A Novel Approach to Gain High Throughput and Low Latency through SR-IOV

Usha Devi G[#1], Kasthuri Theja Peduru[#2], Mallikarjuna Reddy B[#3]
School of Information Technology, VIT University, Vellore – 632014, Tamil Nadu, India
peduru.kasthuritheja2011@vit.ac.in, bmallikarjuna.reddy2011@vit.ac.in

Abstract---**IO Virtualization allows multiple virtual machines to share the underlying hardware resources; it has some challenges to I/O performance. The single-root I/O virtualization (SR-IOV) is a new PCI-SIG standard which allows an I/O device to be shared by multiple Virtual Machines (VMs), without losing runtime performance. We propose common virtualization architecture for SR-IOV devices; SR-IOV capable network devices support direct I/O Throughput and reduced CPU utilization. SR-IOV uses direct I/O assignment of a network device to shared VM's for using the maximum bandwidth capabilities of a network device. We carried out experiments to compare SR-IOV and paravirtualized network driver. Experiments shows that SR-IOV claims good line rate and can scale up to 64 VMs with less CPU overhead.**

Keywords: **Paravirtualized; Virtual Machine; SR-IOV; PCI-SIG;**

I INTRODUCTION

In high performance computer systems I/O performance is critical. According to Moore's Law CPU computing capabilities and memory volumes will gradually increase to support multi core technology and different Memory Architectures, but I/O performance is affected by slow PCI Express (PCIe)[1] link and scalability limitation in hardware devices, like PCIe slots integrated to hardware device. Generally CPU cycles are wasted by I/O intensive servers and clients to fetch available data on regular cycles. It affects overall performance and scalability of system. Multiple OS's are allowed to share a single physical interface through Virtualization; it offers effective utilization of underlying system resources, such as I/O devices, RAM, etc. There are two ways to enable virtualization.

Paravirtualization (PV)[2] requires OS modification to work with hypervisor. Full virtualization requires no modification, using hardware supports like Intel® Virtualization Technology. ESXi is a hypervisor which supports both full virtualization and paravirtualization. ESX Server installs directly on the server hardware, or "bare metal", and inserts a robust virtualization layer between the hardware and the operating system. The privileged part of the guest, the Virtual Machine Monitor (VMM), runs in the VMkernel. The different types of virtual Ethernet adapters available for virtual machines are Vmxnet, Vlance and e1000. Vmxnet is a paravirtualized device that works only if VMware tools are installed in the guest operating system. Vlance is a virtual device that provides strict emulation of the AMD Lance PCNet 32 Ethernet adapter. e1000 is a virtual device that provides strict emulation of the Intel e1000 Ethernet adapter.

To handle virtualization effectively I/O performance and scalability needs to be improved. When a guest OS access the I/O device hypervisor is intervened in data transfer activities to share the physical device securely, it leads to additional IO overhead for a virtual machine. Existing driver such as Xen PV driver suffers from hypervisor intervention in data transfer activities [3]. The overhead caused by the intervention could saturate the CPU in a high throughput scenario such as fast Ethernet and thus affects the overall system performance. Some solutions are proposed to reduce the hypervisor intervention like Virtual Machine Device Queue (VMDq)[4] and Direct I/O. VMDq is technique that classifies the packet based on network adapter and delivers the packet in the guest buffer. However, it needs hypervisor intervention for address translation and memory protection.

The second is Direct I/O which assigns a dedicated device to each virtual machine with I/O Memory Management Unit (IOMMU). IOMMU translates the device DMA addresses to the appropriate physical machine addresses[5]. Direct I/O reduced the hypervisor intervention in address translation and memory protection. However it sacrifices lack of scalability and device sharing which are important virtualization functionalities.

Single Root I/O Virtualization (SR-IOV)[1] is a new PCI-SIG industry standard which defines a methodology to share native devices amongst virtual machines. It relies on hardware virtualization, by-passes the hypervisor involvement in packet processing almost entirely and gives the guest control over the physical network device. SR-IOV overcomes the scalability limitations of direct pass through allowing each network device to be exposed as multiple virtual functions to the guest VMs. SR-IOV relies on Intel's VT-d or AMD-Vi and requires BIOS and system or hypervisor support as well as a SR-IOV capable NICs. Similarly to direct pass through the SR-IOV technology allows by-passing the hypervisor involvement in packet processing saving CPU cycles and providing low response time.

SR-IOV introduces the notion of PF (physical function) which has all the capabilities of a typical PCIe function (configuration space, BARs etc.) and VF (virtual function) which is a "lightweight" function which only has data movement capability. Any activity not related to data movement needs to be communicated and handled by the PF driver. Each virtual function has its own PCI configuration space, dedicated transmit and receive queues, interrupts and its own DMA engine. Each VM can be allocated one or multiple VFs. The VF driver in the VM programs the configuration registers of the VFs allocated to it specifying the memory address space from where the data will be copied in and out. When a packet arrives it is dispatched to the VF queue where it belongs from which it is DMA-ed into the guest VM's machine physical memory space with the help of the hardware address translation provided by Intel's VT-d or AMD-Vi.

We carried out experiments to compare SR-IOV and paravirtualized network driver. Result shows SR-IOV can achieve approximately 10Gbps line rate with better throughput and scalability. Hence SR-IOV provides an appropriate virtualization solution with high I/O performance.

The rest of the paper is organized as follows: In section II, describes SR-IOV in detail. Section III describes SR-IOV architecture and some implementation considerations. Section IV discusses new virtualization overhead and proposed optimization technique. Section V gives performance results and section VI covers related work. We conclude the paper in last Section.

II. SR-IOV

In software point of view, interaction between device and CPU can be done in three ways. They are register, interrupt and shared memory as shown in the below figure. Device interacts with software programs through registers and notifies the processor through interrupts. Massive data moments via DMA uses shared memory.

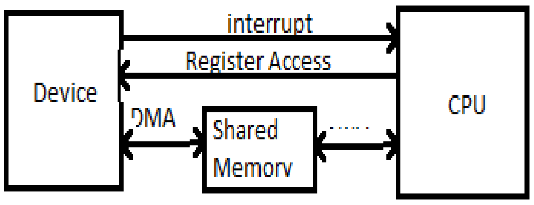


Fig 1: Interaction of device with processor through interrupt, register and shared memory

SR-IOV is a new PCI specification by the PCI-SIG organization which has a rich set of hardware enhancements to PCIe device. It is introduced to eliminate hypervisor intervention in data movement. SR-IOV inherits Direct I/O technology by using IOMMU to maintain address translation and memory protection. In SR-IOV enabled device has one or more physical functions and can be managed to create several VF's as shown in below figure. Each physical function (PF) is a standard function associated with multiple Virtual Function's (VFs). Each VF has critical resources, dedicated to a single software module to support I/O movements. A PCIe function in the PCIe bus has a unique requestor ID.

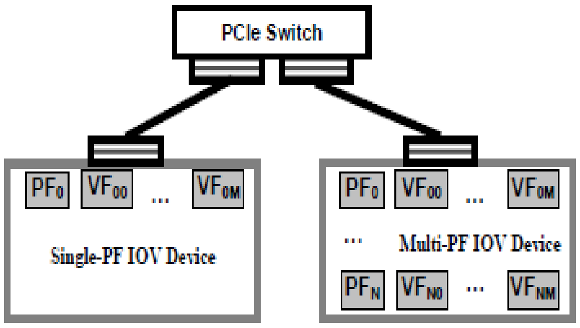


Fig 2: SR-IOV enabled devices

VF's are light weight functions which are created and managed by PF's. Every VF is associated with unique RID, which helps identifying a PCIe data movements. RID is also used as an index to IOMMU page table, so that guest OS can use dedicated page tables. VF has performance-critical resources to share underlying device resources, such as physical layer processing and classification of packets as shown in the below figure.

SR_IOV also includes address translation services for better performance. It owns an I/O device to perform address translation mechanism by catching the Translation Look-a-side Buffer (TLB). This helps the device in translating DMA address, prior to issue the transaction.

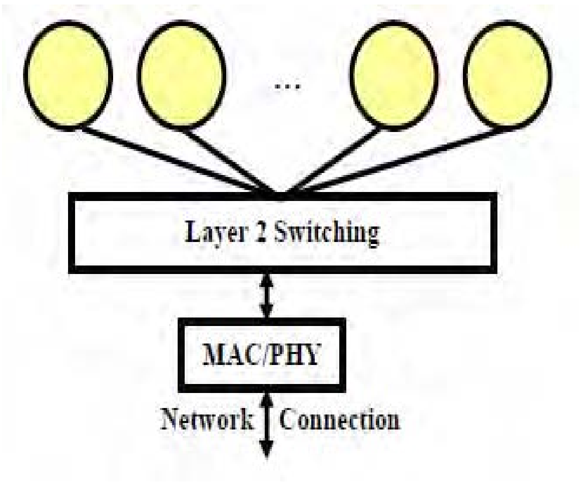


Fig 3: Resource sharing in SR-IOV enabled network devices.

III. SR-IOV VIRTUALIZATION ARCHITECTURE

In this section, we propose a generalized architecture for SR-IOV enabled devices. This can be implemented on various types of hypervisor. The architecture allows VF and PF to be reused across different hypervisor's such as ESXi and KVM[6]. In this section we also reviewed few design considerations for generic virtualization architecture like communication between VF and PF drivers should not be intervened by hypervisor. Our first implementation with NIC exposed new overhead. VF interrupt is still handled by hypervisor to map it from guest to host and vice versa, which is a major SR-IOV performance overhead. Interrupt mask and unmask acceleration are applied to overcome virtualization overhead. Adaptive interrupt coalescing technique is used to minimize CPU utilization.

A. Architecture

Proposed architecture contains PF driver, VF driver and SR-IOV Manager (IOVM). VF driver runs in a virtual machine as a normal device driver, PF driver is to manage physical functions and IOVM sits in service OS to enable and disable control points within PCIe topology and provides a configuration space for VF's to make the architecture as simple and VMM intervention free, each component on the architecture needs to be independent with VMM. For example interaction between VF and PF driver is directly through SR-IOV devices. Below Fig 4 illustrates the SR-IOV architecture.

- PF Driver

PF driver has direct access to all physical resources and it is responsible for creating and managing VF's.it decides the number of VF's to be enabled or disabled and sets up hardware specific configurations such as physical address and virtual LAN settings. It is also responsible for managing layer 2 switching to make sure packets are routed properly.

- VF Driver

VF driver runs on virtual machine as a normal device driver and access the VF dedicated to it, without involving hypervisor.

- IOVM

SR-IOV Manager (IOVM) sits in service OS to enable and disable control points within PCIe topology and provides a configuration space to VF's. When a host OS initializes the SR-IOV enabled devices it cannot see all the VF's by scanning PCIe function device ID. Because VF is a tear down version of "light weight" function, it does not have complete PCIe configuration values. Proposed architecture uses Linux hot had API's to add VF's dynamically. Once a VF is assigned to VM, the VM can utilize the VF as a normal PCIe function.

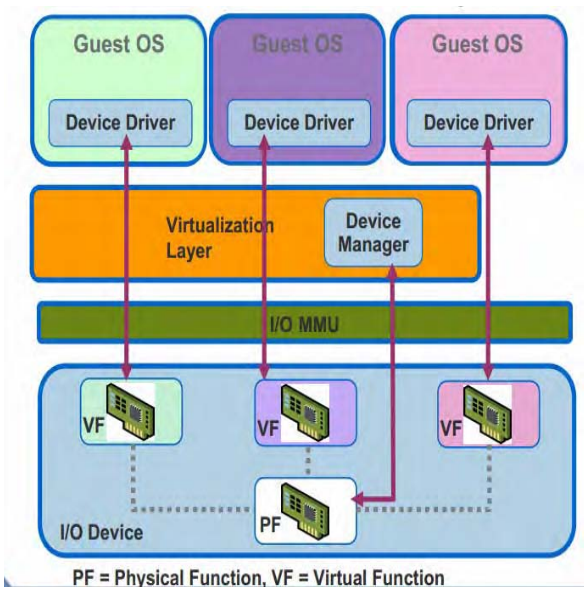


Fig 4: SR-IOV Virtualization Architecture

B. PF Driver/VF Driver Communication

A channel is needed for both PF and VF driver to transfer management and configuration information, as well as interrupts. In proposed architecture communication between PF and VF is based on a private hardware based channel. Intel SR-IOV enabled NIC, implemented this type of hardware based communication. The sender writes a message and rings the door bell, which will notify the receiver that data is ready for consumption. Receiver consumes it and updates the shared register.

C. Security Consideration

SR-IOV allows the PF Driver to enforce policies concerned to VF device bandwidth, interrupt handling, broadcast storms and congestion control etc. PF Driver monitors configuration request and behavior of VF drivers. It will take necessary action if anything found unusual.

IV PROPOSED OPTIMIZATION TECHNIQUE

SR-IOV aims for high performance I/O virtualization and scalability by removing hypervisor intervention. However hypervisor needs to intercept VM interrupt delivery. It captures the physical interrupt raised by VF. We proposed interrupt mask and unmask acceleration optimization which reduce virtualization overhead. Adaptive interrupt coalescing (AIC) achieves low CPU utilization with high throughput.

High frequency interrupts leads to serious performance issues. Contact switching caused by interrupts leads to TLV and cache pollution[7]. Techniques to reduce interrupts, such as NAPI[8] and interrupt coalescing in modern NIC drivers, which throttles interrupt after certain amount of time [9-p6]. Our architecture brings more challenges to interrupt coalescing. Higher bandwidth may be achieved in intervene communication leads to more interrupts and thus may confuse the device driver. Virtualization overhead can be minimized by reducing the frequency of interrupts, but it hurts TCP throughput[8]. AIC is used to configure less interrupt frequency, but won't overflow the application buffer or device driver buffer.

$$bufs = \min(ap\_bufs, dd\_bufs)$$

$$t_d * r = bufs / pps$$

$$IF = 1/t_d = \max(pps/(bufs * r), lif)$$

Here *ap _ bufs* and *dd _ bufs* are application and the device driver respectively, t_d means interval between two interrupts. *pps* are the number of packets received per second. We use lowest acceptable interrupt sequence (*lif*) to cut down the latency. Fig 8 and 9 illustrates the results of bandwidth and CPU utilization for both TCP and UDP streams.

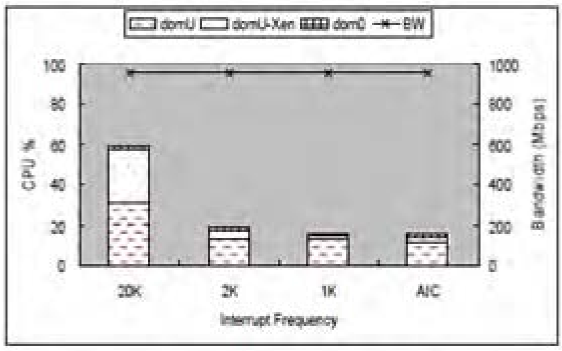


Figure 8 Adaptive interrupt coalescing reduces CPU overhead for UDP_STREAM

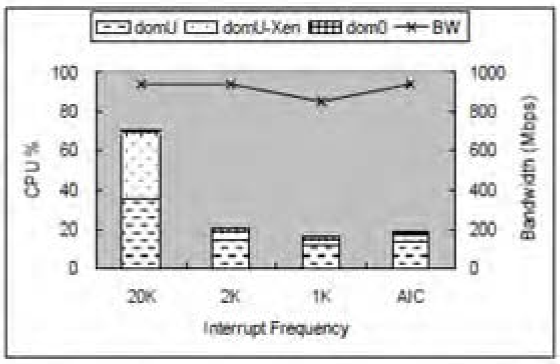
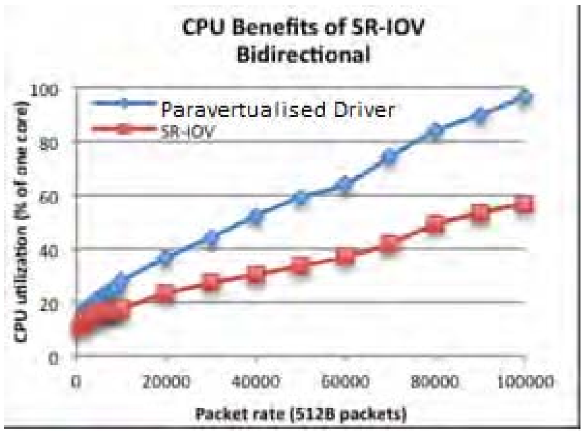


Figure 9 Adaptive interrupt coalescing maintains throughput with minimal CPU utilization for TCP_STREAM

V Experiment and Performance Results

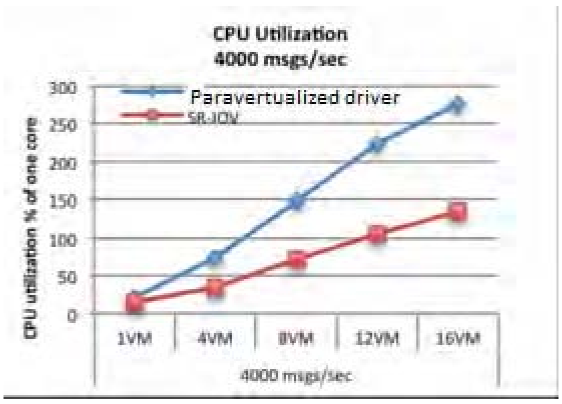
This section evaluates the scalability and performance of the virtualization architecture for SR-IOV enabled devices.

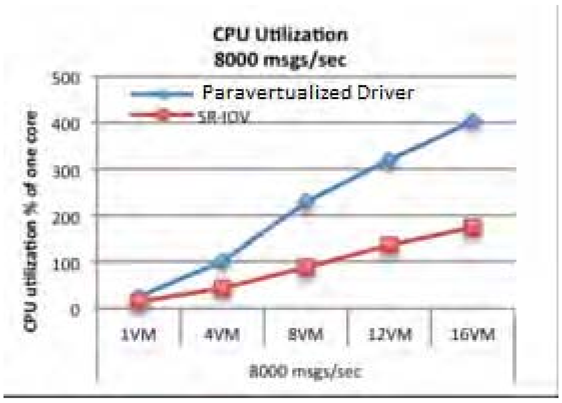
- Comparison of Paravirtualized driver and SR-IOV CPU utilization with Single VM.



CPU utilization delta between SR-IOV and VMXNET3 is small at small packet rates and it becomes larger at large packet rate.

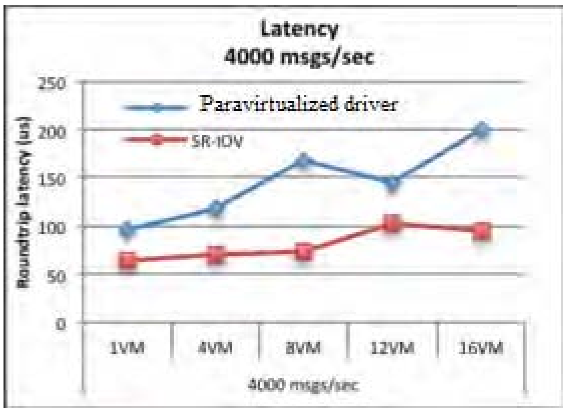
- Comparison of Paravirtualized driver and SR-IOV CPU utilization with Multiple VM's.

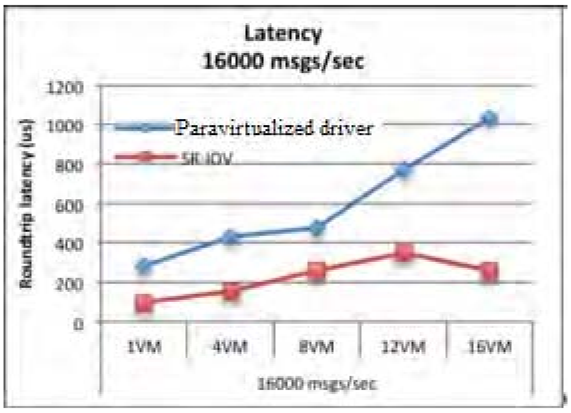




PV driver CPU cost is higher than SR-IOV, with the single VM and 4K messages per sec PV burns more CPU than SR-IOV by ~30%, while at 8K messages per sec it burns ~65% more.

- Comparison of Paravirtualized driver and SR-IOV latency with Single/Multiple VM’s.





The above graphs capture 95% latency. SR-IOV roundtrip latency is less than PV especially at high message rates. When tested with a single VM at 1K messages/second SR-IOV is better than PV by 36% while at 16K messages/second is better by ~280%.

VI. RELATED WORK

Virtualization and performance of Ethernet adapter on Vmware Workstation[9] have been studied. Citrix Xen's PV uses shared memory based data channel to reduce data movement overhead. Various hardware-assisted solutions are proposed in virtualization to achieve high-performance I/O, such as VMDq[3][4] and self-virtualized devices[10]. IOMMU is used to offload address translation and memory protection grabs new attention. Major bottlenecks for high bandwidth I/O are interrupts due to context switching and cache pollution.

VII CONCLUSION

SR-IOV enabled devices provide good I/O throughput and high degree of scalability with reduced CPU utilization. In case of single VF dedicated to a VM increases security capability. SR-IOV based I/O Virtualization is a good base to meet current networking requirements.

VIII ACKNOWLEDGMENT

We would like to thank Durga Gannamani, Gaurish Chaddha ,Akshath Kumar , Ashok Akbote and VMware Ecosystems team members for their excellent insights and suggestions.

REFERENCES

[1] PCI Special Interest Group, http://www.pcisig.com/home
[2] A.Whitaker, M. Shaw, and S. D. Gribble. Denali: Lightweight Virtual Machines for Distributed and Networked Applications. Technical Report 02-02-01, University of Washington, 2002.
[3] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox and S. Rixner, Achieving 10 Gb/s using safe and transparent network interface virtualization, Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, Washington, DC, 2009.
[4] J. R. Santos, Y. Turner, G. Janakiraman and I. Pratt, Bridging the Gap between Software and Hardware Techniques for I/O Virtualization, Proceedings of the USENIX Annual Technical Conference, Boston, MA, 2008. 29-42
[5] Y. Dong, J. Dai, et al. Towards high-quality I/O virtualization. Proceeding of the Israeli Experimental Systems Conference (SYSTOR), Haifa, Israel 2009
[6] Kernel Virtual Machine, http://kvm.qumranet.com/
[7] K. Ramakrishnan, Performance consideration in designing network interfaces, IEEE Journal on Selected Areas in Communications 11(2), 1993 203–219.
[8] J. Salim, When NAPI comes to town, Proceedings of Linux 2005 Conference, Swansea, UK, August 2005.
[9] J. Sugerman, G. Venkitachalam, B. Lim, Virtualizing I/O devices on VMware Workstation's hosted virtual machine monitor, Proceedings of the General Track: 2002 USENIX Annual Technical Conference, Boston, MA, 2001, 1-14
[10] H. Raj, K. Schwan, High performance and scalable I/O virtualization via self-virtualized devices. Proceedings of the 16th international symposium on high performance distributed computing, Monterrey, CA, 2007