Zhou FF, Ma RH, Li J *et al.* Optimizations for high performance network virtualization. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 31(1): 107–116 Jan. 2016. DOI 10.1007/s11390-016-1614-x

Optimizations for High Performance Network Virtualization

Fan-Fu Zhou, Ru-Hui Ma, Jian Li, *Member, ACM, IEEE*, Li-Xia Chen, Wei-Dong Qiu, and Hai-Bing Guan*, *Member, CCF, ACM, IEEE*

Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, Shanghai 200240, China

E-mail: {zhoufanfu, ruhuima, li-jian, trollyxia, qiuwd, hbguan}@sjtu.edu.cn

Received July 5, 2015; revised September 25, 2015.

Abstract The increasing requirements of intensive interoperaterbility among the distributed nodes desiderate the high performance network connections, owing to the substantial growth of cloud computing and datacenters. Network I/O virtualization aggregates the network resource and separates it into manageable parts for particular servers or devices, which provides effective consolidation and elastic management with high agility, flexibility and scalability as well as reduced cost and cabling. However, both network I/O virtualization aggregation and the increasing network speed incur higher traffic density, which generates a heavy system stress for I/O data moving and I/O event processing. Consequently, many researchers have dedicated to enhancing the system performance and alleviating the system overhead for high performance networking virtualization. This paper first elaborates the mainstreaming I/O virtualization methodologies, including device emulation, split-driver model and hardware assisted model. Then, the paper discusses and compares their specific advantages in addition to performance bottlenecks in practical utilities. This paper mainly focuses on the comprehensive survey of state-of-the-art approaches for performance optimizations and improvements as well as the portability management for network I/O virtualization. The approaches include various novel data delivery schemes, overhead mitigations for interrupt processing and adequate resource allocations for dynamic network states. Finally, we highlight the diversity of I/O virtualization besides the performance improvements in network virtualization infrastructure.

Keywords network virtualization, single root I/O virtualization (SR-IOV), virtual machine monitor (VMM)

1 Introduction

The past decade has witnessed the rapid development of cloud computing and datacenter infrastructures, and meanwhile, more and more intensive data exchanges are required among the distributed servers in order to accomplish complex event processing[1], business-related data processing[2], decision support, video conferencing, map-reduce and graph computing tasks[3], etc. According to a Cisco report, the amount of global datacenter IP traffics will nearly quadruple over the next five years, growing at a compounded annual growth rate of 31%[4]. High intensive collaborative workloads require higher physical bandwidth consumption, thereby high performance network devices are widely adopted in datacenter networks. Currently, 10 Gigabit per second (Gbps) Ethernet network interface cards (NICs) have become the mainstream network devices in the cloud computing and datacenters[5-6] and 40 Gbps∼100 Gbps NICs are in the development or close to deployment[7].

Input/output (I/O) virtualization abstracts upper-layer protocols from physical connections and dynamically shares the high-speed interconnect among various requirements without any restrictions on physical port counts[8]. I/O virtualization consolidates many connection traffics to a single physical link to achieve high network resource utilization, and simplifies the management of the I/O devices with software-based

Survey

This work was supported by the National High Technology Research and Development 863 Program of China under Grant No. 2012AA010905, the National Natural Science Foundation of China under Grant Nos. 61272100 and 61202374, the Ministry of Education Major Project of China under Grant No. 313035, and the National Research Foundation (NRF) Singapore under its CREATE Program.

*Corresponding Author

configuration①②. These densified traffics can effectively use fewer cards, cables, and switch ports, so as to significantly reduce datacenter server-to-network and server-to-storage cabling by more than 70 percent③, and significantly lower the cost, complexity, and power requirements④, accordingly. Therefore, I/O virtualization technology achieves outstanding behaviors in terms of greater flexibility, greater utilization and faster provisioning compared with the traditional NIC and HBA (host bus adapter) card architectures⑤.

The continuously increasing speed and density of network traffics result in a higher amount of data movement and higher frequency of network event notifications through the network transmission path. Moreover, the multiplexing of virtual machines (VMs) on sharing physical server resource causes frequent context switch. This costly overhead inflicts the network I/O efficiencies and responsiveness seriously. Therefore, the effectiveness and the efficiency of high performance network virtualization are critical for virtualization-based cloud computings and datacenter systems[9]. To this end, novel network data transmission schemes, effective network event notification processing mechanisms and dynamic system scheduling methods are highly desired to improve the network I/O virtualization performance, especially for 10 Gbps Ethernet NICs and higher speed NICs connections[10].

Generally speaking, I/O virtualization can be classified into three methods according to the network virtualization process, namely the device emulation model, the split driver model, and the hardware assisted model. In the full virtualization scheme, the hypervisor emulates device hardware with clean abstraction, but the I/O operation of guest operating system (OS) needs a costly and complicated trap-and-emulation process to execute the lowest level of the conversation[11]. The para-virtualization (PV) scheme employs split driver model to set up a more efficient cooperation channel between the guest OS and the hypervisor⑥. However, the guest in PV needs to be modified to be aware of its virtualized status, while the guest OS in full virtualization is unaware of virtualization and requires no changes to work[12]. More recently, hardware assisted I/O virtualization offloads the device emulation functions to specific hardware, and provides natively shareable devices for each VM with direct access to I/O device without VMM involvement, such as single- and multi-root I/O virtualization proposed by Peripheral Component Interconnect Special Interest Group (PCI-SIG)⑦.

In this survey, we comprehensively and profoundly discuss the challenges for high performance network virtualization and the requirements according to specific high performance network virtualization architectures. Also, we make a detailed survey of the state-of-the-art approaches for performance optimizations based on data moving, interrupt handling and resource allocation. Besides, we state the management enhancement as well as the diversity of I/O virtualization.

The remainder of the paper is organized as follows. Section 2 introduces the basic I/O virtualization models and their different I/O properties. Afterwards, we conclude the performance bottlenecks and challenges in Section 3. Section 4 concludes and summarizes the state-of-the-art research strategies about performance optimizations and improvements for network I/O virtualization, as well as discusses the diversity of I/O virtualization. Finally, we conclude this paper in Section 5.

2 Network I/O Virtualization Models

Virtualization technology has a long development history for logically dividing the system resources provided by mainframe computers between different applications[13-14]. I/O virtualization is a critical component to achieve the successful and effective virtualized servers. It is recognized that 75% of virtualized servers require seven or more I/O connections per device, and require more frequent I/O reconfigurations⑧.

①http://www.embedded.com/design/mcus-processors-and-socs/4008300/I-O-Virtualization-IOV–its-uses-in-the-Network-Infrastructure-Part-3, Sept. 2015.

②http://searchitchannel.techtarget.com/feature/I-O-virtualization-IOV-Delivering-cost-and-power-savings-to-data-center-servers, Sept. 2015.

③http://www.krome.co.uk/wp-content/uploads/Business-Case-for-Virtual-IO.pdf, Sept. 2015.

④http://searchitchannel.techtarget.com/feature/I-O-virtualization-IOV-Delivering-cost-and-power-savings-to-data-center-servers, Sept. 2015.

⑤http://www.dell.com/downloads/global/power/ps3q09-20090423-Xsigo.pdf, Sept. 2015.

⑥In this article, the term guest is interchangeable with guest domain and guest OS, and the term hypervisor is interchangeable with VMM (virtual machine monitor).

⑦http://www.pcisig.com/specifications/iov, Sept. 2015.

⑧https://virtualizationreview.com/articles/2008/03/31/new-things-to-virtualize.aspx, Sept. 2015.

In what follows, we will introduce three I/O virtualization models: the device emulation model, the split driver model and the hardware assisted model.

2.1 Device Emulation

The device emulation method for I/O virtualization is normally employed in full-virtualization, which mimics I/O operations of the physical devices by virtualizing sensitive and privileged instructions in virtual machine monitor (VMM) layer. This effect fully abstracts and decouples the underlying hardware from guest OS with the combination of binary translation and direct execution techniques. The device emulation method offers the best performance isolation and security for I/O operations, and simplifies migration and portability as guest OS instance and user level instructions run unmodified without the awareness of virtualization. The device emulation method using binary translation has become the de facto standard in VMware product series, such as VMware Workstation[15] and VMware ESX Server[16]. In addition, QEMU[17] of Xen[12] and KVM[18] can also be configured to deploy the device emulation method for I/O virtualization.

However, the device emulation method provides a weak fault isolation because VMM needs to translate all operating system instructions on the fly and caches the results, so that a malfunction of VMM device driver will lead to the I/O outages of all guests. Moreover, it causes the performance challenges owing to the costly trap-and-emulation of these sensitive and privileged instruction requests at runtime. It is reported that every typical trap-and-emulation for an I/O operation will result in the context switch between guest OS and VMM that costs around 3 000∼5 000 CPU cycles[19]. In practice, it significantly degrades the I/O throughput and scalability, and loses the effective utilization of physical line rate. For example, a virtual 1 Gbps Ethernet NIC can only achieve about 250 Mbps throughput with the full utilization of a CPU core[20]. This challenge prevents the device emulation method from being deployed for high performance network in cloud computing and datacenters equipped with modern NIC (e.g., 10 Gbps NIC).

2.2 Split Driver Model

To alleviate high overhead of VMM intervention, the split driver model of PV was first developed by Xen group[12,21]. KVM's virtio[18,22], VMware tools and virtual machine interface[23] can also support split driver model for PV.

The split driver model of Xen PV is illustrated in Fig.1, which consists of a front-end driver, a back-end driver, and event notification channels between the domains. Every guest domain runs one or more virtualized network interfaces, called front-end drivers, to request the network access from driver domain (dom0). Each front-end driver works cooperatively with the back-end driver in dom0. The front-end driver and the back-end driver communicate with virtual descriptor queues, which are implemented with shared rings and memory pages, to exchange requests and responses.

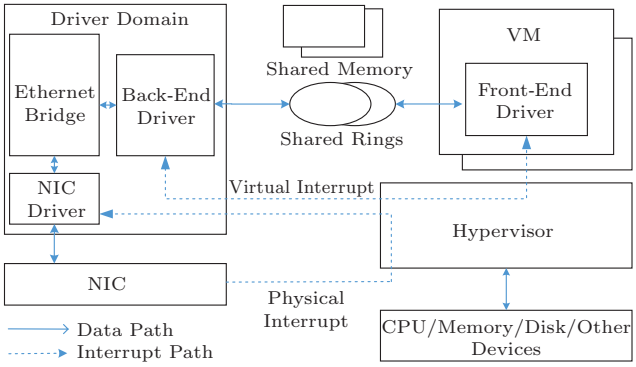


Fig.1. I/O virtualization architecture of split driver model.

The split driver model employs the efficient I/O communication channel (I/O channel) in order to reduce the hypervisor intervention for emulating port I/O and memory mapped I/O functions. Consequently, the split driver can take advantage of host device capabilities to offer more throughput and less CPU utilization than the device emulation method. Even when PV requires deep OS kernel modifications, the split driver model shows its great flexibility in supervision and migration, and support in modern hypervisors and guest domains[11,24].

2.3 Hardware Assisted I/O Virtualization

The device emulation and split driver models are both classified into software-based I/O virtualization which enables a rich set of features and simplified management. However, they suffer from high overhead stemming from either excessive trap-and-emulations or bulk data movements[21,25].

Consequently, the hardware assisted I/O virtualization model was developed for natively shareable devices, which inherits direct I/O technology through the use of I/O memory management unit (IOMMU) to offload memory protection and address translation. The PCI-SIG gives the specifications to standardize

the ways of bypassing the VMM's involvement in data movement by providing independent memory space, interrupts, and DMA streams for each VM. Typically, single root I/O virtualization (SR-IOV)[26] was created by PCI-SIG, which designs a set of hardware enhancements for the PCIe device to remove major VMM interventions for data movement, such as packet classification and address translation.

The SR-IOV virtualization architecture is shown in Fig.2, where an SR-IOV-capable device can be configured by the VMM to appear in the PCI configuration space as multiple virtual functions (VFs). The SR-IOV-capable device provides configurable numbers of independent VFs, each VF with its own configuration space completed with base address registers (BARs). The physical function (PF) driver in a service OS (host OS) is responsible for managing and configuring VFs, and each VF driver runs in a guest OS as a normal device driver to access its dedicated VF directly through IOMMU. Consequently, SR-IOV can bypass the hypervisor to alleviate data moving overhead and enable lower CPU utilization, reduced system latency and improved networking throughput.

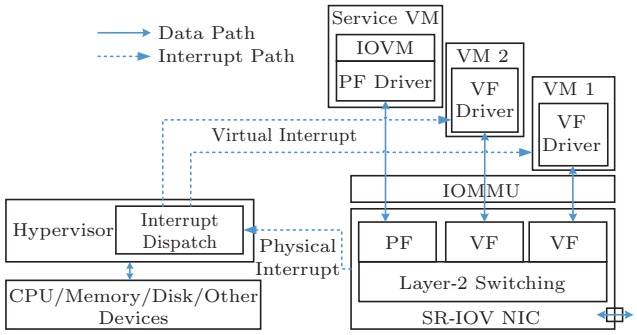


Fig.2. SR-IOV Architecture.

In comparison with software-based I/O virtualization, SR-IOV is able to achieve close to native performance by offloading I/O transmission overhead to the hardware assistance techniques on NIC. However, there are still some challenges. 1) Interrupt handling still involves VMM interception with heavy overhead and excessive frequency[9,27-28]. 2) Hardware assisted technology limits the scalability since the number of VF is limited by the feature of SR-IOV capable device. 3) VM with SR-IOV driver has difficulties to be migrated to a remote server, since VF driver's running status is partly maintained in hardware, and cannot be extracted and restored easily on another server. The detailed challenges will be depicted in the next section.

3 Requirements and Challenges

In the previous section, we have introduced three models of network I/O virtualization. Note that the device emulation model and the split driver model are software-based implementation, while the SR-IOV model offloads VMM intervention of the packet switching and data movement from software implementation to NIC devices with hardware assisted layer-2 switch and IOMMU for memory protection and address translation, respectively. Moreover, the virtual interrupt notification and the related event processing procedure remain similarly in the split driver model and SR-IOV. In what follows, we will introduce the specific and detailed challenges for I/O virtualization models.

3.1 Preliminary of Network Virtualization

Without loss of generality, network I/O virtualization needs to achieve high throughput and low latency with lightweight system overhead[29-32]. Moreover, network I/O virtualization technology should provide high scalability to share the physical I/O device to multiple VMs for effective server consolidation. The flexible manageability and maintainability are also the important properties so that the VM with network virtualization can be migrated across the physical server to attain the efficient power consumption and workload balancing.

3.2 Challenges of Network Virtualization

The primary overheads of network I/O virtualization stem from the data transmission and interrupt handling, especially for high performance NICs with a physical line rate of 10 Gbps.

Excessive Interrupt. Both the split driver model and SR-IOV incur significant notification overhead due to the intervention of the hypervisor when handling interrupts[19,33-34]. In native virtualized environments, each I/O transaction triggers at least three exits: interrupt issue, interrupt injection and interrupt completion. When guests need to issue I/O transactions, they have to exit and switch to the hypervisor to send the privileged I/O notification, and we call this procedure interrupt issue (guest-to-host). After the completion of I/O request, guest VMs need to exit to hypervisor again and hypervisor will inject a virtual interrupt and resume the guests' execution, called interrupt inject (host-to-guest). While the guests complete handling the virtual interrupt and then signal the completion of interrupt handler by writing End of Interrupt

(EOI) register, another exit called interrupt completion (trap-and-emulation of EOI) is triggered. SR-IOV device only triggers two exits: interrupt injection and interrupt completion, as a VM is able to issue I/O instructions directly to the SR-IOV-capable device. The context switches occur due to the physical interrupts and the trap-and-emulation on the virtual interrupt controller. Consequently, an interrupt handling path is lengthened and the cache is polluted. For example, a process of trap-and-emulation EOI costs 8.4K cycles and 47% of advanced programmable interrupt controller (APIC) access exits are caused by EOI write in SR-IOV network[35].

Meanwhile, the interrupt frequency in SR-IOV network increases almost linearly with the number of VMs increasing, as each VF generates its own interrupts in parallel[28,35]. In the split driver model, upon processing an inbound packet, the back-end driver interrupts a guest OS immediately, causing excessive virtual interrupts[19].

Data Moving/Transmission. In the split driver, the data moving involves copying packet not only from NIC to driver domain, but also from driver domain to guest domain. Moreover, it is heavy system overhead to execute the software implementation for I/O delivery functions components, such as Ethernet bridge, back-end driver. For example, it costs about 18 200 CPU cycles in a packet delivering[36].

Manageability. On one hand, SR-IOV compromises the resource control capability of the hypervisor, and suffers the problem of hardware stickiness. Once the VF is "initialized" and is assigned to a VM, all packet processing operations are performed in hardware, and the VMM needs only to handle I/O interrupts without data moving overheads. Since the device state is partly retained by hardware, virtualization enabled device operations like migration and cloning are complicated. On the other hand, current SR-IOV based solution does not support the dynamic assignment of VFs to VMs. This static assignment prevents VF from being dynamically provisioned across VMs to meet SLA (Service Level Agreement) requirements under different workload conditions.

Scalability. Even without theoretical configuration limitation, the software-based I/O virtualization method can only share the physical NIC with a very limited number of VMs, because of the expensive system overheads. In contrast, hardware-assisted I/O virtualization achieves the linear scalability, but PCI-SIG specification has an upper bound of 64 VF drivers, Intel 82576 Gigabit Ethernet controller only supports no more than 8 VFs[37], and Intel 82599 10 Gigabit Ethernet NIC supports at most 64 VFs[38].

Due to these challenges for I/O virtualization, many researchers have dedicated to searching for performance optimization schemes and flexible management mechanisms. The next section will demonstrate the state-of-the-art approaches for software-based I/O virtualization and SR-IOV, respectively.

4 Overhead Alleviation and Performance Optimizations

As specific challenges described in Subsection 3.2, data transmission and excessive interrupts affect network performance, while manageability has enormous impact on network maintenance and management. Thus, we survey the network performance improvements for data transmission and excessive interrupts in Subsection 4.1 and Subsection 4.2, respectively. Methods for resource allocation speed both data moving and interrupt handling, and they are summarized in Subsection 4.3. Then we briefly survey the improvements of network manageability in Subsection 4.4 as well as discuss the diversity of I/O virtualization in Subsection 4.5.

4.1 Improvements for Data Transmission

SR-IOV bypasses the hypervisor involvement in data transmission, and achieves near native performance with unshrinkable overheads for data transmission. However, the split driver model has a long data path during the data transmission, which not only incurs significant overheads in extra data copy, but also poses remarkable expenses to maintain functions of packet delivery. In this subsection, we introduce the enhancement for network performance optimization from the middle components, including Ethernet bridge, I/O channel, guest domain, and data copy.

4.1.1 Optimization of Bridge

Bridge in the split driver model is utilized to multiplex network traffics for multiple guests. It is discovered that Linux bridge executes 4K instructions and consumes about 11K cycles to switch one packet, which is mainly incurred by Netfilter interface to integrate filter rules of received/transmitted packets[36,39]. The simplified bridges are proposed to bypass most of the functions of Netfilter interface with the re-implementation of the internal interfaces to minimize extra function

costs except the bridge itself. Liao *et al.*[39] verified that the simplified bridge prototype can increase the bandwidth by 7% and save 10% in core utilization per gigabit when the message size is greater than 1 KB. Martins *et al.*[40] proposed an efficient software-based middlebox on Xen platform, named ClickOS, whose key component is to replace costly Xen's Ethernet bridge with ClickOS switch to boost network performance.

4.1.2 Optimization of Front-End Driver

The front-end driver in guest domain posts full page buffers as fragments in socket buffers, even for normal size Ethernet frames. This fragment approach thereby introduces copy overheads and socket buffer memory allocation overheads. Researchers intended to modify the front-end driver by pre-allocating socket buffers and posting those buffers directly into the I/O channel instead of posting fragment pages[36]. The efficiency was verified by saving 5% of the total CPU cost during a packet delivery. Similarly, researchers provided the large receive offload (LRO) approach to aggregate small I/O requests, reducing the guest domain processing cost by 15% (reduced from 4 721 to 4 013 CPU cycles per packet)[41]. Moreover, Menon *et al.* presented a virtual driver to offload scatter/gather I/O, TCP/IP checksum and TCP segmentation[25] to optimize the I/O performance.

Ram *et al.* found the costly cache miss penalties caused by the front-end driver for accessing the packets, and modified the front-end driver to prefetch packet headers. This reduces the processing cost per packet in the guest domain from 4 013 to 3 662 CPU cycles. Ram *et al.* also noted that it is wasting to allocate full 4 KB pages larger than MTU (maximum transmit unit) of IP traffics, and suggested a half-page allocation technique (2 KB pages) to reduce TLB miss rate and improve grant table re-usage, cutting the guest domain processing cost from 3 662 to 3 266 cycles for each packet processing[41].

4.1.3 Optimization of Grant Mechanism

The grant mechanism in Xen enables the driver domain to access I/O buffers in guest memory safely without compromising the isolation property. Researchers[36,41] realized that the grant operations incur massive system cost due to acquiring and releasing the spin lock, pinning pages, and atomic swap operations to update grant status. Consequently, they combined the operations of multiple grants in a single critical section to reduce the number of spin lock operations, reused the pages already pinned and mapped to save pinning overhead, and separated grant fields in different words to avoid the use of expensive atomic operations. It is proven that these methods can save about 4% CPU resource consumptions for each packet moving cost.

4.1.4 Data Copy Cost Alleviation

Researchers also discovered that the remarkable CPU overheads are introduced by misaligned data copy during packet delivery[36], and proposed to use aligned data copy methods. Of the 18 200 CPU cycles consumed per packet delivery, this approach decreased 1 850 cycles per packet.

Menon *et al.* recognized that guest domains suffer from much higher L2 cache miss rate than driver domain in [42], and they supported superpage mapping and global page mapping in virtual I/O environment for lowering L2 cache miss rates in [25]. Similarly, Liao *et al.* proposed the cache-aware scheduler to arrange the two virtual central processing units (vCPUs) with data sharing on the two cores sharing L2 cache[39], and this method increases the bandwidth by 13% and saves 11% core utilization. Moreover, researchers exploited the facility of inter-domain shared memory provided by Xen hypervisor to achieve higher performance, such as XenSockets[43], IVC[44], Xway[45] and Xenloop[46].

4.2 Optimization for Interrupt Handling

Note that the most effective way to decrease the overhead of packet transmission is to tackle the extra copy by bypassing the hypervisor involvement with the help of hardware assistance[47-48]. This method not only eliminates the extra overhead of data copy, but also diminishes overheads spent in driver domain, gust domain and I/O channel. However, the SR-IOV model treats with I/O event processing similarly to the split driver model, which remains the critical system overhead. Therefore, the mitigation methods on excessive interrupt overheads for both the split driver model and SR-IOV are concluded in this subsection. The optimizations to alleviate interrupt handling overhead can be classified into interrupt coalescing, Exit Less Interrupt and event-based polling.

Interrupt Coalescing. Interrupt coalescing is a technique which triggers a hardware interrupt until a certain amount of work is pending, or a timeout timer triggers. High performance NIC incurs excessive interrupts

and many optimization results are obtained with the interrupt coalescing idea[19,49] to throttle interrupts. Its efficiency can achieve a CPU utilization reduction by about 71% with the full use of physical line rate[19,33]. Moreover, the adaptive interrupt rate control method to limit the number of interrupts can further reduce CPU utilization and retain high throughput, according to the awareness of practical VM run-time status or application behaviors[28,35]. The side effect of interrupt coalescing is that it may lengthen the I/O latency even with lower CPU consumption.

Exit Less Interrupt. Recalling each interrupt in SR-IOV network incurs at least two exits: interrupt injection and interrupt completion in Subsection 3.2. Dong *et al.* proposed the direct virtual APIC EOI write emulation to bypass the exit[35] which can reduce 5 900 CPU cycles for each interrupt completion exit. Gordon *et al.* proposed ELI (Exit Less Interrupt) scheme[34] to reduce the exit frequency, by delivering the hardware interrupts directly to a given core with the help of the guest's shadow interrupt descriptor table (shadow IDT) and exposing the EOI register immediately to the guest. Furthermore, ELVIS (Efficient and Scalable Para-Virtual I/O System)[50] enhances the scalability and performance for paravirtual I/O by alleviating the competition among multiple co-hosted I/O-intensive guests. Similarly, Tu *et al.* proposed the DID (Direct Interrupt Delivery) scheme[51] that completely eliminates most of the exits and priority inversion problem due to direct EOI write with interprocess interrupt (IPI) based virtual interrupt injection mechanism.

Event-Based Polling. sEBP[9,27] exploits various system events such as system calls and exits to imitate the notification role of I/O interrupts. This completely eliminates the interrupt virtualization in the critical I/O path by accompanying events in virtualized environment, and achieves up to a 59% performance improvement with a 23% higher scalability.

4.3 Resource Allocation

Due to the high dynamic I/O connection feature, dynamic and enhanced system resource allocation methods are also employed, such as I/O stealing scheduling and parallelized driver.

Multicore Scheduling. Dong *et al.*[19] found that the single threaded back-end driver in the split driver is not capable of treating with the packets for 10 Gbps NICs, and proposed virtual receive side scaling (vRSS)[37] to effectively leverage multi-core processors. Further, Huang *et al.* proposed multi-threaded network driver (MTND) which allows SR-IOV to make full use of multi-core resources, improving the throughput by 38% with an additional 73% CPU cost[28].

I/O Stealing Scheduling. Liao *et al.* proposed the dynamic and temporal credit sharing scheme among vCPUs on Xen platform with I/O split model, and the scheme assigns the favor of credits to I/O vCPUs that are busy with processing network packets[39]. This scheme can improve the network performance by 18% and save 5% in CPU utilization. Further, researchers studied the efficiency of SR-IOV, and proposed an adjustable credit scheduler. This method monitors the SR-IOV driver to distinguish I/O intensiveness of every VM, and I/O-intensive domains can obtain extra credits from CPU-intensive domains with a constraint of a predefined ratio[52-53]. This approach improves I/O performance, and keeps the fairness of CPU allocation since I/O-intensive domains may suffer from occasionally idle network traffic at credit allocation time.

4.4 Manageability Enhancement

Software I/O virtualization supports convenient management for VM cloning and migration, while SR-IOV networks suffer from portability issue due to hardware stickiness. Dont *et al.* proposed HYVI[24], which obtains efficient packet delivery with SR-IOV driver between host and guest and achieves flexible network management with software-based IP network packet filtering mechanism for external network. Also, Dey *et al.* presented Vagabond method in [11] which can seamlessly and dynamically switch the network between the split driver model and SR-IOV solutions. They both combine the high performance of SR-IOV with the flexible manageability of para-virtualizaiton.

4.5 Diversity of I/O Virtualization

Note that I/O virtualization is the architecture for general I/O devices, which include Disk, other network connection such as Infiniband, and other I/O devices such as graphic card. However, in the article, we mainly focus on the I/O virtualization for high speed Ethernet NICs. For example, HyV[54] utilizes hardware assisted I/O virtualization for Infiniband networks. Also, Younge *et al.* leveraged IOMMU and SR-IOV technologies to support NVIDIA Tesla GPUS device[55-56]. There are also other research interests such as network security and availability, involving in network I/O virtualization. However, in this article, we concentrate on performance optimizations.

5 Conclusions

Network I/O virtualization decouples virtual device function from physical device, which is an effective way to reduce device cost and provide flexible manageability. In this survey, we reviewed existing I/O virtualization architectures, categorized them into the device emulation model, the split driver model and the hardware assisted model, and discussed the characteristics and differences for each model. Afterwards, we profoundly analyzed the challenges in network virtualization, and gave a detailed survey of the state-of-the-art optimizations for network performance as well as network manageability. As we focused on I/O virtualization on Ethernet network's performance and manageability, we briefly noted that I/O virtualization also confronts the performance challenges for diverse network connections other I/O devices, as well as the I/O access security and availability issues, etc.

References

[1] Zhang H, Diao Y, Immerman N. On complexity and optimization of expensive queries in complex event processing. In *Proc. the 2014 ACM SIGMOD International Conference on Management of Data*, June 2014, pp.217-228.

[2] McWherter D T, Schroeder B, Ailamaki A, Harchol-Balter M. Priority mechanisms for OLTP and transactional web applications. In *Proc. the 20th International Conference on Data Engineering*, Mar.30-Apr.2, 2004, pp.535-546.

[3] Chen R, Shi J, Chen Y, Chen H. PowerLyra: Differentiated graph computation and partitioning on skewed graphs. In *Proc. the 10th European Conference on Computer Systems*, Apr. 2015.

[4] Cisco. Cisco global cloud index: Fore-cast and methodology, 2014-2019. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf, Sept. 2015.

[5] Amman M, Briceno C, Connell K, Rungta S. Upgrading data center network architecture to 10 gigabit Ethernet. 2011. http://intelethernet-ibm.com/pdf/Upgrading_Data_Center_Network_Architecture_10Gb_Ethernet2_Final.pdf, Sept. 2015.

[6] IDC. The new need for speed in the datacenter network. 2015. http://www.cisco.com/c/dam/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c-11-734328.pdf, Sept. 2015.

[7] Porter G, Strong R, Farrington N, Forencich A, Sun P C, Rosing T, Fainman Y, Papen G, Vahdat A. Integrating microsecond circuit switching into the data center. In *Proc. the 2013 ACM SIGCOMM*, Aug. 2013, pp.447-458.

[8] Waldspurger C, Rosenblum M. I/O virtualization. *Communications of the ACM*, 2012, 55(1): 66-73.

[9] Guan H, Dong Y, Tian K, Li J. SR-IOV based network interrupt-free virtualization with event based polling. *IEEE Journal on Selected Areas in Communications*, 2013, 31(12): 2596-2609.

[10] Liu J, Abali B. Virtualization polling engine (VPE): Using dedicated CPU cores to accelerate I/O virtualization. In *Proc. the 23rd International Conference on Supercomputing*, June 2009, pp.225-234.

[11] Dey K, Mishra D, Kulkarni P. Vagabond: Dynamic network endpoint reconfiguration in virtualized environments. In *Proc. the 5th ACM Symposium on Cloud Computing*, Nov. 2014, pp.21:1-21:13.

[12] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. In *Proc. the 19th ACM SOSP*, Oct. 2003, pp.164-177.

[13] Goldberg R P. Survey of virtual machine research. *Computer*, 1974, 7(9): 34-45.

[14] Creasy R J. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development*, 1981, 25(5): 483-490.

[15] Sugerman J, Venkitachalam G, Lim B H. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proc. the 2001 USENIX Annual Technical Conference*, June 2001.

[16] Ahmad I, Anderson J M, Holler A M, Kambo R, Makhija V. An analysis of disk performance in VMware ESX server virtual machines. In *Proc. the 6th Annual Workshop on Workload Characterization*, Oct. 2003, pp.65-76.

[17] Bellard F. QEMU, a fast and portable dynamic translator. In *Proc. the 2005 USENIX Annual Technical Conference*, Apr. 2005, pp.41-46.

[18] Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. kvm: The linux virtual machine monitor. In *Proc. the Linux Symposium*, June 2007, pp.225-230.

[19] Dong Y, Xu D, Zhang Y, Liao G. Optimizing network I/O virtualization with efficient interrupt coalescing and virtual receive side scaling. In *Proc. the IEEE CLUSTER*, Sept. 2011, pp.26-34.

[20] Ben-Yehuda M, Day M D, Dubitzky Z, Factor M, Har'El N, Gordon A, Liguori A, Wasserman O, Yassour B A. The turtles project: Design and implementation of nested virtualization. In *Proc. the 9th USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2010, pp.423-436.

[21] Fraser K, Hand S, Neugebauer R, Pratt I, Warfield A, Williamson M. Safe hardware access with the Xen virtual machine monitor. In *Proc. the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure* (*OASIS*), Oct. 2004.

[22] Russell R. Virtio: Towards a de-facto standard for virtual I/O devices. *ACM SIGOPS Operating Systems Review*, 2008, 42(5): 95-103.

[23] Amsden Z, Arai D, Hecht D, Holler A, Subrahmanyam P. VMI: An interface for paravirtualization. In *Proc. the 2006 Linux Symposium*, July 2006, pp.363-378.

[24] Dong Y, Zhang X, Dai J, Guan H. HYVI: A hybrid virtualization solution balancing performance and manageability. *IEEE Transactions on Parallel and Distributed Systems*, 2014, 25(9): 2332-2341.

[25] Menon A, Cox A L, Zwaenepoel W. Optimizing network virtualization in Xen. In *Proc. the 2006 USENIX Annual Technical Conference*, May 30-June 3, 2006, pp.15-28.

[26] Intel. PCI-SIG SR-IOV primer: An introduction to SR-IOV technology. 2011. http://www.intel.com/content/www/-us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html, Sept. 2015.

[27] Tian K, Dong Y, Mi X, Guan H. sEBP: Event based polling for efficient I/O virtualization. In *Proc. the 2012 CLUSTER*, Sept. 2012, pp.135-143.

[28] Huang Z, Ma R, Li J, Chang Z, Guan H. Adaptive and scalable optimizations for high performance SR-IOV. In *Proc. the 2012 CLUSTER*, Sept. 2012, pp.459-467.

[29] Wilson C, Ballani H, Karagiannis T, Rowtron A. Better never than late: Meeting deadlines in data-center networks. In *Proc. the 2011 ACM SIGCOMM*, Aug. 2011, pp.50-61.

[30] Vamanan B, Hasan J, Vijaykumar T. Deadline-aware datacenter TCP (D 2 TCP). In *Proc. the 2012 ACM SIGCOMM*, Aug. 2012, pp.115-126.

[31] Alizadeh M, Kabbani A, Edsall T, Prabhakar B, Vahdat A, Yasuda M. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Proc. the 9th USENIX Conference on Networked Systems Design and Implementation*, Apr. 2012, pp.253-266.

[32] Meisner D, Sadler C M, Barroso L A, Weber W D, Wenisch T F. Power management of online data-intensive services. In *Proc. the 38th Annual International Symposium on Computer Architecture* (*ISCA*), June 2011, pp.319-330.

[33] Ahmad I, Gulati A, Mashtizadeh A. VIC: Interrupt coalescing for virtual machine storage device IO. In *Proc. the 2011 USENIX Annual Technical Conference* (*ATC*), June 2011, pp.45-58.

[34] Gordon A, Amit N, Har'El N, Ben-Yehuda M, Landau A, Schuster A, Tsafrir D. ELI: Bare-metal performance for I/O virtualization. *ACM SIGARCH Computer Architecture News*, 2012, 40(1): 411-422.

[35] Dong Y, Yang X, Li X, Li J, Tian K, Guan H. High performance network virtualization with SR-IOV. In *Proc. the 16th International Symposium on High Performance Computer Architecture* (*HPCA*), Jan. 2010.

[36] Santos J R, Turner Y, Janakiraman G J, Pratt I. Bridging the gap between software and hardware techniques for I/O virtualization. In *Proc. the 2008 USENIX Annual Technical Conference*, June 2008, pp.29-42.

[37] Intel. Intel® 82576 gigabit Ethernet controller: Datasheet. 2014. http://www.intel.com/content/www/us/en/intelligent-systems/networking/82576eg-gbe-datasheet.html, Sept. 2015.

[38] Intel. Intel® 82599 10 gigabit Ethernet controller: Datasheet. 2015. http://www.intel.com/content/www/us/en/embedded/products/networking/82599-10-gbe-controller-datasheet.html, Sept. 2015.

[39] Liao G, Guo D, Bhuyan L, King S R. Software techniques to improve virtualized I/O performance on multi-core systems. In *Proc. the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, Nov. 2008, pp.161-170.

[40] Martins J, Ahmed M, Raiciu C, Olteanu V, Honda M, Bifulco R, Huici F. ClickOS and the art of network function virtualization. In *Proc. the 2014 USENIX NSDI*, Apr. 2014, pp.459-473.

[41] Ram K K, Santos J R, Turner Y, Cox A L, Rixner S. Achieving 10 Gb/s using safe and transparent network interface virtualization. In *Proc. the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Mar. 2009, pp.61-70.

[42] Menon A, Santos J R, Turner Y, Janakiraman G J, Zwaenepoel W. Diagnosing performance overheads in the Xen virtual machine environment. In *Proc. the 1st ACM/USENIX International Conference on Virtual Execution Environments*, June 2005, pp.13-23.

[43] Zhang X, McIntosh S, Rohatgi P, Griffin J L. XenSocket: A high-throughput interdomain transport for virtual machines. In *Proc. the 8th Middleware*, Nov. 2007, pp.184-203.

[44] Huang W, Koop M J, Gao Q, Panda D K. Virtual machine aware communication libraries for high performance computing. In *Proc. the 2007 ACM/IEEE Conference on Supercomputing*, Nov. 2007, pp.9:1-9:12.

[45] Kim K, Kim C, Jung S I, Shin H S, Kim J S. Interdomain socket communications supporting high performance and full binary compatibility on Xen. In *Proc. the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Mar. 2008, pp.11-20.

[46] Wang J, Wright K L, Gopalan K. XenLoop: A transparent high performance inter-VM network loopback. In *Proc. the 17th International Symposium on High Performance Distributed Computing*, June 2008, pp.109-118.

[47] Willmann P, Shafer J, Carr D, Menon A, Rixner S, Cox A L, Zwaenepoel W. Concurrent direct network access for virtual machine monitors. In *Proc. the 13th HPCA*, Feb. 2007, pp.306-317.

[48] Raj H, Schwan K. High performance and scalable I/O virtualization via self-virtualized devices. In *Proc. the 16th International Symposium on High Performance Distributed Computing*, June 2007, pp.179-188.

[49] Salim J H. When NAPI comes to town. In *Proc. the Linux 2005 Conf*, Apr. 2005.

[50] Har'El N, Gordon A, Landau A, Ben-Yehuda M, Traeger A, Ladelsky R. Efficient and scalable paravirtual I/O system. In *Proc. the 2013 USENIX Annual Technical Conference*, June 2013, pp.231-242.

[51] Tu C C, Ferdman M, Lee C, Chiueh T C. A comprehensive implementation and evaluation of direct interrupt delivery. In *Proc. the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Mar. 2015, pp.1-15.

[52] Chang Z, Li J, Ma R, Huang Z, Guan H. Adjustable credit scheduling for high performance network virtualization. In *Proc. the 2012 CLUSTER*, Sept. 2012, pp.337-345.

[53] Guan H, Ma R, Li J. Workload-aware credit scheduler for improving network I/O performance in virtualization environment. *IEEE Transactions on Cloud Computing*, 2014, 2(2): 130-142.

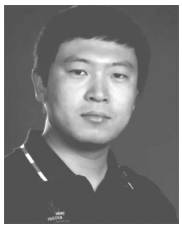
[54] Pfefferle J, Stuedi P, Trivedi A, Metzler B, Koltsidas I, Gross T R. A hybrid I/O virtualization framework for RDMA-capable network interfaces. In *Proc. the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Mar. 2015, pp.17-30.

[55] Younge A J, Walters J P, Crago S P, Fox G C. Supporting high performance molecular dynamics in virtualized clusters using IOMMU, SR-IOV, GPUDirect. In *Proc. the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Mar. 2015, pp.31-38.

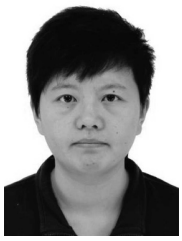
[56] Walters J P, Younge A J, Kang D I, Yao K T, Kang M, Crago S P, Fox G C. GPU passthrough performance: A comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL applications. In *Proc. the 7th IEEE International Conference on Cloud Computing* (*CLOUD*), June 27-July 2, 2014, pp.636-643.

Fan-Fu Zhou is currently a Ph.D. student in computer science and engineering at Shanghai Jiao Tong University, Shanghai. Prior to this, he received his M.S. and B.S. degrees in computer science in 2012 and 2008, respectively, both from Shanghai Jiao Tong University, Shanghai, and Huazhong University of Science and Technology, Wuhan, respectively. His main interest lies in computer network and security.

Ru-Hui Ma received his Ph.D. degree in computer science from Shanghai Jiao Tong University in 2011. Now he works as a postdoctoral researcher at Shanghai Jiao Tong University. His main research interests are in virtual machines, computer architecture and compiling.



Jian Li is an associate professor in the School of Software at Shanghai Jiao Tong University. He obtained his Ph.D. degree in computer science from the Institute National Polytechnique de Lorraine (INPL)-Nancy, France, in 2007. His research interests include real-time scheduling theory, network protocol design and embedded system. He is a member of ACM and IEEE.



Li-Xia Chen got her B.S. degree in computer science and technology from Nankai University, Tianjin, in 2014. She is now a Ph.D. student in computer science and engineering at Shanghai Jiao Tong University. Her main research interests include cloud computing and virtualization.

Wei-Dong Qiu got his M.S. degree in cryptography from Xidian University, Xi'an, in 1998, and Ph.D. degree in computer software theory from Shanghai Jiao Tong University in 2001. Before he came to Shanghai Jiao Tong University in 2004, he had been working as a postdoctoral researcher in Hagen FernUni of Germany (2001∼2003). He is now a professor and head assistant at the School of Information Security and Engineering, Shanghai Jiao Tong University. His main research area includes cryptographic theory, technology of network security and computer forensic.

Hai-Bing Guan received his Ph.D. degree from Tongji University, Shanghai, in 1999. He is a professor of School of Electronic, Information and Electronic Engineering, Shanghai Jiao Tong University, and the director of the Shanghai Key Laboratory of Scalable Computing and Systems. His research interests include distributed computing, network security, network storage, green IT and cloud computing.