Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

High performance network virtualization with SR-IOV

Yaozu Dong [a], Xiaowei Yang [a], Jianhui Li [a], Guangdeng Liao [a], Kun Tian [a], Haibing Guan [b,*]

[a] Intel Asia-Pacific Research and Development Ltd., China
[b] Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, China

ARTICLE INFO

Article history:
Received 17 July 2011
Received in revised form
1 December 2011
Accepted 29 January 2012
Available online 15 February 2012

Keywords:
Virtualization
Virtual Machine
SR-IOV
Xen
Performance

ABSTRACT

Virtualization poses new challenges to I/O performance. The single-root I/O virtualization (SR-IOV) standard allows an I/O device to be shared by multiple Virtual Machines (VMs), without losing performance. We propose a generic virtualization architecture for SR-IOV-capable devices, which can be implemented on multiple Virtual Machine Monitors (VMMs). With the support of our architecture, the SR-IOV-capable device driver is highly portable and agnostic of the underlying VMM. Because the Virtual Function (VF) driver with SR-IOV architecture sticks to hardware and poses a challenge to VM migration, we also propose a dynamic network interface switching (DNIS) scheme to address the migration challenge. Based on our first implementation of the network device driver, we deployed several optimizations to reduce virtualization overhead. Then, we conducted comprehensive experiments to evaluate SR-IOV performance. The results show that SR-IOV can achieve a line rate throughput (9.48 Gbps) and scale network up to 60 VMs, at the cost of only 1.76% additional CPU overhead per VM, without sacrificing throughput and migration.

1. Introduction

I/O performance is critical to high performance computer systems. With the rapid development of multi-core technology, computing capabilities keep increasing according to Moore's Law, but I/O performance is still suffering from both long latency PCI Express (PCIe) [21] and hardware scalability limitations, such as the limited number of PCIe slots. I/O intensive servers may waste CPU cycles, waiting for I/O data or spinning on idle cycles. This reduces system performance and sacrifices scalability.

Virtualization allows multiple OSs to share a single physical interface, maximizing the use of computer system resources. An additional software layer, called Virtual Machine Monitor (VMM) or hypervisor [25], is introduced to provide the abstraction of Virtual Machines (VMs), on top of which, each OS assumes owning resources exclusively. There are two approaches to enable virtualization. Paravirtualization (PV) [35] requires OS modification to work cooperatively with VMM. Full virtualization requires no modification, using hardware supports like Intel® VT [31]. Xen [1] is an open source VMM, which supports both paravirtualization and full virtualization. It runs a service OS in a privileged domain (domain 0) and multiple guest OSs in the guest domain.

Virtualization poses great challenges on I/O performance and its scalability. When a guest accesses the I/O device, VMM needs to intervene in the data processing to share the physical device. The VMM intervention leads to additional I/O overhead for a guest OS. Existing solutions, such as the Xen split device driver [8], also known as the PV driver, suffer from VMM intervention overhead, due to packet copy [28,23,13]. The virtualization overhead could saturate the CPU in high speed networks, such as 10 Gigabit Ethernet, impairing overall system performance. Several techniques are proposed to reduce VMM intervention. The Virtual Machine Device Queue (VMDq) [28] offloads packet classification to the network adapter and can put received packets directly into the guest buffer. However, it still needs VMM intervention for memory protection and address translation. Direct I/O assigns a dedicated device to each guest VM with I/O Memory Management Unit (IOMMU) translating DMA addresses from the guest programmed physical addresses to machine's physical addresses [6]. Direct I/O allows VM to directly access an I/O device, without VMM intervention. However, Direct I/O sacrifices device sharing and lacks scalability, two important capabilities of virtualization.

Single Root I/O Virtualization (SR-IOV) [21] proposes a set of hardware enhancements for the PCIe device, which aims to remove major VMM intervention for performance data movement, such as the packet classification and address translation. SR-IOV inherits Direct I/O technology, using IOMMU to offload memory protection and address translation. An SR-IOV-capable device is able to create multiple "light-weight" instances of PCI function entities, known

* Corresponding author.
E-mail addresses: eddie.dong@intel.com (Y. Dong), xiaowei.yang@intel.com (X. Yang), jian.hui.li@intel.com (J. Li), guangdeng.liao@intel.com (G. Liao), kun.tian@intel.com (K. Tian), hbguan@sjtu.edu.cn (H. Guan).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.01.020

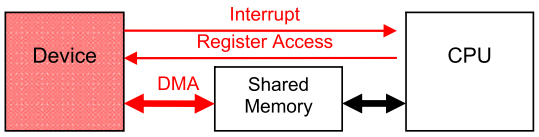

Fig. 1. The device interacts with processor through interrupt, register and shared memory.

as Virtual Functions (VFs). Each VF can be assigned to a guest for direct access, but still shares major device resources, achieving both resource sharing and high performance.

This paper proposes a software solution to efficiently use the SR-IOV-capable devices. The contributions of this paper are:

1. This paper designs and implements a generic virtualization architecture for SR-IOV-capable devices (SR-IOV virtualization, or SR-IOV architecture for short), which can be implemented on various kinds of VMM. It contains a VF driver, a Physical Function (PF) driver, and an SR-IOV Manager (IOVM). The VF driver runs in the guest OS as a normal device driver, the PF driver in service OS (domain 0 in Xen) to manage the PF, and the IOVM in VMM.
2. Three optimizations are applied to eliminate the virtualization overhead remaining in the SR-IOV virtualization. The remaining overhead includes: interrupt mask and unmask acceleration, virtual End Of Interrupt (EOI) acceleration, and adaptive interrupt coalescing.
3. A dynamic network interface switching (DNIS) scheme is proposed to address an additional challenge to virtual machine migration, introduced by the additional hardware stickiness between the VF driver and the SR-IOV-capable device. DNIS takes advantage of the OS bonding network driver mechanism and virtual hot plug support of the VF device, to switch the network interface between VF and software emulated NIC, for both performance and migration support.
4. Comprehensive experiments are conducted to evaluate the SR-IOV virtualization performance and scalability up to 60 VMs, including both paravirtualized virtual machine (PVM) and hardware virtual machine (HVM).

The results show that our new SR-IOV virtualization can achieve a 10 Gbps line rate and scale to 60 VMs, at the cost of 1.76% additional CPU overhead in a PVM and 2.8% in an HVM, without sacrificing throughput and VM migration. All of these results reveal that the new SR-IOV virtualization is a promising solution to high performance virtualization.

The rest of the paper is organized as follows: In Section 2, we give a more detailed introduction to SR-IOV, followed by the related work in Section 3. Section 4 describes the common SR-IOV architecture and some design considerations. Section 5 discusses the newly exposed virtualization overhead and optimizations to reduce them. Section 6 gives performance evaluation results. We conclude the paper in Section 7.

2. SR-IOV introduction

From the software point of view, a device interacts with the processor/software in three ways: interrupt, register, and shared memory, as shown in Fig. 1. Software programs the device through registers, while the device notifies the processor through asynchronous interrupt. Shared memory is widely implemented for the device to communicate with the processor for massive data movement through DMA [6].

SR-IOV is a new specification released by the PCI-SIG organization, which proposes a set of hardware enhancements to the PCIe device. It aims to remove major VMM intervention for performance data movement. SR-IOV inherits Direct I/O technology by using IOMMU to offload memory protection and address translation.

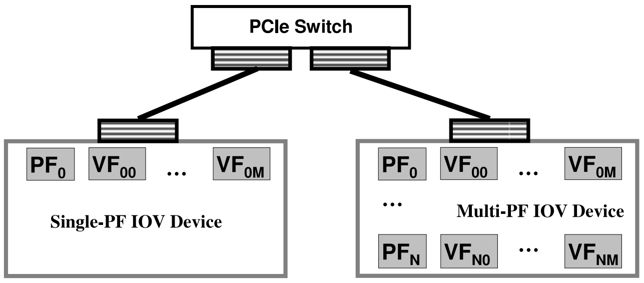

Fig. 2. SR-IOV-capable devices.

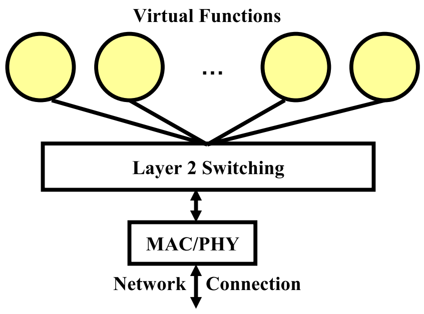

Fig. 3. Resource sharing in an SR-IOV-capable network device.

An SR-IOV-capable device is a PCIe device which can be managed to create multiple VFs. A PCIe function is a primary entity in the PCIe bus, with a unique requester identifier (RID), while a PCIe device is a collection of one or more functions. An SR-IOV-capable device has single or multiple Physical Functions (PFs), as shown in Fig. 2. Each PF is a standard PCIe function, associated with multiple VFs. Each VF owns performance-critical resources, dedicated to a single software entity to support performance data movement in runtime, while sharing major device resources, such as network physical layer processing and packet classification, as shown in Fig. 3. It is viewed as a "light-weight" PCIe function, configured and managed by PFs.

A VF is associated with a unique RID, which uniquely identifies a PCIe transaction source. RID is also used to index the IOMMU page table, so that different VMs can use different page tables. The IOMMU page table is used for memory protection and address translation in runtime DMA transactions. Resources for device initialization and configuration, such as PCIe configuration space registers implemented in conventional PCIe devices, are no longer duplicated in each VF so the device can have more VFs within the limited chip design budget, resulting in better scalability, compared with conventional multi-function PCIe devices [21].

3. Related work

A wide spectrum of studies have been conducted on both native and virtualization I/O, to understand and improve performance. Authors in [14,12,15] did extensive studies on native I/O processing and proposed a new I/O architecture to improve its performance. In the area of virtualization, network performance of the Vmware Workstation [30,29] has been studied. Xen [8] uses a shared-memory-based data channel to reduce data movement overhead. Page remapping and batch packets transferring could be used to further improve performance [17]. Areas of study include: cache-aware scheduler, virtual switch enhancement, and transmit queue length optimization [13]. They all suffer from the overhead of extra data copy. Xenloop [33] improves inter-VM communication performance using shared memory, but requires new user APIs. I/O latency caused by VM scheduling is mitigated with a variety of scheduler extensions [20], but cannot be fully eliminated.