation phase. Standard Ethernet network drivers already have to support a number of common network configurations such as multicast addresses and VLAN setup along with their related hardware supported offloads. Other cases which are not universally supported yet still commonly used involve support for interrupt throttling, TCP offloads, jumbo frames, and others.

When the SR-IOV capabilities for distributing resources to virtual functions come into consideration, the complexity of these configuration requests becomes more complicated. The PF driver must handle multiple, and sometimes conflicting request, for configuration from the virtual function device

drivers. Good hardware design will help the PF driver manage this complexity by allowing it to configure resources and offloads effectively on a per VF basis.

3.3 PF/VF Communication

The MDD and VDD will require a method of communication between them. One of the SR-IOV hardware design rules is to implement only those performance critical resources in VF side, while leaving non-performance critical resources virtualized by an I/O virtualization intermediary [2] (dom0 and hypervisor in Xen). PF/VF communications are designed for MDD and VDD to cooperatively share the global network resources.

There are two approaches to handling communications between the PF and the VF. One is to implement a private HW based interface that is specific to the device. The other would be to use a communication channel provided by the VMM vendors.

The primary advantage to using the private HW based interface is that there would be a single, consistent interface for communication between the PF/VF. The SW based channel would have the advantage of providing a consistent interface that all drivers could use for common tasks and event communications. Guest configuration space access which is emulated by device model could also be treated as a kind of PF/VF communication channel, with standard interface per PCI specification,

At this time no VMM vendor has implemented or specified a SW communication channel between the PF and the VF. While the SW based communication channel is likely the ideal method of communication, until such a communication channel is specified and implemented among the various VMM vendors it will be necessary for those who wish to develop and release SR-IOV products to use a private HW based communication method.

Intel Corporation has implemented such a HW based communication method in its SR-IOV capable network devices. It is implemented as a simple mailbox and doorbell system. The MDD or VDD establishes ownership of the shared mailbox SRAM through a handshaking mechanism and then writes a message to the mailbox. It then “rings the doorbell” which will interrupt the PF or VF as the case may be, sending notification that a message is ready for consumption. The MDD or VDD will consume the message, take appropriate action and then set a bit in a shared register indicating acknowledgement of message reception.

3.4 Network Sharing

Most network configurations are shared among VFs such as security, synchronization, etc., and thus must use PF/VF communication to pass guest side request on to the MDD for virtualization by the VDD. For example, the VDD may have a request from the guest OS to set up a list of multicast addresses. In this case the VDD would send a request to the MDD with the attached list of multicast addresses. The MDD would examine the list of multicast addresses from the guest, determine if there are duplicates from other guests and then add the new multicasts to its multicast address filter entries. The MDD will use its layer 2 switch capabilities to direct the duplicate multicast addresses that are already in the filters to the requesting VF.

Other common network configuration requests from the guest OS to the VDD would be VLAN setup, enabling jumbo frames, enabling or disabling various offloads such as TSO or check-summing and others. These requests will be handled via the PF/VF communications discussed in section 3.3.

3.5 Network Event Notifications

Physical network event happening in MDD side will have to be forwarded to notify each VDD for the shared resource status change, as another part of VDD/MDD cooperative virtualization process. These include but are not necessarily limited to:

- impending global device reset
- link status change
- impending driver removal and, if appropriate, a matching MDD re-insertion.

4 Performance analysis

4.1 Virtual Functions Benefit from Direct I/O

The VF device has direct access to its own registers and IOMMU technology allows translation of guest physical addresses (GPA) into host physical addresses for direct I/O the VF will achieve near native (or bare metal) performance running in a guest OS. Each VM using a VF device will get the benefits of higher throughput with lower CPU utilization compared to the standard software emulated NIC. Another significant benefit of a Virtual Function device using Direct I/O is that register reads and writes do not have to be trapped and emulated. The CPU paging features are used to directly map the VF device MMIO space into the guest. Trapping and emulating register reads and writes are very expensive in terms of CPU utilization and extra task switches.

4.2 Virtual Functions Benefit from Improved Interrupt Remapping

Newer IOMMUs that are being developed right now and are soon to be released into the market have improved interrupt remapping technology that will reduce latency from the time the HW interrupt is triggered to when the actual interrupt handler in the guest is invoked. Interrupt latency is one of the primary bottlenecks in a virtualized environment. By using streamlined MSI-X interrupt technology with IOMMU interrupt remapping latency will be reduced which will lead to even more performance gains.

4.3 Virtual Functions Benefit from Improved Scalability

Directly assigned NICs provide the benefits of native, bare metal performance to a guest VM under Xen, however the problem with this approach is that the entire bandwidth of a physical Ethernet port can only be utilized by a single VM. In most cases this defeats the original purpose of virtualization,

i.e. sharing of machine and I/O resources to achieve maximum utilization of the end user's expensive hardware investment. Unless the VM is continuously running a high bandwidth consumption application it is likely that the resources of the directly assigned or paravirtualized NIC are under utilized and could be used by other VMs. The solution to this problem is PCI SIG SR-IOV virtual devices.

A single virtual device can provide near native bandwidth performance to a VM or multiple virtual devices can provide aggregate near native bandwidth performance to multiple VMs. In this manner the SR-IOV capable network device is much more scalable and a far better investment than a network device without this capability. Additional flexibility in the deployment and allocation of resources to VMs is an advantage over directly assigned PCI devices which actually place such a burden on system administrators that they are only used under the most demanding circumstances.

5 Conclusion

SR-IOV enabled network devices provide a high degree of scalability in a virtualized environment as well as improved I/O throughput and reduced CPU utilization. Even in cases where only a single VF is allocated for the physical device and dedicated to a single VM the increased security capability makes the SR-IOV solution superior to the traditional direct assignment of the entire physical PCI device to a VM as the MDD is capable of monitoring the VF(s) for security violations, preventing spoofing and in cases where a rogue driver has been installed in the VM it can even shut down the device.

References

1. Getting 10 Gb/s from Xen: Safe and Fast Device Access from Unprivileged Domains. http://www.solarflare.com/technology/documents/Getting_10Gbps_from_Xen_cognidox.pdf
2. PCI SIG: PCI-SIG Single Root I/O Virtualization. http://www.pcisig.com/specifications/iov/single_root/
3. Intel Virtualization Technology. http://www.intel.com/technology/itj/2006/v10i3/1-hardware/6-vt-x-vt-i-solutions.htm
4. Intel Corporation: Intel Virtualization Technology for Directed I/O. http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art02.pdf
5. FRASER, K., HAND, S., NEUGEBAUER, R., PRATT, I., WARFIELD, A., AND WILLIAMS, M. Safe hardware access with the Xen virtual machine monitor. In 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS) (October 2004). http://www.cl.cam.ac.uk/research/srg/netos/papers/2004-oasis-ngio.pdf
6. Paul Willmann, Scott Rixner, and Alan L. Cox, Protection Strategies for Direct Access to Virtualized I/O Devices. In USENIX Annual Technical Conference (USENIX 08), Boston, MA, June 2008. http://www.usenix.org/events/usenix08/tech/full_papers/willmann/willmann_html/index.html
7. PCI SIG: PCI-SIG Local Bus Specification 3.0. http://www.pcisig.com/members/downloads/specifications/conventional/PCI_LB3.0-2-6-04.pdf
8. Xen.org. http://www.xen.org/

9. Intel® 82576 Gigabit Ethernet
Controller. http://download.intel.com/design/network/ProdBrf/320025.pdf
10. I. Pratt, K. Fraser, S.Hand, C. Limpach, A. Warfield, Dan. J Magenheimer, Jun Nakajima, and Asit Mallick. Xen 3.0 and the art of virtualization. In Proceedings of the 2005 Ottawa Linux Symposium, pages 65–77, July 2005.
11. DONG, Y., LI, S., MALLICK, A., NAKAJIMA, J., TIAN, K., XU, X., YANG, F., AND YU, W. Extending Xen with Intel Virtualization Technology. In Intel Technology Journal (August 2006), vol. 10.
12. Jose Renato Santos, Yoshio Turner, G.(John) Janakiraman, Ian Pratt. Bridging the Gap between Software and Hardware Techniques for I/O Virtualization, In USENIX Annual Technical Conference (USENIX 08), Boston, MA, June 2008.
13. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In Proceedings of the nineteenth ACM symposium on Operating Systems Principles (SOSP19), pages 164.177. ACM Press, 2003
14. The AMD-V Story http://www.amd.com/us-en/0,,3715_15781_15785,00.html