分类号 TP311 学号 GS08061022

UDC 密级 公 开

工程硕士学位论文

SR-IOV 虚拟化技术的研究与优化

硕士生姓名 李超

学 科 领 域 计算机科学与技术

研 究 方 向 计算机应用

指 导 教 师 戴华东 副研究员

国防科学技术大学研究生院

二〇一〇年四月

Y1795829

Research and Optimization of SR-IOV Virtualization Technology

Candidate: Li Chao
Advisor: Associate Prof. Dai Huadong

A thesis
Submitted in partial fulfillment of the requirements
for the professional degree of Master of Engineering
in Computer Technology
Graduate School of National University of Defense Technology
Changsha, Hunan, P.R.China
April, 2010

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：SR-IOV虚拟化技术的研究与优化

学位论文作者签名：李超　　日期：2010年6月9日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：SR-IOV虚拟化技术的研究与优化

学位论文作者签名：李超　　日期：2010年6月9日

作者指导教师签名：戴维东　　日期：2010年6月9日

目 录

表 目 录

图 目 录

摘 要

经过近几年的发展，CPU 虚拟化与内存虚拟化技术已日趋成熟， I/O 虚拟化技术的发展却相对滞后，影响了虚拟机的整体性能。如何改进 I/O 虚拟化技术，提高 I/O 设备的虚拟化性能及其利用效率，一直是当前虚拟化技术的研究重点之一。

当今国际上主流的 I/O 虚拟化技术有三种：Split I/O，Direct I/O 及 Passthrough I/O，这些 I/O 虚拟化技术在不同程度上实现了 I/O 设备的虚拟化功能，并通过硬件级的支持不断提高 I/O 设备虚拟化的性能。但是由于体系结构和硬件支撑条件的限制，上述几种主流 I/O 虚拟化技术在性能方面仍然存在差距。其中，Passthrough I/O 支持客户域直接访问物理 I/O 设备，具有最高的性能，但是它只能将单个 I/O 设备分配给一个客户域独占使用，无法实现 I/O 设备在多个客户域之间的共享。为此，Intel 提出了最新的解决虚拟化 I/O 的 SR-IOV 技术，该技术不仅能够继承 Passthrough I/O 虚拟化方式的的高性能优势，同时还支持 I/O 设备的跨域共享，具有较好的应用前景。但作为一种新技术，SR-IOV 还不够完善，一是它支持的虚拟机数量有限，二是 SR-IOV 需要特殊的硬件支持，限制了其应用范围。

本文在深入研究和分析 SR-IOV 虚拟化技术的原理与实现的基础上，针对 SR-IOV 在可扩展性和依赖特殊硬件支持这两方面的不足，进行了改进，提出了一种扩展 VF 的共享模型 VFM 和一种无需特殊硬件支持的虚拟 SR-IOV 模型，并在 Linux 操作系统和 Xen 虚拟监控器中进行了实现。测试结果表明，改进的 SR-IOV 虚拟化技术具有较好的性能与可扩展性。

本课题得到国家 863 计划重大项目“高端容错计算机研制与应用推广”（课题编号：2008AA01A203）的支持。

关键词：虚拟化，I/O 虚拟化， Passthrough I/O ， SR-IOV

ABSTRACT

Through years of development in the field of virtualization, CPU virtualization and memory virtualization have developed to be quite advanced, however, in respect of I/O virtualization, the relatively poor performance of I/O device has affected the performance of virtualization as a whole. Thus currently in the field of virtualization research, experts have been focusing on how to improve I/O virtualization and the efficiency of physical device

Three kinds of I/O virtualization domain the international stream currently, they are Split I/O, Direct I/O and Passthrough I/O, all of which improve the performance of I/O devices in a degree, and gradually improve I/O devices virtualization performance by hardware support. But, limited by the design of its systematic structure and hardware limitations, the above three I/O virtualization technologies still have a long way to be mature in respect of performance. Among them, Passthrough I/O has the best performance to support client domain to visit I/O devices directly, but it can only allocate a single I/O to one client domain, instead of sharing the I /O devices in multi-client domain. Therefore, Intel brought up the latest SR-IOV technology to settle I/O virtualization, which not only inherits the high performance of Passrthrough I/O virtualization, but also support I/O devices cross-domain share, thus it has a bright future. But as a new technology, SR-IOV lacks itself from two aspects: one is the limited VF quantity; the other is SR-IOV needs special hardware support which limits its application area.

By deeply going into the theory and application of SR-IOV virtualization technology, and according to the two limitations of scalability and dependence on hardware support, the paper brought up a scalable VF share model (VFM) and a virtualized SR-IOV model without special hardware support, , and made it realizable in Linux operation system and Xen virtualization monitor. The result shows that the improved SR-IOV virtualization technology has a better performance and scalability.

The topic is supported by the National 863 Plan as a significant program "Research and Development of High-end Fault-tolerant Computer". (No.: 2008AA01A203)

Key words: Virtualization, I/O virtualization, Passthrough I/O, SR-IOV

术语列表

SR-IOV	Single Root I/O Virtualization	(单根 I/O 虚拟化)
VFM	Virtual Function Management	(虚拟功能管理)
IOMMU	I/O Memory Management Unit	(I/O 存储管理单元)
GOS	Guest Opration System	(客户操作系统)
HOS	Host Opration System	(宿主操作系统)
PF	Physical Function	(物理功能)
VF	Virtual Function	(虚拟功能)
VM	Virtual Machine	(虚拟机)
VMM	Virtual Machine Monitor	(虚拟机监视器)
HVM	Hardware Virtual Machines	(硬件虚拟机)
VT-x	Virtualization Technology For X86	(基于 X86 的虚拟化技术)
VT-d	Virtualization Technology For Directed I/O	(直接 I/O 的虚拟化技术)
PCI-e	Peripheral Component Interconnect - Express	(快速-外围组件互联)
ATS	Address Translation Service	(地址翻译服务)
MMIO	Memory Mapped I/O	(内存映射 I/O)
MSI	Message Signaled Interrupt	(消息中断)
BAR	Base Address Register	(基地址寄存)

第一章 绪论

随着计算机硬件技术、网络技术和应用需求的飞速发展，虚拟化技术已经渗透到信息系统的各个领域，并不断呈现出新的发展趋势。I/O 虚拟化是虚拟化技术的重要组成部分，其性能对系统整体性能的提高有至关重要的作用。然而，目前 I/O 虚拟化面临的一个重大挑战是：如何在虚拟化的情况下，获得良好的 I/O 性能并且有效地共享 I/O 设备。本章介绍 I/O 虚拟化的基本概念和特点，分析 I/O 虚拟化技术的研究现状，及其对虚拟化技术提出的挑战，阐述本文欲解决的科学问题以及所取得的主要成果。

1.1 课题背景及研究意义

近些年来，随着计算机硬件技术的飞速发展以及计算机体系结构的不断创新，计算机系统日趋强大的计算能力和相对落后的计算模式之间的矛盾日益突出，虚拟化技术能够在快速发展的硬件系统和复杂多变的应用需求之间找到新的平衡点。虚拟化技术能够为企业级应用提供更多的优势：如提高资源的利用率，降低管理成本，提高使用灵活性，增加系统安全性，带来更高的可用性、可扩展性和可操作性等。因此，虚拟化技术引起了国内外学术界、企业界的广泛关注，被视为改变IT产业现有格局的十大关键技术之一[1]。

虚拟技术发展十分迅速，据 IDC 统计，至 2009 年，在 x86 服务器上安装的虚拟机由 2006 年底的 54 万台猛增至 400 万台以上；IDC 同时预计，虚拟化微机数量也将从 2007 年的不足 500 万台增至 2011 年的 6.6 亿台。虚拟技术的应用也涉及如下领域：（1）服务器领域，包括数据中心、云计算、分布式计算、虚拟服务器等对硬件平台需求高的项目；（2）企业管理软件，包括基于虚拟机的可信桌面，方便有效地管理和支持员工桌面电脑；（3）个人用户，包括基于虚拟机的杀毒技术、程序的开发和调试、操作系统内核学习以及个人隐私攸关的应用。随着虚拟技术的进一步发展，其广泛的应用空间必将进一步被扩展，其有前景的应用有：服务器整合，虚拟化应用，云计算和数据中心，虚拟执行环境，，沙箱，系统调试、测试与质量评价等。

虚拟化技术通过对硬件和软件的划分或整合，部分或完全的对物理机器进行模拟或仿真，将计算资源合并或切分成一个或多个运行环境。虽然目前虚拟机依赖的硬件平台和实现细节多种多样，但基本上都是基于图 1.1 这个经典的虚拟机模型。虚拟机监控程序，或者叫虚拟机监视器[2]（Virtual Machine Monitor，VMM）介于物理硬件和虚拟机（或客户操作系统）之间，运行在特权模式，隔离并且管

理上层运行的多个虚拟机。它为每个虚拟机虚拟出一套独立于实际硬件的虚拟硬件环境（包括虚拟CPU，虚拟内存，虚拟I/O 设备等）。从应用程序的角度看，程序运行在虚拟机上与运行在其对应的真实计算机上一样。虚拟机技术使得一台物理计算机可以支持多个不同的虚拟机分别运行不同或相同的操作系统。

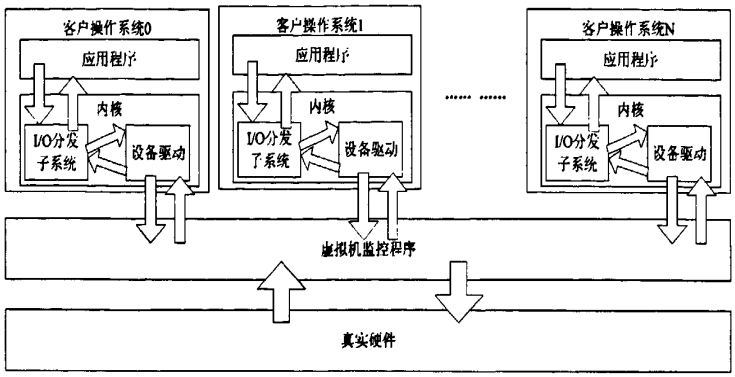

图 1.1 虚拟机中的 I/O 模型

从图 1.1 中可以看出，对于客户操作系统中的应用程序来说，它发起 I/O 操作的流程和在真实硬件平台上的操作系统是一样的。整个 I/O 流程有所不同的在于设备驱动访问硬件这个部分。对于不同的虚拟 I/O 模型，客户操作系统中的设备驱动有不同的访问方式。如采用设备模拟虚拟 I/O 模型，驱动访问硬件的方式和在真实硬件平台上完全一样。采用泛虚拟化设备驱动 I/O 模型，则必须使用特殊的驱动程序才能发起 I/O 操作。无论是哪种方式，客户操作系统的 I/O 请求最终必须由虚拟机监控器或处理器提供的硬件机制来截获并代理。虚拟机监控器会根据不同的虚拟 I/O 模型，采取不同的 I/O 分发策略。

在传统的体系结构中，I/O 是一个重要的环节，是影响系统性能的关键部分，在虚拟环境中更是如此。虚拟化技术发展的初期主要致力于CPU与内存的虚拟化，注重于CPU及内存性能的提高。随着硬件技术的进步，虚拟化技术在部件、系统及应用级都取得全面发展，Intel和AMD先后推出的一系列支持虚拟化技术的CPU，在很大程度上提高了虚拟机环境下CPU和内存的性能。然而在外围设备虚拟化方面，I/O设备的性能却损失很大。I/O设备的利用率成为了提高虚拟机使用性能的瓶颈。因此，提高在虚拟机下I/O设备的性能成为一个至关重要的问题。当客户操作系统直接面对真实的物理设备时，它的I/O性能可能是很高的，但是由于设备是有限的，所以更多的情况下，虚拟机中都是模拟的逻辑设备。逻辑设备对物理设备的I/O访问受到虚拟机模型中访问特权的影响，而且I/O请求从逻辑设备转换到物理设备又要经过一个复杂冗长的过程，延长了I/O访问的时间，使得虚拟机的I/O性能和传统操作系统的I/O性能存在很大的差距[3]。

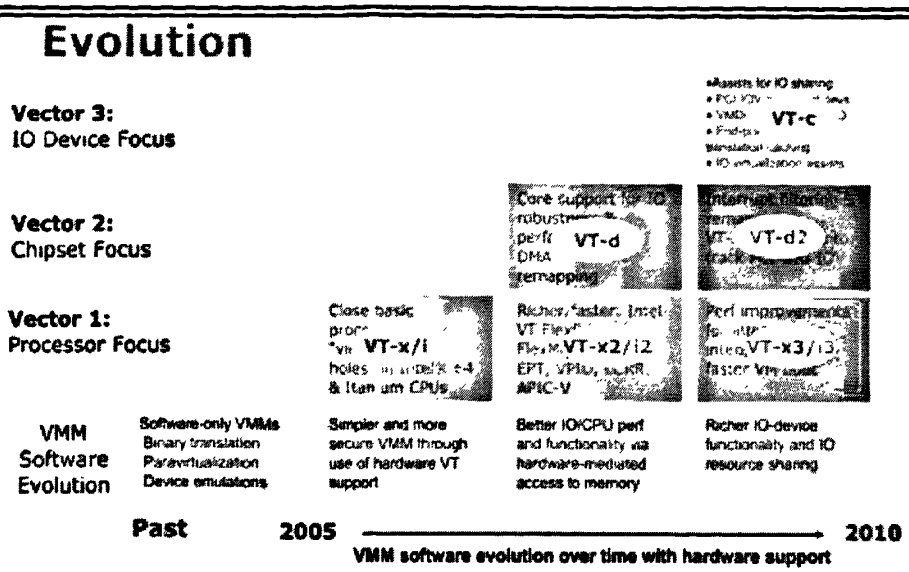

图 1.2 虚拟化技术的发展趋势

近年来，虚拟化的重心逐渐由软件级虚拟化技术、处理器级虚拟化支持、chipset级虚拟化支持向着I/O虚拟化硬件设备级支持的方向倾斜，而且由虚拟机实现的I/O功能也逐步移向硬件，如图 1.2 所示[4]。目前，I/O虚拟化技术是虚拟化技术重要的研究领域，其解决方法仍然涉及如下一些问题：

- 功能性：虚拟化 I/O 必须要完成相应的功能，从而达到资源的合理利用和重新分配的目的。
- 高性能：虚拟化 I/O 合理地利用了系统的 I/O 设备，但是在性能上也要尽可能接近真实 I/O 的性能。另外，在完成虚拟机的实现以后，专门的性能提升和优化也是必不可少的。
- 共享性：I/O 设备被多个虚拟机所使用，必须提高 I/O 设备的共享能力，为一个以上虚拟机提供服务。
- 安全性/隔离性：虚拟化技术是对传统的操作系统架构的颠覆，必然导致新的入侵方式和漏洞的产生。因此在设计虚拟化 I/O 的时候必须要周密地考虑。
- 透明性：虚拟机能够自动保存 I/O 设备的设备状态，能够为虚拟机中的每一个 I/O 设备提供驱动，为硬件升级提供必要的驱动支持。

本课题就是基于上述背景，重点针对 I/O 虚拟化的性能和共享性开展研究。

1.2 课题相关工作

正如服务器、微机的性能在很大程度上受限于 I/O 性能一样，虚拟化技术的性能也受限于虚拟化 I/O 技术。虚拟化服务由 CPU、内存、I/O 三部分组成，它们构

成了虚拟化物理计算机的三要素。随着虚拟化技术的不断发展，CPU 虚拟化与内存虚拟化逐渐成熟，I/O 性能成为限制虚拟机整体性能的瓶颈，I/O 虚拟化越来越受到人们的重视。如何利用底层有限的物理设备为运行于其上的多个虚拟机提供高效的 I/O 访问支持是 I/O 虚拟化所关注的首要任务。

目前针对虚拟化技术中对I/O相关的研究主要集中在几个方面[5]：

- 实现更好的 I/O 性能。能够实现硬件直接 I/O 访问对密集 I/O 负载来说是一个重要的目标。
- 实现 QoS 和分离性。必须控制每个客户操作相关的 I/O 传输量。
- I/O 的扩展性和多样性。一个好的 I/O 虚拟化平台能够给虚拟化服务器提供足够大的管道和连接多个网络的能力。
- I/O 迁移。当一个客户从一个物理服务器移到另一个时，最好能够保持客户的 I/O 连续性。

下面介绍几种典型的虚拟化 I/O 模型。

1.2.1 虚拟化 I/O 的基本方式

目前，虚拟化 I/O 方式主要有三种：设备接口完全模拟方式、前端/后端模拟方式和直接划分方式。

设备接口完全模拟方式是一种纯软件方式，通过软件精确模拟，实现与物理设备完全一样的接口，客户操作系统（Guest OS）驱动程序无须修改就能直接驱动虚拟的设备。

前端/后端模拟是指虚拟机监控器提供一个简化的驱动程序作为后端（即Back-End），客户操作系统中的驱动程序为前端（Front-End)。前端驱动将来自其他模块的请求通过与客户操作系统间的特殊通信机制直接发送给客户操作系统的后端驱动，后端驱动在处理完请求后再发回通知给前端。该方法可分为：Split I/O[6]（又称分离I/O）模型和Direct I/O[7][8]（又称直接I/O）模型。

直接划分是指直接将物理设备分配给某个客户操作系统，由客户操作系统直接访问I/O 设备（不经虚拟机监控器）。该方法又称为Passthrough I/O[9]（又称穿透I/O）模型。目前与此相关的技术有 IOMMU[10][11]（I/O存储管理单元）、Intel 的 VT-d[12]、PCI-SIG的SR-IOV 等。

这三种模型各有优缺点，后文中将对它们进行具体的分析。

1.2.2 虚拟化 I/O 的实现技术

Sun始终是处在虚拟化I/O的领先地位。CrossBow[13]是Sun在I/O虚拟化方面的开源项目，它主要致力于网络虚拟化和资源控制。该项目允许系统管理员将物理

网卡划分为多个虚拟网卡。这些虚拟网卡和真实网卡非常类似，而且像真实网卡一样进行管理。在共享的网卡上，虚拟网卡可以分得自己的优先权和带宽，却不会引起任何性能下降。虚拟网卡可以拥有自己的网卡硬件资源（Rx/Tx 环、DMA 通道）、MAC 地址、内核线程和队列等，它们专属于虚拟网卡，且在所有通信过程中不被共享。在采用Solaris Container的情况下，每个container可以分得一个虚拟堆栈实例（Stack Instance），以及一个或多个虚拟网卡，这样，一个虚拟网卡的通信可以完全与其他通信隔离，并能受到所使用带宽量方面的限制或保证。

Intel和AMD为避免纯软件虚拟化的不足，在 2005 年分别推出了全新的硬件辅助的虚拟化技术，它们是Intel的VT（Virtualization Technology）[14]技术和AMD的Pacifica[15]技术。硬件辅助的虚拟化技术在计算密集型应用、预测例外和系统调用方面的性能带来了大幅提升。目前Intel的VT技术包含CPU、内存和I/O三方面的虚拟化技术，其中虚拟化I/O技术中包括虚拟机设备队列（以下简称VMDq）[16][17]和I/O加速技术。

针对x86 对虚拟化技术支持的先天缺陷，Intel 在芯片级为对虚拟机提供支持的一种技术VT-x[18]。VT-x大大提升了虚拟机监控器对虚拟机的掌控灵活性和粒度，而且充分考虑了如何有效减小虚拟机的开销。VT-x提供了两个运行环境：根（Root）环境和非根（Non-Root）环境。根环境专门为虚拟机监控器准备，很像原来没有VT-x的x86，只是多了对VT-x 支持的几条指令。非根环境作为一个受限环境用来运行多个虚拟机。运行环境之间可以通过特定指令进行相互转化，同时也意味着I/O设备可以更加简便地在为不同运行环境提供支持。

Intel 公司于 2007 年推出硬件支持的 I/O 虚拟化技术，全称为直接 I/O 虚拟化技术（VT-d）。VT-d 技术是一种基于北桥芯片的硬件辅助虚拟化技术，通过在北桥内置提供 DMA 虚拟化和 IRQ 虚拟化硬件，实现了新型的 I/O 虚拟化方式。它主要提供两种功能：设备直接分配和 DMA 重映射。

2007 年 10 月，PCI- SIG发布了SR-IOV（Single Root I/O Virtualization，即单根I/O虚拟化）规范[19]，其中详细阐述了硬件供应商在多个虚拟机中如何共享单个I/O设备硬件。SR-IOV规定了PCI Express (PCIe)[20]规格的扩展，它促使各类系统镜像或用户随机存取物理I/O资源上的子集，从而获得更好的数据移动，达到基础硬件资源的共享。一个SR-IOV设备呈现单个或多个物理功能（Physical Function，简称PF），PF是标准的PCIe功能。每个PF可以有零个或多个虚拟功能（Virtual Function，简称VF），VF可以视为一个“轻量级”的PCIe功能，但至少支持大量的数据移动。

随着虚拟机和虚拟化I/O技术的发展，提高在虚拟机下的I/O设备性能就是要在设备共享和设备使用效率二者之间达到一个平衡[21]。同时，这二者又是矛盾的，

如果在多个虚拟机之间共享一个设备，那么设备的使用效率必然是很低的，因为设备要频繁的在多个虚拟机之间进行切换；如果某个设备只提供给一个虚拟机使用，虽然使用效率很高，但是设备的共享性没有得到体现。

1.3 研究内容和主要创新点

目前，在高速服务器中，I/O 子系统的性能已经成为整个系统的瓶颈，大大限制高性能服务的应用。为了能将 CPU 等待 I/O 子系统响应的时间利用起来，人们引入虚拟技术，在一台服务器上同时运行多个操作系统，以提高 CPU 利用率。在虚拟机环境下，虚拟 I/O 的性能是制约整个系统性能的关键。测试表明，虚拟环境下的纯运算任务已能获得接近无虚拟机环境下运算性能。相对而言，I/O 交互频繁的任务还远远滞后于无虚拟机环境下的 I/O 性能。

目前，虚拟化 I/O 面临的一个重大挑战是：如何在虚拟化的情况下，获得良好的 I/O 性能并且有效地共享 I/O 设备。随着 SR-IOV 技术规范的提出和硬件的逐步实现，虚拟化 I/O 技术进入了新的起点，SR-IOV 技术能够有效地实现 I/O 的高性能和有效共享，但同时也面临着如下一些挑战，具体包括：（1）目前虚拟化 I/O 的基本模型有哪些，其 I/O 利用率、CPU 使用率、共享能力和可扩展性各有什么特点，如何为 SR-IOV 提供有效的借鉴？（2）SR-IOV 规范从硬件方式能够有效解决 I/O 虚拟化问题，减化虚拟机监控器的设计和实现，现有操作系统和虚拟机如何支持 SR-IOV 规范，其实现原理是什么，它们之间需要哪些协同机制？（3）如何在虚拟机中更有效地利用 SR-IOV 技术，进一步提高 SR-IOV 设备的适用性，如何对现有实现技术做出改进？（4）SR-IOV 技术与已有的虚拟化 I/O 技术相比有哪些优势和不足，其性能测试结果怎样？

针对以上问题，本文将分别从三个方面对基于 SR-IOV 的虚拟化 I/O 技术进行了研究，从原理和具体实现方面对 SR-IOV 技术进行了深入的分析；针对 SR-IOV 技术的不足，提出了两种改进模型并进行了初步实现；通过 SR-IOV 网卡，测试了本文提出的改进模型的性能和共享性。

本文的主要工作和创新点包括：

- 研究了SR-IOV的规范、原理和技术要求，分析了SR-IOV目前在Linux2.6内核和Xen3.1[22]中的实现框架；
- 提出了一种扩展 VF 共享模型 VFM，并在 VMM 中进行了实现；
- 提出了一种改进的虚拟 PF/VF 通信模型，对该模型进行了分析，并进行了初步实现；
- 对 SR-IOV 实现模型的网络性能进行了测试，对测试结果进行了分析。

1.4 论文结构

本文主要工作是基于SR-IOV规范，研究了SR-IOV的高效设备访问原理和模型，分析了Linux内核和Xen对SR-IOV规范的支持，研究了SR-IOV设备在Linux和Xen中的VMM中的整体实现框架[23][24]。在Xen中提出了一种扩展VF共享模型VFM和一种改进的虚拟PF/VF通信模型，并进行了分析和实现，最后对SR-IOV的改进实现进行了评测。

本文共分六章，论文各章节的内容安排如下：

第一章：绪论。本章论述了近年来虚拟化 I/O 技术的发展及其面临的挑战，介绍了论文的相关工作，说明了课题的背景及研究意义，列举了论文的主要工作和创新点。

第二章：SR-IOV 虚拟化技术研究与分析。本章研究了当前主流的高效 I/O 访问模型，介绍了 SR-IOV 概念与整体结构。分析了 Linux 内核和 Xen 对 SR-IOV 规范的支持，研究了 SR-IOV 技术的优势与存在的问题。

第三章：一种扩展的 VF 共享模型。本章针对 SR-IOV 技术中 VF 数量受限的问题，提出了一种扩展共享虚拟功能的访问模型 VFM，并基于 Linux 和 Xen 平台进行了实现。

第四章： 虚拟 PF/VF 通信模型。本章针对 SR-IOV 的硬件设备依赖性问题，提出了一种虚拟 PF/VF 通信模型，并结合 Xen VMM 进行了实现。

第五章：测试与分析。本章建立了测试的软硬件环境，并通过 netperf 标准测试程序对本文实现的 SR-IOV 改进模型在网络性能方面进行了测试，对测试结果进行了分析。

第六章：总结与展望。对全文工作进行了总结，并探讨了下一步的研究方向。

第二章 SR-IOV 虚拟化技术研究与分析

随着计算机硬件的飞速发展，CPU 的速度已经越来越快，但 I/O 设备的发展一直滞后。在虚拟机环境下，虚拟 I/O 的性能是制约整个系统性能的关键。相对而言，I/O 交互频繁的任务还远远滞后于无虚拟机环境下的 I/O 性能。本章研究了当前基于软件和硬件的典型 I/O 虚拟化技术，分析了 SR-IOV 技术的产生背景和规范，剖析了 SR-IOV 技术的结构与具体实现，并对该技术的优势与不足进行了总结。

2.1 几种 I/O 虚拟化模型

本节对当前主流高效I/O设备虚拟化模型进行了研究，通过对比分析基于软件的I/O设备模拟技术（包括Split I/O、Direct I/O、Passthrough I/O），以及硬件辅助虚拟化技术[25]（VT-x、VT-d等），总结了虚拟化I/O技术的发展趋势。

2.1.1 基于软件的 I/O 虚拟化模型

基于软件的I/O虚拟化模型是将I/O硬件的逻辑部分移入到虚拟机中，模拟层位于客户机与底层硬件之间。根据模拟层的具体位置，软件模拟I/O技术大致分为特权的宿主机模拟和虚拟机管理器（以下简称VMM）模拟。宿主机模拟通常采用Split I/O技术，又称为前端/后端模拟；VMM模拟一般采用Direct I/O技术[26]。

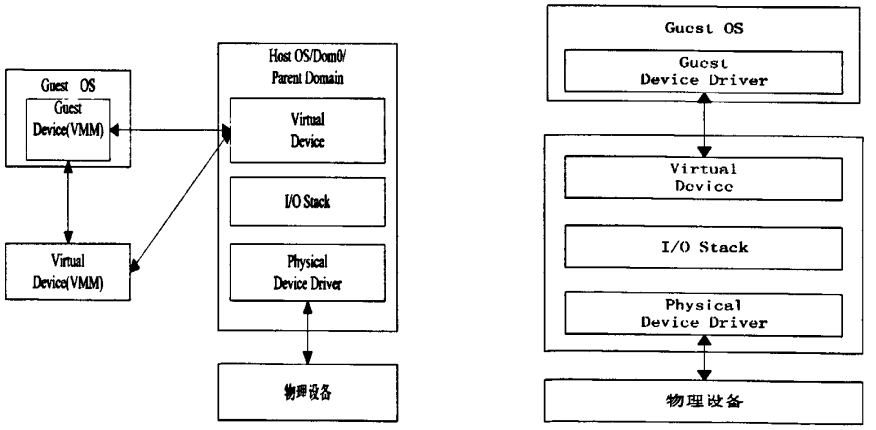

图 2.1 软件模拟技术：Split I/O 和 Direct I/O 模型

Split I/O 技术是指将传统的 I/O 驱动模型分成两部分：一部分在无 I/O 访问特权的虚拟机（简称 DomN）中，称为前端驱动，不能直接对 I/O 设备进行访问；另一部分则在有 I/O 访问特权的虚拟机（Dom0）中，称为后端驱动，可以调用物理 I/O 设备驱动，访问硬件。前端驱动接收 DomN 上层应用的 I/O 请求，通过事件通

道机制（多个虚拟机之间相互通讯的机制）传递给后端驱动。后端驱动处理前端驱动发送的请求，根据请求调用相应设备驱动程序对 I/O 设备进行访问，如图 2.1（a）所示。

Split I/O 通常采用高效的通信机制，这在很大程度上减少客户机之间上下文切换开销。但 Split I/O 也存在以下一些不足：（1）DomN 进行 I/O 操作都需要通过 VMM 转发请求，这会带来虚拟机与 VMM 之间的场景切换开销。因此，该方法会使整个 I/O 架构中的 Dom0 成为瓶颈；（2）在有大量的 I/O 请求时，更会带来巨大的客户机之间场景切换的开销，消耗大量 CPU 资源，影响虚拟 I/O 系统的效率；（3）为了实现 Split I/O， 需要修改客户端操作系统，以达到 VMM 和客户机之间的协同工作，因而无法支持非开源的操作系统。

DirectI/O 模型主要包括客户端驱动程序（相对于传统驱动不需修改）、设备虚拟层、I/O 数据传输的虚拟栈、VMM 中直接和底层设备交互的驱动程序以及物理设备。如图 2.1（b）所示，其中，设备虚拟层虚拟出各种 I/O 设备；I/O 虚拟栈将客户机的 I/O 地址映射到 VMM 的地址空间，处理 VMM 内部的通信，支持客户机与物理设备之间的 I/O 多路转发。当客户机发起 I/O 请求时，直接自陷到 VMM 中，以达到对物理设备的直接访问操作。但是由于 I/O 设备需要在多个客户机之间共享，因此需要通过 VMM 的介入以保证各个虚拟机对设备访问的合法性和一致性，这就导致了虚拟机的每次 I/O 操作都需要 VMM 的介入。

与 Split I/O 相比，Direct I/O 的实现方式更加有效和易于升级；兼容性方面 Direct I/O 模型更易于设备驱动程序的复用；故障隔离方面，Split I/O 通过将驱动部署在一个特定的客户机中来达到隔离性，而 Direct I/O 更容易发挥沙箱技术和其它技术的优势。但 Direct I/O 存在以下一些不足：（1）由于每次 I/O 访问都需要 VMM 的介入，对于 I/O 密集型访问或者多虚拟机同时进行 I/O 操作时，VMM 将迅速成为瓶颈，导致 I/O 延迟增大，I/O 效率大幅度降低；（2）驱动程序部署于 VMM 内部增加了 VMM 设计的复杂程度，同时难以移植设备驱动；（3）完成一次 I/O 操作需要涉及多个寄存器的操作，这要求 VMM 截获每个寄存器访问并进行相应的模拟，导致多次上下文切换，使得性能下降。

Passthrough I/O模型[27]是指在客户机内部能够直接对硬件进行操作，如图 2.2 所示。客户机与硬件的交互只需要经过少量、或者不需要经过VMM的管理。Passthrough I/O模型将设备独占式地分配给指定的客户域，使该域具有最高的I/O访问性能。这样做的优点是：由于不需要模拟设备进行请求转换，所以访问速度高；客户机能根据最新硬件，加载对应驱动，可充分发挥新硬件的功能。由于客户机可以内部直接的操纵硬件设备，这大大的提高了I/O性能。

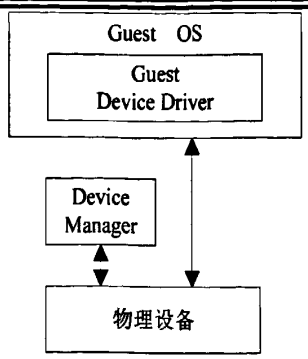

图 2.2 Passthrough I/O 模型

与基于软件的I/O虚拟化模型相比，Passthrough技术可以直接对I/O设备进行操作，大大降低了CPU的开销；Passthrough的I/O操作不需要经过一个有特权的客户机，不存在I/O瓶颈问题；Passthrough I/O模型不需要修改客户机驱动；Passthrough具有相对比较薄的VMM，VMM的设计相对简单，可移植性相对较好。但Passthrough I/O也存在一定的不足：（1）虽然Passthrough最大程度上提高的I/O性能，但这却是以牺牲设备共享能力为代价的；（2）Passthrough I/O中的设备只能被某个客户机所独占，难以充分发挥I/O设备的利用率[28]。

2.1.2 硬件辅助的模型

由于目前基于软件的 I/O 虚拟化模型增加了 CPU 的负担，人们自然希望借助硬件辅助技术完成一部分虚拟化 I/O 的功能，代表性的工作包括 Intel 和 AMD 公司的 VMDq、I/OAT、IOMMU、VT 等技术。

虚拟机设备队列（VMDq）是一个专门用于提升网卡虚拟化 I/O 性能的技术。VMDq 技术提供了分类/排序引擎实现了交换机的部分功能。为了提高网络吞吐量，它使用了堆栈缓存队列，支持 RSS 接收方扩展功能。支持 VMDq 的网卡能够通过 MAC 地址或者 VLAN 将报文发送到指定的虚拟机队列中去，VMM 只需要进行数据复制工作便可以完成虚拟交换机的任务，从而极大地提升了虚拟化网络服务效率。

I/O 加速技术（I/OAT）是提高数据在 I/O 设备、网络设备、存储器、处理器的加速技术，可以有效地提高系统性能。I/OAT 能够加快 CPU 中的协议堆叠，从而使芯片能够专注于本身的任务而无需花费太多的时间用于等待子系统所要求的网络流量。该技术还支持由芯片组执行的数据拷贝、数据和指令的并行处理、以及网络控制器的直接内存访问等功能。在测试环境下，集成于芯片的 I/OAT 功能在性能上要比独立网卡的性能提高 90%。

VT-x（Virtualization Technology for X86）技术引入了一种新的处理器操作，称为 VMX（Virtual Machine Extensions）。VMX 操作可以在根态（root）或非根

态（non-root）下执行，通常 VMM 在根态下执行而客户机软件则运行在非根状态。VMM 可以通过虚拟机入口（VM entry）使处理器进入到 VMX 非根态，相反，通过虚拟机退出（VM exit）可以回到 VMX 根态。在 VT 技术支持下，客户机可以运行在其原先希望运行的优先级（即优先级 0）上而 VMM 仍能捕获客户机访问特定的系统资源。

Intel 和 AMD 都在其处理器架构中提供了对Passthrough I/O（设备透传）的支持（以及辅助管理程序的新指令）。Intel 将这种支持称为VT-d（Virtualization Technology for Directed I/O），AMD 称之为IOMMU（I/O Memory Managemnt Unit）。这种技术的CPU提供将 PCI 物理地址映射到客户机的方法。当这种映射发生时，硬件将负责访问和保护，客户机像宿主系统一样可以直接使用该设备。除了将客户机映射到物理内存外，新的架构还提供隔离机制，以便预先阻止其他客户机（或管理程序）访问该区域[29]。

传统的 IOMMU 提供了一种集中的方式管理所有的 DMA—除了传统的内部 DMA，还包括 AGP、GARTner、TPT、TCP/IP 等这些特别的 DMA，它通过内存地址范围来区别设备，却不容易实现 DMA 隔离，因此 VT-d 通过更新设计的 IOMMU 架构，实现了多个 DMA 保护区域的存在，最终实现了 DMA 虚拟化，也叫做 DMA 重映射（DMA Remapping）。

I/O 设备会产生非常多的中断请求，I/O 虚拟化技术必须正确地隔离这些请求，并路由到不同的虚拟机上。传统设备的中断请求可以具有两种方式：一种是通过 I/O 中断控制器路由，一种是通过 DMA 写请求直接发送出去的 MSI（消息信号中断），由于需要在 DMA 请求内嵌入目标内存地址，因此该架构需要访问所有的内存地址，并不能实现中断隔离。VT-d 实现的中断重映射架构通过重新定义 MSI 的格式来解决这个问题，新的 MSI 仍然是一个 DMA 写请求的形式，不过并不嵌入目标内存地址，取而代之的是一个消息 ID，通过维护一个表结构，硬件可以通过不同的消息 ID 辨认不同的虚拟机区域。VT-d 实现的中断重映射可以支持所有的 I/O 源和中断类型，MSI 以及扩展的 MSI-X（扩展的信息信号中断）。

VT-d 技术可以隔离和保护硬件资源只给指定的虚拟机使用，硬件同时还需要具备多个 I/O 分区来同时为多个虚拟机服务。前文提到的 Passthrough I/O 模型在 VT-d 技术支持下可以顺利的实施并进行部署。

2.1.3 I/O 虚拟化技术对比

虚拟化技术通过在一台物理机器上增加工作量提高了硬件资源的利用率，但是随着硬件平台处理速度和性能的快速提升，许多平台未能充分利用；虚拟机的数量不断增加使得性能/功耗比变得愈加重要。

根据前面的分析，我们对基于软件的模型（包括 Split I/O、Direct I/O 和 Passthrough I/O）在功能性、性能、共享性、安全性和透明性上的比较如表 2.1 所示。

表 2.1 三种虚拟化 I/O 模型的比较

测试项	Split I/O	Direct I/O	Passthrough I/O
I/O 性能	良	差	优
CPU 开销	差	良	优
共享性	良	优	差
安全性	优	良	优
透明性（移植性）	差	优	良

虚拟化 I/O 解决方法需要提供隔离机制，确保客户机运行环境之间的隔离。隔离性应该保证内存空间隔离，也包括分离 I/O 流、中断、控制操作、I/O 操作和错误等。这需要提供相应的可扩展性，使得大量的虚拟机能够充分地利用 I/O 设备的空闲资源。同时，也要求虚拟化 I/O 设备能够减少 VMM 干预，降低 CPU 开销，提供接近本机的 I/O 操作性能。

从目前的虚拟化 I/O 的发展趋势可以看出，虚拟化 I/O 的许多实现技术由软件模拟方式向着硬件实现以及软硬件结合的方式发展，其目标是实现 I/O 设备的高性能、高利用率，同时保证 I/O 访问的安全性和 I/O 设备的共享性。

2.2 SR-IOV 概念与整体结构

传统的基于软件和硬件辅助的虚拟化 I/O 方法虽然能够从不同角度提高虚拟化 I/O 的能力，但无法同时获得 I/O 设备的高性能和共享性。SR-IOV 技术规范正是针对这一问题提出了相应的解决方法。

2.2.1 SR-IOV 概念

从软件的角度来看，I/O设备通过三种方式与CPU进行通信，分别是：中断、寄存器读写和存储共享。软件通过寄存器对设备进行操作，而I/O设备通过中断的方式通知CPU处理的情况。存储共享则通过DMA使I/O设备与CPU之间进行大规模数据通信[30]。

SR-IOV 是 PCI-SIG 组织公布的一个新规范，旨在消除 VMM 对虚拟化 I/O 操作的干预，以提高数据传输的性能。SR-IOV 继承了 Passthrough I/O 技术，通过 IOMMU 减少存储保护和地址转换的开销。

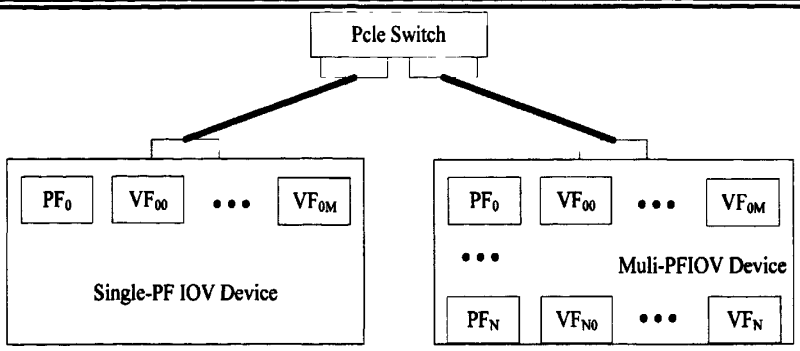

图 2.3 有 SR-IOV 能力的 I/O 设备

具有 SR-IOV 功能的 I/O 设备是基于 PCIe 规范的，可以用来管理并创建多个 VF（虚拟功能）。PCIe 功能在 PCIe 总线上是主要实体，具有唯一的申请标识 RID，一个 PCIe 设备具有一个或多个功能。SR-IOV 设备有一个或多个 PF（物理功能），如图 2.3 所示。每个 PF 都是标准的 PCIe 功能，并且关联多个 VF。每个 VF 都有与性能相关的资源，专门用于软件实体在运行时的性能数据运转，同时这些 VF 共享主要设备资源，如网络物理层处理和报文分类，如图 2.4 所示。因此，VF 可以视为由 PF 进行配置和管理的“轻量级”PCIe 功能。

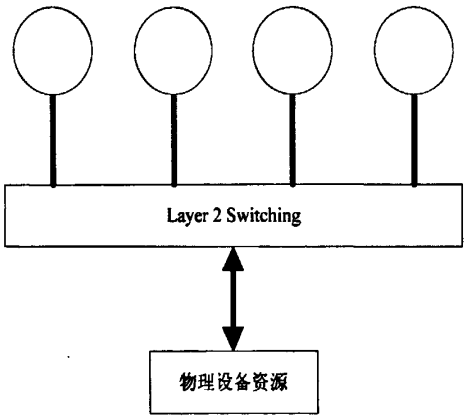

图 2.4 SR-IOV 网络设备的资源共享

每个VF对应唯一的RID，RID则确定了唯一的PCIe交换源。RID也能够用于索引IOMMU页表，因此不同的VM可以使用不同的页表。IOMMU页表是在DMA交换中用于存储保护和地址转换。设备初始化和配置资源（如用在传统的PCIe设备上的PCIe配置空间寄存器）没有应用在VF上，因此与传统的多功能PCIe设备相比，在有限的芯片设计预算里SR-IOV设备可以包含更多的VF，具有更好的可扩展性[31]。

SR-IOV 提出了地址翻译服务 ATS(Address Translation Service)，用以提高性能。ATS 需要每个 I/O 设备都使用地址传转换机制，通过本地 I/O TLB（I/O 转换

旁路缓冲）作为地址转换 Cache。这使得设备在传输之前就能够转换 DMA 地址，从而避免了 I/O TLB 在 IOMMU 地址转换过程中失效。

2.2.2 SR-IOV 的整体结构

SR-IOV 的整体结构包括 VF 驱动、PF 驱动、IOVM（SR-IOV 管理器）。VF 驱动是运行在客户机上的普通设备驱动；PF 驱动则部署在宿主机上对 VF 进行管理；在宿主机上的 IOVM 用于管理 PCIe 拓扑的控制点以及表示每个 VF 的配置空间；整体结构如图 2.5 所示。

为了使该结构独立于底层的 VMM，每部分都不能使用特定的 VMM 接口。例如，PF 驱动和 VF 驱动的通信可以直接使用 SR-IOV 设备，其接口不会依赖于特定的 VMM 接口。

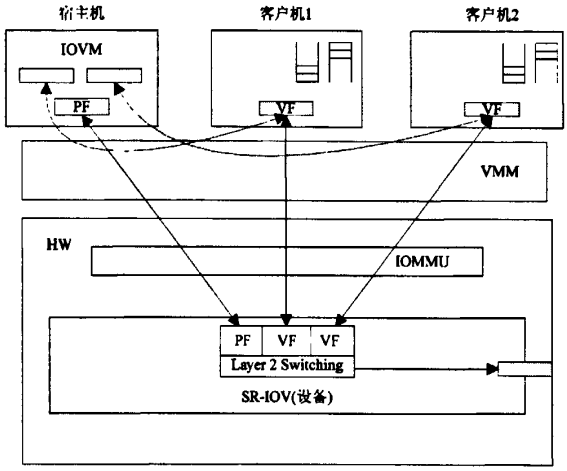

图 2.5 SR-IOV 的整体结构

一、PF 驱动

PF 驱动可以直接访问 PF 的所有资源，并负责配置和管理所有 VF。它可以设置 VF 的数量，全局的启动或停止 VF，还可以进行设备相关的配置，比如网络 SR-IOV 设备可以配置 MAC 地址和虚拟 LAN 环境。PF 驱动同样负责配置第 2 层的交换，以确保从 PF 或 VF 进入的报文可以正确地路由。在广播和多播的情况下，设备会为每个 PF 或 VF 复制相应的报文。

二、VF 驱动

VF 驱动像普通的 PCIe 设备驱动一样在客户机上运行，直接访问特定的 VF 设备完成数据的转移，期间不需要 VMM 的干预。从硬件成本角度来说，VF 只需要模拟临界资源（如 DMA 等），而对性能要求不高的资源则可以通过 IOVM 和 PF 驱动进行模拟。

三、IOVM

IOVM 为每个 VF 分配了完整的虚拟配置空间，因此客户机能够像普通设备一样模仿和配置 VF。当宿主机初始化 SR-IOV 设备时，它无法通过简单的扫描 PCIe

功能的供应商 ID 和设备 ID 列举出所有的 VF。由于 VF 只是修改过的“轻量级”功能，不包含完整的 PCIe 配置空间，因而无法响应普通的 PCI 扫描。该模型可以使用 Linux PCI 热插拔 API 动态地为宿主机增加 VF，然后将 VF 分配给客户机。IOVM 只是概念模型，负责对 PCI 设备进行扫描、识别，可以将 VF 转换成完整的功能，并对 SR-IOV 资源进行分配。

由于 IOVM 为每个 VF 分配了虚拟的完整的配置空间，一旦发现 VF 并分配给客户机，该客户机能够像使用普通 PCIe 功能那样初始化和配置该 VF。这部分功能可以在应用层面上完成，如 Xen 中 HVM 的设备模式，内核中的后端驱动。

影响SR-IOV性能关键是用于处理I/O（网络）设备的中断。在SR-IOV中，VMM不再干预I/O操作，从而提高了I/O（网络）设备的性能。2 层分发根据MAC和VLAN地址对报文进行了分类，通过DMA直接将报文存储到接收方的缓冲区上，并产生MSI或者MSI-x[32]。IOMMU重映射了接收方DMA缓冲器地址，将VF驱动的客户机物理地址转换为物理地址。VMM可以捕捉该中断并根据向量识别到具体的客户机，这种方式通过全局分配从而避免中断冲突。然后，VMM将虚拟MSI中断通知给客户机，并读出在当地缓冲器里的报文。客户机VF驱动处理虚拟中断，从本地缓冲区中读取接收的报文。从而一次客户机中断能够处理多个接收的报文。

PF和VF驱动之间需要为配置、管理信息和事件发送进行通信。例如，VF驱动需要将客户机的请求发送给PF驱动，比如设置多播地址和VLAN。PF驱动也需要将一些I/O（网络）事件转发给每个VF驱动，通知资源状态的变化。这些事件包括等待全局设备重置、链接状态改变、驱动移除等。目前，SR-IOV规范并没有规定PF/VF之间的通信方式，它们驱动之间的通信可以由底层硬件平台或VMM实现。例如，Intel的 82576 Gbit[33]网卡通过简单的邮箱和门铃机制进行通信：发送方将消息写入邮箱，然后按响“门铃”，这将会产生一个中断并通知接收方消息已经发送。接收方接收信息后，将共享寄存器中的一位进行设置，表明信息已接收。

SR-IOV 提供一个安全的运行环境，允许 PF 驱动监控和实施 VF 设备的带宽分配、中断屏蔽、拥塞控制、广播和多播等，从而增加了 VM 之间的性能和安全隔离。PF 驱动监督 VF 驱动的请求，并对 VF 驱动行为和其使用的资源进行监控。PF 如果发现异常可以立即采取正确的行为。例如，PF 在发现异常时能够停止分配给某个虚拟机的 VF。

2.3 SR-IOV 的功能模块

SR-IOV 规范使虚拟化 I/O 技术同时获得 I/O 设备的高性能和共享性，具有良好的发展前景。目前，SR-IOV 技术在操作系统和虚拟机环境中的实现方面还没有广泛应用。本章分析了 Linux 内核和 Xen 对 SR-IOV 规范的支持，研究了 SR-IOV

设备在 Linux 和 Xen 中的 VMM 中的整体框架，分析了 SR-IOV 设备的启动方法和基本运行方式。

2.3.1 实现环境简介

Linux 内核 2.6 版本中增加了对 SR-IOV 设备的支持，主要涉及的文件包括 drivers/pci 目录下的 iov.c，probe.c 等，提供对 SR-IOV 设备操作的 API，包括启动、停止、迁移、设置 VF 数目等。

Xen 本身主要基于开源的Linux核心代码移植而来，是一个开放源代码虚拟机监视器，目前最新的版本是 3.4.2。在Xen中，客户机被称之为虚拟域（Domain），其中 0 号虚拟域为服务域作为监控程序的扩展提供系统的管理服务。监控程序拥有部分硬件I/O资源，如定时器设备、中断设备等，其他虚拟域也可以拥有部分的I/O资源如硬盘、网卡等。拥有物理设备的虚拟域称为隔离设备驱动域（Isolated Driver Domain）或简称设备驱动域（Driver Domain）[34][35]。Xen具有如下一些特点：

（1）支持完全虚拟化和泛虚拟化：Xen 在客户机和硬件之间插入一层，用于模拟完整的虚拟硬件平台，客户操作系统不必修改，就可以在这层虚拟硬件平台上运行；同时，Xen 也支持泛虚拟化，不需要特殊硬件的支持，但是需要客户操作系统配合 VMM 做一些修改。

（2）支持 I/OAT、VT-x，VT-d 等硬件虚拟化技术：随着 Intel 公司和 AMD 公司对硬件虚拟化技术逐渐重视，已经提供了 IOMMU，DMA 重映射，MSI 等技术。Xen 也不断为这些虚拟化 I/O 技术提供必要的支持。

（3）支持Split I/O、Direct I/O和Passthrough I/O模型：在Split I/O方面， Xen 常规的块设备驱动程序分成前端驱动程序（Front-end Driver）和后端驱动程序（Back-end Driver）[36]。前端设备驱动程序在接到来自操作系统的读写请求时，通过事件通道和共享内存向位于隔离驱动域的后端设备提出服务请求。后端设备在收到服务请求后，则通过该虚拟机上的原生设备完成读写请求；在Direct I/O方面，客户机的I/O访问（含IOMMU）被Xen捕获，由设备模型（Device Model）进行仿真。从客户机操作系统的角度看，仿真设备同真实的物理设备并没有区别；在Passthrough I/O方面，Xen将DMA操作的客户机地址转换成物理地址的方法实现物理设备的直接内存存取。Xen硬件虚拟机采用的硬件设备虚拟化技术，通过VMM向硬件提供客户机DMA物理内存到机器物理地址的映像页表，实现未经修改的操作系统对直接分配设备的访问支持。

2.3.2 整体技术框架

SR-IOV 技术在 Linux 内核和 Xen 的应用主要是对内核和 QEMU 进行了相应的修改，图 2.6 概括了 SR-IOV 虚拟技术的整体结构。SR-IOV 虚拟化 I/O 模型大体上分为四个部分：SR-IOV 硬件初始化，宿主机对 VF 的设置，抽象 SR-IOV 的 VF 设备，客户机 N 的 VF 驱动初始化。

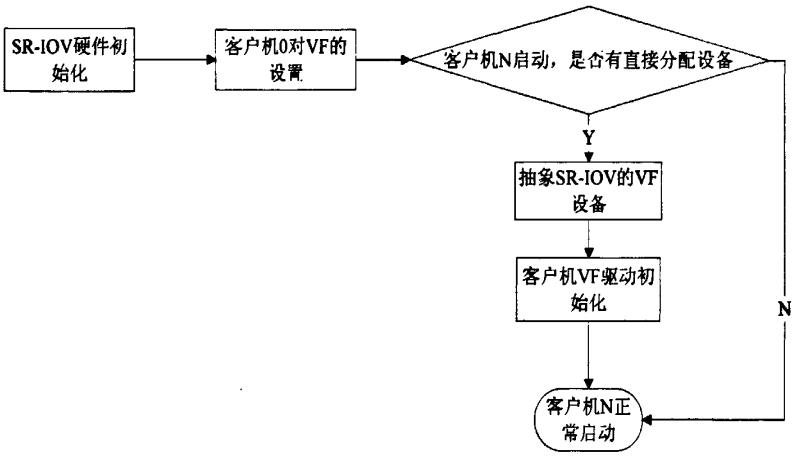

图 2.6 SR-IOV 虚拟技术的实现结构

使用 SR-IOV 需要有硬件支持，因此在 BIOS 选项中应该包含 SR-IOV 项，该项打开后系统启动就进入 SR-IOV 硬件初始化模块；宿主机分配 SR-IOV 设备，并完成 VF 设备隐藏功能；启动客户机 N 时可以选择直接分配设备或不分配设备，若给客户机 N 直接分配 VF 设备，则进入抽象 SR-IOV 的 VF 设备模块，以及客户机 N 的 VF 驱动初始化。整个流程的具体说明如下：

一、SR-IOV 硬件初始化模块

该模块主要完成 SR-IOV 硬件的探测和初始化，是 Xen 中最先加载的模块。由于 VMM 是 XEN 启动时最先启动的部分，拥有对所有设备的控制权，也包括了 SR-IOV 初始化模块。在该部分完成 SR-IOV 的设备探测、初始化根条目表、初始化上下文条目表、初始化 IOTLB 和初始化 SR-IOV 设备的工作。

二、宿主机对 VF 的设置模块

该模块主要给宿主机分配 SR-IOV 设备，并完成 VF 设备隐藏功能。宿主机可以利用设备直接分配功能把 VF 分配给某个客户机专属。这就要求运行在 VMM 上的其它客户机不能访问 VF，包括宿主机也不能访问。只要对宿主机隐藏了 VF，则 VF 对其它客户机也不可见，因为它们使用的 I/O 模型必须依靠宿主机才能运行。该模块依据此原理，对宿主机隐藏设备。同时，启用 SR-IOV 技术的平台，所有 DMA 操作都要经由 SR-IOV 硬件而到达指定内存，该模块需要为宿主机分配设备、配置 SR-IOV 硬件。

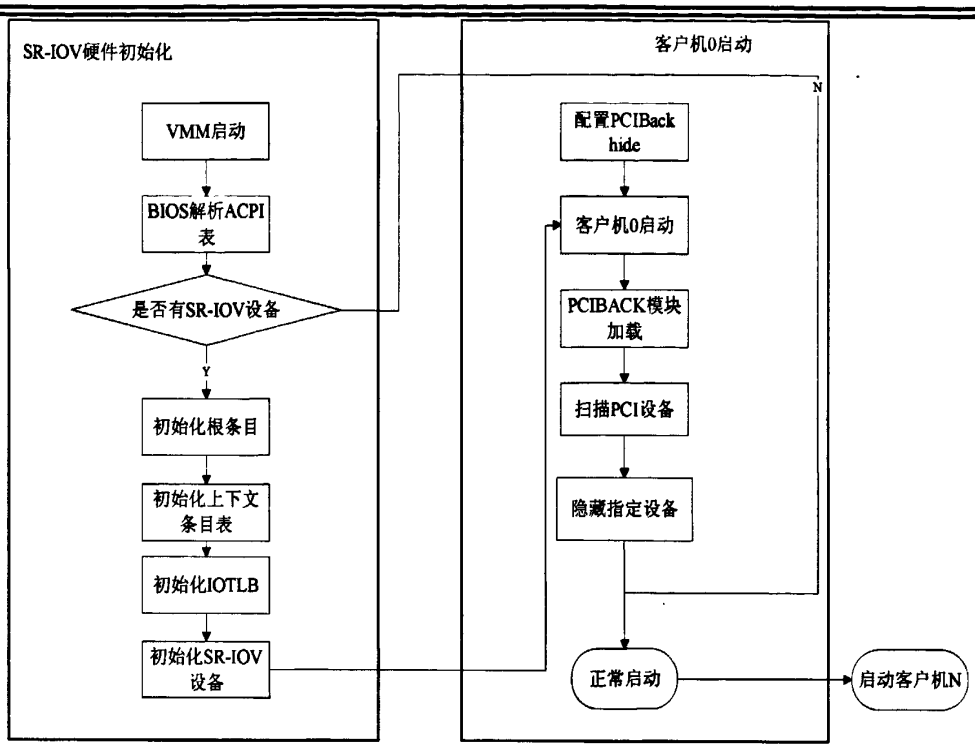

图 2.7 SR-IOV 硬件初始化和 Dom0 启动

三、抽象 SR-IOV 的 VF 设备模块

该模块使用了 Passthrough 模型，Passthrough 设备是在 QEMU 设备模拟器中新增的一类设备，它的作用是抽象隐藏的硬件，为此类设备分配标识符、获取 PCI 配置空间信息、生成 BAR 映射。Passthrough 设备是在客户机 N 启动的过程中，经由 QEMU 设备模拟器创建的，可以看成是客户机 N 启动过程中的一部分。在完成自身的主要功能后，它把 BAR 映射的信息传递给 VMM，以创建相应的端口 I/O（PIO）、内存映射 I/O（MMIO）转换表。

四、客户机 N 的 VF 驱动初始化模块

该模块主要为直接分配设备的客户机准备相应的数据结构和分配设备。客户机 N 是指直接拥有 VF 的使用权(即 VF 被直接分配给客户机 N)，通过 Passthrough 虚拟技术模型进行 I/O 操作的客户机。与普通的客户机启动流程不同，客户机 N 需要一张端口 I/O 转换表、MMIO 转换表，以及 DMA 重映射硬件进行地址转换需要的 I/O 页表。端口 I/O 转换表或 MMIO 转换表是与 Passthrough 设备产生的 Bar 映射相对应，主要用于客户机 N 通过设备 PCI BAR 发起 I/O 操作时做虚拟 PCI BAR 到真实 PCI BAR 的转换。这些数据结构都必须在客户机 N 启动前创建好。

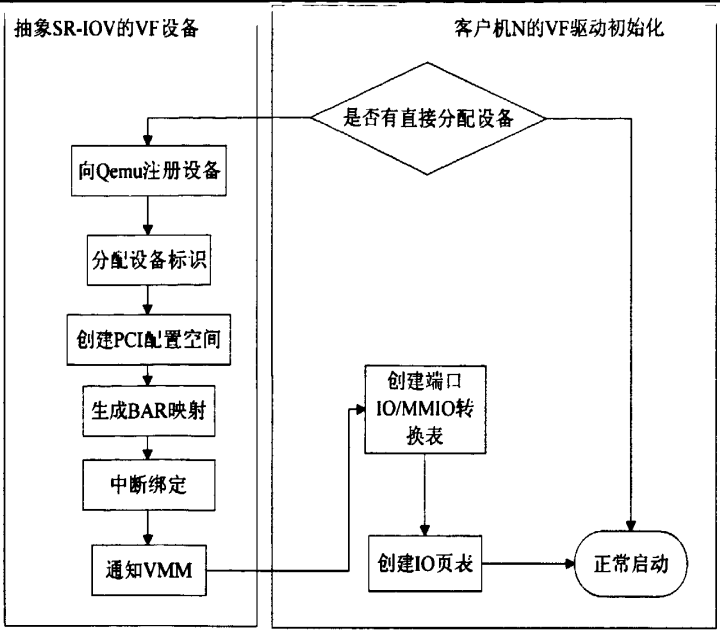

图 2.8 抽象 SR-IOV 的 VF 及其初始化

2.3.3 SR-IOV 操作与配置方式

利用 SR-IOV 的 API，通过以下代码就可以开发出具体 SR-IOV 设备的基本操作函数。对于具体的 SR-IOV 设备，探测设备的初始化函数可以调用启动 SR-IOV 功能的 API；注销设备函数可以调用停止 SR-IOV 功能的 API。最后，通过设置 pci_driver 结构，为该 SR-IOV 设备指定具体的操作函数。

```
static int __devinit dev_probe(struct pci_dev *dev, const struct pci_device_id *id)
{
    pci_enable_sriov(dev, NR_VIRTFN);
    ...
    return 0;
}
static void __devexit dev_remove(struct pci_dev *dev)
{
    pci_disable_sriov(dev);
    ...
}
static int dev_suspend(struct pci_dev *dev, pm_message_t state)
{    ...
    return 0;
}
static int dev_resume(struct pci_dev *dev)
{    ...
```

```
    return 0;
}
static void dev_shutdown(struct pci_dev *dev)
{ ...
}
static struct pci_driver dev_driver =
{   .name =      "SR-IOV Physical Function driver",
    .id_table =  dev_id_table,
    .probe =     dev_probe,
    .remove =    __devexit_p(dev_remove),
    .suspend =   dev_suspend,
    .resume =    dev_resume,
    .shutdown = dev_shutdown
};
```

为了充分利用 SR-IOV 设备的虚拟化功能，Xen 启动了 SR-IOV 功能并创建多个 VF，利用 Passthrough 方式将 VF 分配给客户机。例如，对于 Intel 82576 网卡，可以采用如下步骤进行设置：

1. 启动网卡的 SR-IOV 功能；
   在 Linux 内核启动选项中设置 Add “pci=assign-busses”
2. 创建 VF（3 个）；
   # modprobe igb max_vfs=3
3. 利用 PCIBack 隐藏 VF；
   # echo –n 0000:02:10.0 > /sys/bus/pci/drivers/pciback/new_slot
   # echo –n 0000:02:10.0 > /sys/bus/pci/drivers/pciback/bind
4. 设置 VF 的网络带宽；
   # echo “600 300 100” > /sys/class/net/eth1/device/　　/*设置网络带宽的分配为 600 300 100 */
   bandwidth_allocation
   : 600, 300 and 100 are throughput for 1st Virtual Function
   : (VF#1), VF#2 and VF#3 respectively by Mbps
5. 利用 Passthrough 方式将 VF 分配给不同的客户机。

2.4 SR-IOV 的优势与不足

2.4.1 SR-IOV 的优势

SR-IOV 平台提供了一系列的技术优势，包括提高 I/O 性能，提高系统的性价比，增加可扩展性，数据保护和安全性等。

一、提升系统性能

Passthrough 方式的优势：（1）VF 设备可以直接访问寄存器，IOMMU 技术使得 GPA（客户机物理地址）转换为宿主机物理地址，这些使得客户机几乎可以达到本机的性能。与软件模拟的 I/O 设备相比，每个虚拟机能够通过较低的 CPU 开销获得很高吞吐量。（2）传统的陷入和模拟 I/O 寄存器的读写和任务切换占用了大量的 CPU 利用率。VF 的另一个优势是可以直接进行 I/O 寄存器的读写，而不需要陷入和模拟，CPU 页表机制可以直接将 VF 设备的 MMIO 空间映射到客户机上面。

中断重映射的优势：IOMMU 技术可以改善中断重映射技术，减少客户机从硬件中断到虚拟中断的处理延迟。由于中断延迟是虚拟环境的主要瓶颈之一，采用 IOMMU 的 MSI-x 技术将大大减少中断延迟，降低了由 VMM 处理 I/O 导致的系统开销，提高系统性能。

共享性的优势：Passthrough 技术将设备分配给指定的虚拟机，可以达到几乎本机的性能，其缺点是整个 I/O 设备只能供一个虚拟机使用。这种方式违背了虚拟化的本意，即 I/O 资源的共享是为使得硬件利用的最大化。SR-IOV 技术能够使分配给每个虚拟机的 VF 都达到其最高性能，这使得所有虚拟机能够充分利用 I/O 设备资源，达到该设备的最高性能。

二、减轻系统管理员负担

减少物理设备的优势：使用 VF 替代多个物理 I/O 设备，降低硬件开销，简化布线，降低功耗，减少转换器端口的使用数目，降低设备数目。

安全性优势：通过硬件辅助数据保护和安全得到了加强，使得数据和 I/O 流在虚拟机之间的创建和隔离得到了增强。

可扩展性优势：系统管理员可以使用单个更高带宽的I/O设备（例如 10Gb以太网卡）代替多个带宽较低的设备（比如 1Gb以太网卡）达到带宽的要求。利用VF将带宽进行隔离，使得网络资源好像是隔离的物理网卡。此外，这还可以为其他类型的设备节省了插槽[37]。

三、简化虚拟机设计

通用性优势：SR-IOV 技术不需要在客户机中安插任何前端驱动，也不需在 VMM 中维护任何后端驱动，没有额外的维护开销。

减少对宿主机依赖：SR-IOV 技术不依赖宿主机进行 I/O 操作，所以当运行的客户机数量多时，不会增加宿主机的负荷。

2.4.2 SR-IOV 存在的问题

作为一种新技术，SR-IOV 技术难免还存在一些问题，其中包括：

（1）扩展性不足。SR-IOV 通过直接分配获得几乎本机的性能，但是直接分配也丧失了软件模拟、虚拟化软件的灵活性和兼容性。由于 VF 的数量有限，造成直接分配给客户机的设备数量受限，当多个客户机对设备的需求大于 SR-IOV 设备提供的 VF 数量时，无法充分发挥设备的共享能力。针对这一问题，本文第三章提出了一种扩展 VF 共享模型。

（2）需要特殊硬件支持。SR-IOV 技术需要特殊的硬件设备支持，如普通网卡不能借助 SR-IOV 技术带来的优势，本文的实验就是基于一种特殊的 Intel 82527 网卡对 SR-IOV 技术进行验证。为了提高 SR-IOV 技术的通用性，有必要开发一种机制使得客户机能够与 VMM 协作，在硬件平台不具备 SR-IOV 能力的情况下，通过软件模拟的方式来支持 SR-IOV 模型。针对这一问题，本文第四章提出了一种虚拟 PF/VF 通信模型。

第三章　一种扩展的 VF 共享模型

SR-IOV 技术融合了 I/O 设备模拟和 Passthrough 方式的优势，具有良好的可扩展性、共享性和高性能的特点，为虚拟化 I/O 的发展提供了有效的硬件支持。但随着对虚拟化 I/O 技术需求的快速增长，SR-IOV 技术也面临着一些问题，比如 SR-IOV 的 VF 数目可能少于虚拟机的数量，那么如何才能充分发挥 SR-IOV 技术的优势？本章将针对这个问题提出一种扩展的 VF 共享模型，并阐明了其实现原理和实现框架。

3.1. 问题分析

尽管一些设备支持 SR-IOV 规范，但设备提供的 VF 的数量仍然是有限的。这引发了一个问题：如果虚拟机的数量大于 VF 的设备，如何有效地利用该设备，使得 SR-IOV 设备能够提供高性能和良好的可扩展性？

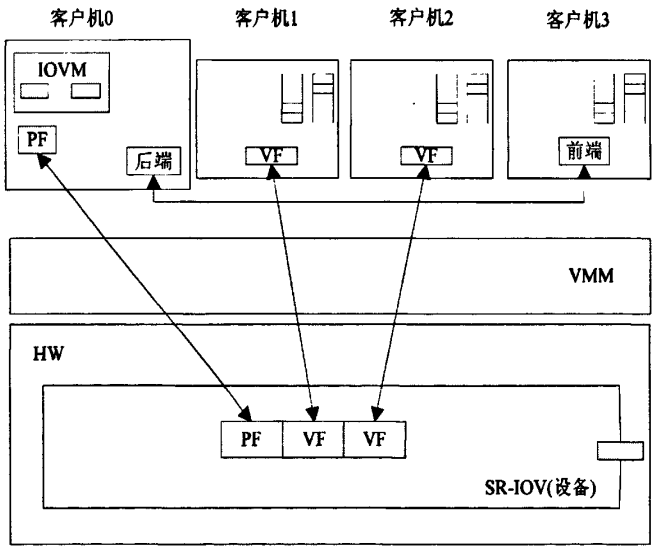

图 3.1 前端/后端模型使用 SR-IOV 设备

一种比较直观的方法是，将一些对性能要求较高的客户机利用 Passthrough 方式直接使用 VF，其他对性能要求较低的客户机则利用 Xen3.0 的虚拟化 I/O 结构，采用前端/后端的方式访问 PF，具体的方法如图 3.1 所示。其中客户机 1 和客户机 2 通过 Passthrough 方式直接访问 SR-IOV 设备的 VF，而客户机 3 则通过前端/后端驱动的方式访问 SR-IOV 设备的 PF。即当客户机 3 对前端驱动接管的设备发起一次 I/O 操作时，前端驱动通过 VMM 提供的 Hypercall 利用事件通道通知运行在宿主机内核中的后端驱动。后端驱动在收到通知后，调用同样运行在宿主机内核中

的 SR-IOV 设备驱动 PF，发起真实的 I/O 操作。当后端驱动取得了 I/O 操作的结果后由事件通道通知 I/O 操作完成，前端驱动从缓冲区取得结果并返回给客户机 3。

这种方法的实现较为简便，在 Linux 和 Xen 中很容易进行配置和实施，但存在三方面不足：

- VF 被特定的虚拟机独占，但 VF 资源在大多数时候是处于闲置状态的，浪费了设备的大量带宽和物理资源；
- 其它虚拟机不能充分利用 SR-IOV 的优势，无法有效地共享设备，造成较低的 I/O 操作处理效率；
- 虚拟机难以有效地利用 Xen 提供的机制进行动态迁移操作，降低了虚拟机的可扩展性。

3.2 模型和原理

针对上述方法的不足，我们提出一种扩展 VF 共享模型，该模型可以使宿主机上的所有虚拟机都有平等的机会共享 SR-IOV 设备的 VF，充分利用 SR-IOV 设备的物理资源，避免了需要使用 SR-IOV 设备的 VF 的客户机处于闲置状态，有效增加了 I/O 利用率。扩展 VF 共享模型的基本思想是：对于申请使用 SR-IOV 设备 VF 的客户机进行排队，优先级高的客户机可以使用 VF，优先级低的客户机将处于等待状态。该模型根据客户机的等待时间动态地调整客户机的优先级，并剥夺优先级低的客户机使用 VF 的权限，将剥夺后的 VF 分配给优先级高的客户机，从而达到有效利用 VF 的目标。

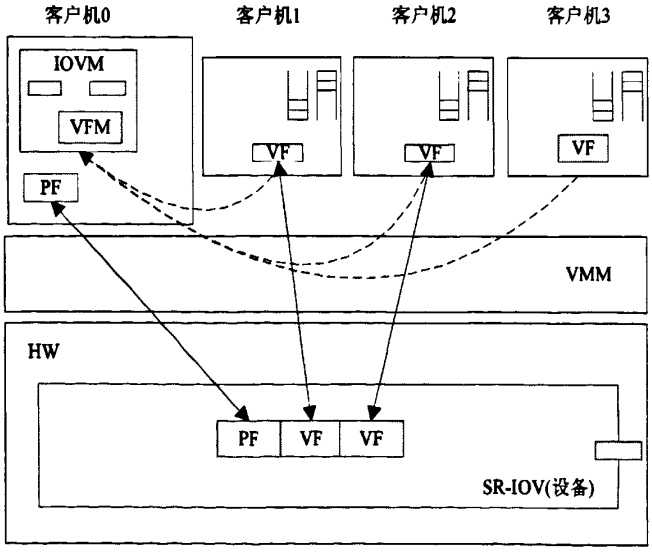

图 3.2 扩展 VF 共享模型使用 SR-IOV 设备

扩展 VF 共享模型的基本原理如图 3.2 所示。假设图中的 SR-IOV 设备具有两个 VF，虚拟机上共有三个使用 VF 的客户机。我们在宿主机的 IOVM 中增加了一

个模块 VFM（Virtual Function Manager），该模块负责对 VF 进行实时管理，并根据客户机的优先级动态地进行分配。

图中客户机 1 和客户机 2 具有 VF 的访问权，而客户机 3 不具备访问 VF 的权限。如果客户机 3 需要进行 I/O 操作，客户机 3 的 VF 驱动会向 VFM 发起 I/O 申请，VFM 将记录此次申请，并且不断提高客户机 3 的优先级，直至客户机 3 的优先级超过客户机 2（假设客户机 1 的优先级大于客户机 2 的优先级）。VFM 剥夺客户机 2 对 VF 的访问权，使客户机 2 处于等待状态。在此之后，VFM 将该 VF 分配给客户机 3，并将其 I/O 请求转发给该 VF，客户机 3 通过 Passthrough 方式直接对该 VF 进行 I/O 操作。

扩展的 VF 共享模型根据第二章介绍的 Linux 和 Xen 中 SR-IOV 规范的实现原理，对其实现流程做了相应的扩展，增加了 VFM 结构、对客户机处理 I/O 请求的获取以及动态调整 VF 的算法。该模型的基本工作流程如下：

（1）当客户机中的 VF 设备驱动程序发起一次 I/O 操作时，由于 I/O 缺页错误会陷入到 VMM 中，由 VMM 将此请求转发给 VFM；

（2）VFM 接收客户机 VF 驱动发出的 I/O 请求并记录此次请求，在与客户机对应的数据结构中设置最近访问时间、处理量等基本信息，重新计算所有客户机的优先级；

（3）VFM 根据客户机 VF 的相关信息，对使用 VF 的客户机队列按照优先级进行排序；

（4）VFM 使用动态 VF 分配算法，对客户机使用 VF 的映射关系进行动态调整。如果该客户机正在使用某个 VF，则不做调整；否则，剥夺使用 VF 优先级最低的客户机，将 VF 分配给该客户机；

（5）VFM 将客户机 I/O 操作请求转发给对应的 VF，客户机对其 VF 进行 I/O 操作。

（6）VMM 定时清空其 IOTLB 中所对应的条目，VFM 同时也更改 VF 的状态，以便客户机下次进行 I/O 操作请求时可以跳转至步骤 1。

3.3 基本工作流程

根据本文提出的 SR-IOV 扩展 VF 共享模型原理，作者修改了 SR-IOV 的基本工作流程，主要包括硬件初始化、宿主机对 VF 的设置以及 VF 的分配与剥离三个方面，具体实现基于 Linux 2.6.34 内核和 Xen 3.1.0 平台。

3.3.1 SR-IOV 硬件的初始化

SR-IOV 硬件设备及其相关结构是 BIOS 通过 ACPI 表传送给上层系统软件，

在 BIOS 解析 ACPI 表时探测 SR-IOV 设备，并得到探测的结构信息。利用这些信息，VMM 初始化 DMA 重映射硬件单元定义（DRHD）结构，若 DMA 重映射硬件单元定义结构包含设备域结构（DSS），则还需解析设备域结构从而得到其中的所有设备。

设备域结构字段指向 DMA 重映射硬件单元管理的设备域结构首地址。进行解析前首先要扫描 PCI 总线，计算出设备域结构组内的设备数量。设备域中的设备可以是 PCI 设备，也可用是 PCI 桥（PCI Bridge）设备。如果是 PCI 设备，设备域结构中的计数值递增；如果是 PCI 桥设备，则要遍历该 PCI 桥下的所有设备。取得设备域结构组中的外设数量后，使用设备域结构指针从数组头部开始遍历所有设备，并把遍历到的设备增添到各自设备的链表中。完成终端设备探测后，DMA 重映射硬件单元数据结构中的设备链表保存了 DMA 重映射硬件单元所管理的设备。

在探测 SR-IOV 设备之后，需要创建 SR-IOV 设备的实例，SR-IOV 在 Linux 内核中的数据结构定义为：

```
/* SR-IOV数据结构 */
struct pci_sriov {
        int pos;                /* 功能的位置 */
        int nres;               /* 资源数 */
        u32 cap;                /* SR-IOV功能寄存器 */
        u16 ctrl;               /* SR-IOV控制 */
        u16 total;              /* VF的总数 */
        u16 initial;            /* 初始化VF的数量 */
        u16 nr_virtfn;          /* 可用VF的数量 */
        u16 offset;             /* 第一个VF的RID偏移 */
        u16 stride;             /* VF步长 */
        u32 pgsz;               /* BAR的页大小 */
        u8 link;                /* 功能依赖链接 */
        struct pci_dev *dev;    /* 最低编号的PF指针 */
        struct pci_dev *self;   /* PF指针 */
        struct mutex lock;      /* VF总线锁 */
        struct work_struct mtask; /* VF迁移任务 */
        u8 __iomem *mstate;     /* VF迁移状态表 */
};
```

SR-IOV 的初始化函数为 pci_iov_init，具体流程如图 3.3 所示。初始化 SR-IOV 设备需要判断该设备是否为 SR-IOV 设备。如果具备 SR-IOV 功能，则需要进一步判断该设备是否包含 VF。在确认之后，VMM 会设置该 SR-IOV 设备的 ARI 标志和 VF 的数目。通过读取设备寄存器上的 VF 偏移量和步长，计算 BAR 页面的大小和资源数。根据计算结果，创建 SR-IOV 设备，设置初始值。对 SR-IOV 数据结构中的字段的赋值，主要包括功能的位置、PF 指针、VF 总线锁等。最后，将该设备设置为 SR-IOV 设备，并设置 PF 位。

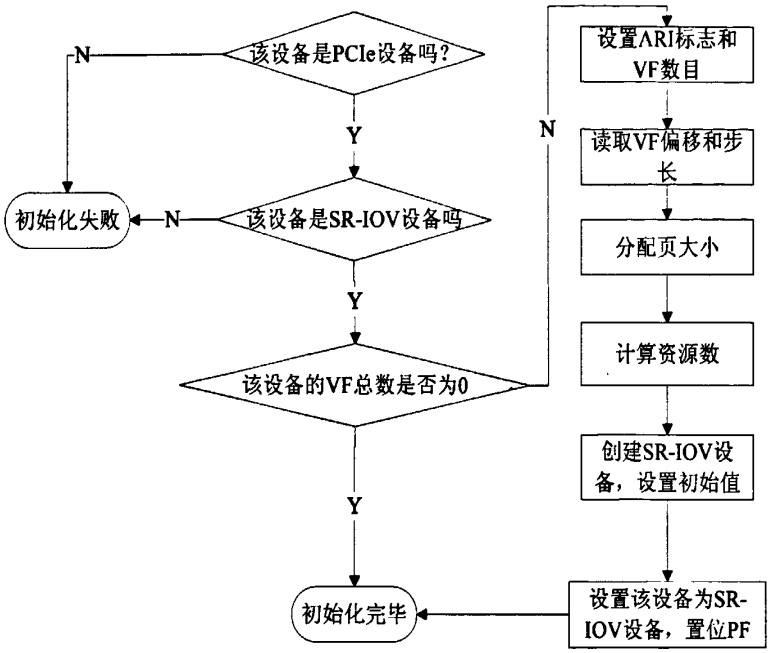

图 3.3 SR-IOV 设备初始化

3.3.2 宿主机对 VF 的设置

由于 Passthrough 模型使得某个客户机独享某一物理设备，因此需要在客户机使用设备之前阻止来自其它客户机对该特定物理设备的访问请求。虚拟机的解决方案中是将设备隐藏，使得其他客户机看不到该设备。具体在 Xen 中实现，通过宿主机来隐藏某一特定设备。Xen 使用一个称为 PCIBack 的模块，PCIBack 具有最高优先级。该模块支持一个特殊的内核参数，称为 PCIBack.hide，其格式为 PCIBack.hide = “Bus: Dev:Func”。其中 Bus、Dev、Function 分别代表了要隐藏设备的总线号、设备号、功能号（简记为 BDF）。通过在客户机的操作系统引导文件中指定该内核参数，PCIBack 模块会在加载时接管 BDF 指定的设备。由于 Linux 只为设备加载优先级最高的驱动，而 PCIBack 正具有最高优先级，所以 PCIBack 接管的设备就不会再加载其它内核驱动。这样，对于宿主机来说，没有该设备的驱动加载，就无法使用，从而达到隐藏设备的目的。

因此，该模块主要是加载 SR-IOV 的 PF，对 VF 进行隐藏，而其他客户机就可以通过 Passthrough 直接访问该设备的 VF。

对于 SR-IOV 设备，其 VF 在 PF 驱动加载之后出现。一旦卸载 PF 驱动，所有与 PF 相关的 VF 也将消失。在 VT-d 的使用环境中，VF 可以看成一般的 PCI 设备。我们需要隐藏 VF，即在客户端中配置文件中使用的 VF 的 BDF，然后启动 HVM 客户端。同时也需要 VF 的驱动，该驱动是一般的 PCI 设备驱动。这就意味着也需

要启动 Xen/MSI 支持。由于 VF 是由 PF 驱动动态分配的，这需要使用动态的隐藏方法。Linux 内核对 SR-IOV 的操作提供三个主要的 API，分别是：启动 SR-IOV 功能的 pci_enable_sriov，停止 SR-IOV 功能的 pci_disable_sriov，提供 SR-IOV 设备 VF 迁移功能的 pci_sriov_migration。

启动 SR-IOV 功能的函数形式为：int pci_enable_sriov(struct pci_dev *dev, int nr_virtfn)，其中 dev 表示 PCI 设备，nr_virtfn 表示虚拟化 VF 的数目。该函数返回 0 表示启动成功，负数则表示启动失败。具体启动流程如图 3.4 所示。首先判断是否是物理功能，并从设备寄存器中读取 VF 数目信息，判断 nr_virtfn 数目是否合法；然后，读取偏移量和步长信息，并根据 BAR 计算设备资源，判断是否与硬件资源数匹配；然后判断 SR-IOV 链接与设备功能是否匹配？若不匹配，获取 PCI 插槽，释放该设备，创建链接；否则，设置 SR-IOV 的 VF 标志和存储空间标志，设置 SR-IOV 的初始 VF 数目，增加 nr_virtfn 个 VF；最后查看 SR-IOV 是否设置了迁移，如果 SR-IOV 设置了迁移，则启动迁移功能。

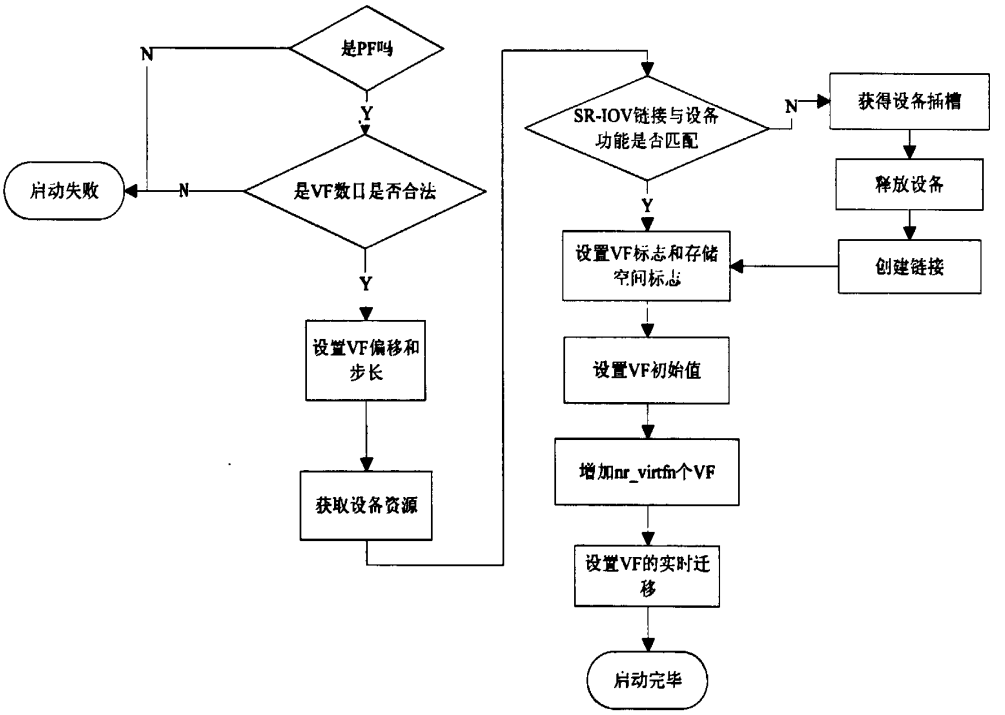

图 3.4 启动 SR-IOV 功能

停止 SR-IOV 功能的函数形式为：void pci_disable_sriov(struct pci_dev *dev)，其中 dev 表示 PCI 设备。具体启动流程如图 3.5 所示。首先判断是否是物理功能，判断 VF 数目为 0，若是则退出；如果 SR-IOV 设置了迁移功能，则停止迁移功能；去除所有 VF，去掉 SR-IOV 的 VF 标志和存储空间的标志，等待 1 秒后退出。

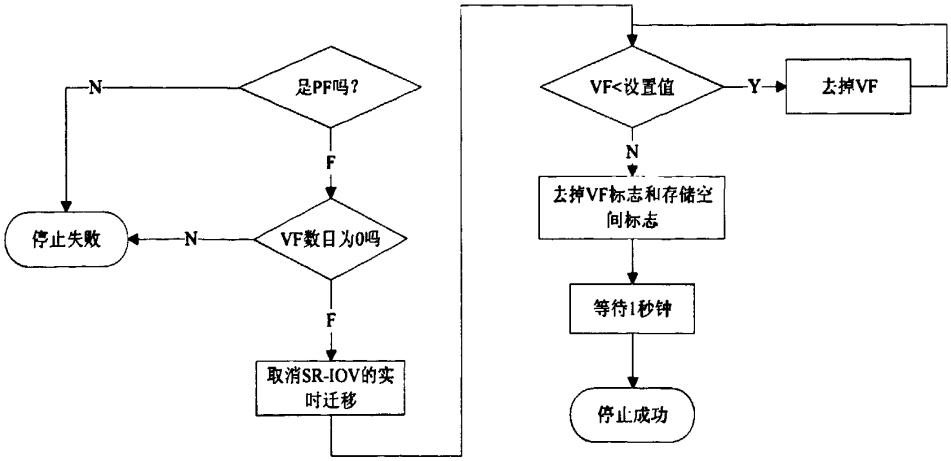

图 3.5 停止 SR-IOV 功能

迁移 SR-IOV 中 VF 功能的函数形式为：irqreturn_t pci_sriov_migration(struct pci_dev *dev)，其中 dev 表示 PCI 设备。该函数返回 IRQ_HANDLED 表示中断已处理，否则返回 IRQ_NONE。具体启动流程如图 3.6 所示。首先判断是否是物理功能，判断 VF 数目是否为 0，若是则返回 IRQ_NONE；如果没有设置 VF 迁移或者 VF 迁移状态置位，则返回 IRQ_NONE；迁移任务进行调度每个 VF，其中需要读取迁移状态，如果为迁入状态，则写入激活状态，增加并重置虚拟设备；如果为迁出状态，则写入非激活状态，删除并重置虚拟设备；最后，取消 VF 迁移状态。

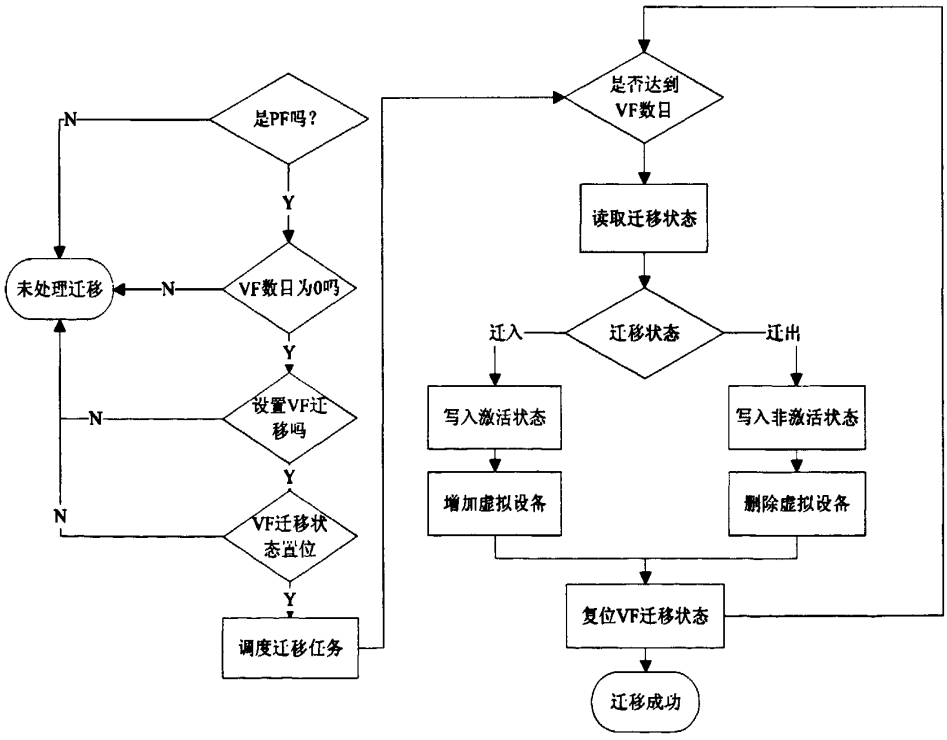

图 3.6 SR-IOV 迁移中断处理

PF 可以通过封装在功能中的寄存器的形式动态地对 VF 进行分配。在默认情况下，该功能没有启用，PF 的行为与传统的 PCIe 设备一样。一旦启动该功能，每个 VF 的 PCI 配置空间可以通过其自身的 BDF 进行访问。每个 VF 同样也包括 PCI 存储空间，用于映射寄存器。VF 设备驱动可以对寄存器进行操作，因而其功能看起来与真实的 PCI 设备一样。

设备驱动（PF 驱动）可以通过使用内核对 SR-IOV 操作提供的的 API 来控制功能的启动和停用。如果硬件具有 SR-IOV 的功能，加载 PF 驱动可以启动该设备和所有的 VF。VF 可以视为是在内核中具有热插拔能力的 PCI 设备。因此，它们可以像真正的 PCI 设备一样工作，VF 的设备驱动与一般的 PCI 设备一样。Xen 支持 PCI 设备的热插拔，并提供 do_pci_add 和 do_pci_del 表示 PCI 设备的热插入和热删除。

3.3.3 VF 的分配与剥离

Xen 虚拟机环境可以动态地启动和关闭客户机，同样设备分配也要求是动态的，所以客户机具有动态的分配设备和动态剥离设备功能。在设备分配部分，将从宿主机剥离某个设备并重新分配给某个客户机。向客户机分配/剥离设备，是向客户机提供动态获得真实设备直接访问能力的关键。它使得一个客户机可以在创建初期获得直接分配的设备，在销毁时将设备归还给宿主机。由于在启动阶段把所有的设备分配给了宿主机，为客户机 N 分配设备就是从宿主机剥离设备，向客户机 N 分配设备的过程；在客户机 N 销毁时，又是一个从客户机 N 剥离设备，向宿主机分配设备的逆过程。

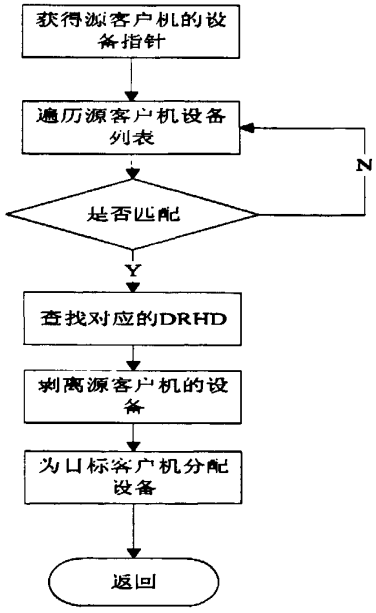

图 3.7 VF 分配/剥离过程

Xen 提供了 reassign_device_ownership 函数作为设备分配/剥离操作，其基本流程如图 3.7 所示。剥离源客户机的设备调用了函数 domain_context_unmap，该函数通过对 PCI 设备类型的判断，并释放 PCI 设备和 PCI 桥设备下的所有设备；为目标客户机分配设备调用了函数 domain_context_mapping，该函数与剥离源客户机的过程相反。

3.4 VF 动态分配算法

3.4.1 基本数据结构和算法

我们在 IOVM 中增加了 VFM，VFM 需要记录每个客户机中 VF 对应的基本状态，包括优先级、最近访问时间、I/O 处理量、是否处于空闲状态、对应的客户机等基本信息。因此，我们定义了具有优先级的 VF 设备，其数据结构的主要字段为：

```
/* 具有优先级的 VF 设备 */
struct vf_dev {
    time_t lasttime;        /* 最近访问时间 */
    int proprity;           /* 优先级 */
    long size;              /* 处理 IO 的字节数 */
    int dom_index;          /* 对应的客户机 */
    int vf_index;           /* 对应 SR-IOV 的 VF */
    unsigned int is_idle; /* 该客户机的 VF 是否处于空闲状态 */
    struct pci_sriov *sriov;/* SR-IOV 功能指针 */
}
```

计算 vf_dev 优先级是对该数据结构操作的重要算法，该算法借鉴了 LRU 算法，采用等待时间优先的策略，其代码如下：

```
//计算vfdev的优先级
static void vf_proprity(struct vf_dev *vfdev)
{
    long waittime;  //等待时间

    waittime = now - vfdev->lasttime;
    if (vfdev->vf_index == -1){  //该客户机尚未分配VF
        //等待时间优先
        vfdev->proprity = waittime * vfdev->size;
    }
    else{
```

```
        //客户机空闲，优先级为处理I/O的字长
        if (vfdev->is_idle)
            vfdev->proprity = vfdev->size;
        else
            vfdev->proprity += vfdev->size;
    }
}
```

该优先级算法可以保证处于等待状态的客户机随着等待时间的增加，其优先级不断增加。优先级高的客户机使用 VF 的机率大于优先级低的客户机，这可以确保客户机使用 VF 的公平性。考虑一种特殊情况，即某客户机在初始进行 I/O 操作后，不再对进行 I/O 操作。由于 VMM 和 VFM 能够定时对 IOTLB 和 vf_dev 重新进行设置，在此后其客户机 VF 的优先级为 0（size 为 0），因而该客户机所使用的 VF 会被剥夺，从而体现出算法的公平性。

VFM 负责对虚拟机上的所有客户机进行优先级计算，维护 vf_dev 列表，调整 VF 的动态分配，其数据结构的主要字段为：

```
/* VF 管理器 */
struct vfm {
    struct vf_dev *vfdevs;    /* VF 有序链表的头元素 */
    int nr_virtfn;            /* VF 的可用数 */
    int nr_dom;               /* 使用 VF 的客户机数 */
}
```

vfdevs 链表的排序主要针对该数据结构进行的操作，其排序算法采用基本的有序链表方法，根据 vf_dev 中的优先级对 vfdevs 链表进行降序排列。vfdevs 排序主要用于在 VF 分配过程中快速查找客户机。

3.4.2 VF 动态分配算法

VF 动态分配算法可以剥夺优先级低（较长时间没有使用 VF）的客户机对 VF 的使用权限，并将该 VF 分配给优先级较高的客户机（等待较长时间）。

动态调整 VF 分配算法的基本流程如图 3.8 所示。当 VFM 接收到客户机 VF 的 I/O 操作请求后，首先将其对应的 vf_dev 的字段进行设置（is_idle 字段为 0），然后计算 vfdevs 链表中每个 vf_dev 的优先级，并对 vfdevs 链表进行排序。此后，VFM 搜索 vfdevs 链表是否有优先级高但没有分配 VF 的客户机。如果存在，进一步查找优先级低（很长时间没有使用 VF）且处于空闲状态的客户机，最后剥夺该客户机的 VF，并把该 VF 分配给优先级高的客户机。

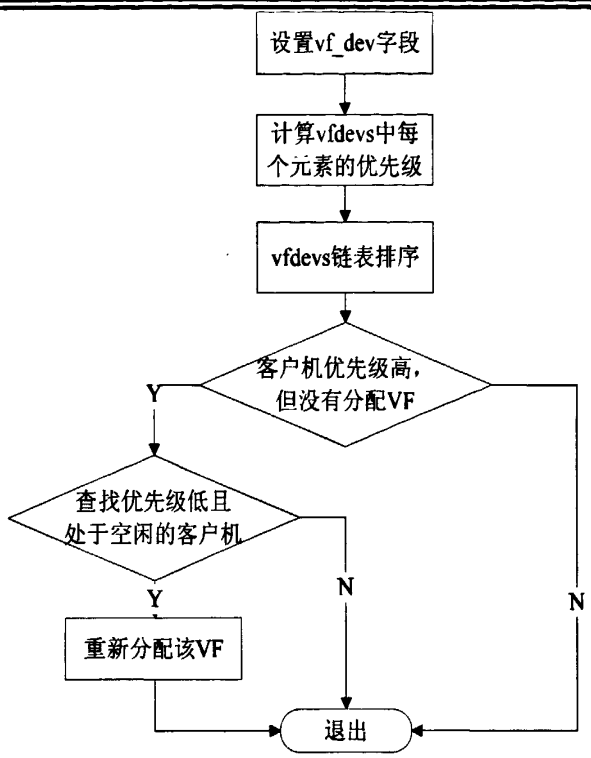

图 3.8 动态调整 VF 分配算法的基本流程

在算法中，VFM 对 VF 的重新分配使用了函数 reassign_device_ownership。此外，在 VMM 中需要增加监控对 VF 进行操作的进程，该进程负责将客户机 VF 驱动发出的 I/O 请求转发给 VFM，并且定时清空 VF 的 IOTLB 以及重新设置 VFM 中的 vf_dev。

VF 动态分配算法具有如下一些优势

（1）高利用率：尽管在 VMM 增加了对 VFM 的排序和分配，但是开销并不大。这使得具有 I/O 请求的客户机可以充分利用 SR-IOV 设备的有限资源，提高利用率；

（2）共享性：虚拟机上的所有客户机都可以使用 SR-IOV 设备，而且在使用的过程中对 VF 是独占的；

（3）公平性：客户机能够公平地竞争和使用 VF 资源，不会造成客户机的死锁和活锁。

3.5 小结

本章针对目前 SR-IOV 技术中面临的 VF 数目有限这一问题，提出了一种扩展 VF 共享模型，用以解决 VF 共享的问题。论文阐述了扩展 VF 共享模型的实现原理、关键方法和代码框架。

当虚拟机数量超过实际可分配的 VF 数量时，通过扩展 VF 共享模型，可以保

证多个虚拟机对物理设备的共享使用；通过引入优先级判断机制和 VF 动态分配算法，可以保证多个客户机之间公平地竞争和使用物理 I/O 资源。

第四章　虚拟 PF/VF 通信模型

本章提出一种虚拟的 SR-IOV 的通信模型，该模型在 VMM 中模拟 SR-IOV 设备的 PF/VF 通信以及 I/O 处理流程，能够避免 SR-IOV 技术对特殊硬件平台的依赖，进一步支持使用 SR-IOV 设备虚拟机在不同硬件平台之间的迁移。

4.1　问题分析

现有的 SR-IOV 技术依赖于特殊的硬件设备，这类硬件设备支持多个虚拟机之间的共享，其核心功能主要是在硬件级实现了 VF 驱动与 PF 驱动之间的通信。例如，对于 SR-IOV 网卡，其 VF 驱动需要将设置多播地址和 VLAN 的客户机请求发送给 PF 驱动。PF 驱动也需要将一些设备状态变化事件转发给每个 VF 驱动，包括设备重置、链接状态改变、驱动移除等。PF/VF 通信的必要组成部分是通信通道，其作用是传送消息，并产生中断通知 PF 或 VF。

为了在软件层次上模拟 SR-IOV 硬件设备，最重要的就是以软件的形式实现 VF 与 PF 之间的通信。SR-IOV 规范能够使得虚拟化 I/O 具有高性能和可扩展的能力，但目前 SR-IOV 规范没有规定 PF 与 VF 通信的实现方式。

PF 与 VF 的通信可以通过三种途径进行解决，其中包括：客户机之间的通信、通过 VMM 进行通信和硬件支持的通信。这三种方式的实现方式不同，各有其优势和不足，具体表现为：

（1）客户机之间的 PF/VF 通信采用软件模拟的泛虚拟化方式，可以在一定程度上提高 PF/VF 的通信效率，但是需要根据 VMM、客户机操作系统修改 VF 驱动，需要为客户机提供额外的 Hypercall 调用，而且很多商用操作系统（例如，Windows）不支持客户机之间通信的 API，如果只在 Xen 上发布相应的 API，无法取得较好的应用。

（2）VMM 中支持 PF/VF 通信，这种方式不必修改 VF 驱动，但需要按照 SR-IOV 规范对 PCIM（PCI 管理器）进行扩展，这种方式可以很好地进行实现 PF/VF 通信，其问题是在 VMM 中的修改较多，需要增加消息传送通道和虚拟中断，还要为端口 IO 和 MMIO 重映射提供支持，而且执行通信的开销相对较大。

（3）硬件支持的 PF/VF 通信，这种方式不必修改 VF 驱动，对 VMM 的影响最小，具有几乎本机的性能优势，是最佳的解决方法，但问题是并不是所有的硬件提供商对 PF/VF 的通信支持。

由于 SR-IOV 规范没有规定 PF/VF 通信的实现方式，而硬件提供商生产的设备有所不同，已生产的设备又不会立即退出市场，这给目前基于 SR-IOV 设备的虚

拟化技术带来了一个问题：一个客户机使用了在硬件中实现 PF/VF 通信的 VF，如何使该客户机迁移到在硬件中没有实现 PF/VF 通信的 SR-IOV 设备上？这又引入了 SR-IOV 技术在两个方面的不足：

- 客户机无法有效地进行跨硬件平台迁移。由于硬件平台的不同，而使用 SR-IOV 的设备通常采用 Passthrough 方式对 VF 进行 I/O 操作。因此，客户机在不同的 SR-IOV 设备上进行迁移的时候，会导致设备操作失败。
- SR-IOV 设备的应用受到限制。由于现有的 VMM 尚未提供 PF/VF 的通信机制，难以支持所有 SR-IOV 设备，不能充分发挥 SR-IOV 设备高性能和共享的优势。

4.2 模型和原理

为了解决上述问题，我们采用了 VMM 支持 PF/VF 通信的方式。SR-IOV 设备在 Xen 中的虚拟 PF/VF 通信方式如图 4.1 所示。由于 Xen 大体上是由用户端的控制面板（Control Panel）和设备模型（Device Model）以及下层的 Hypervisor（VMM）组成，因此该模型需要对这三部分进行扩展。

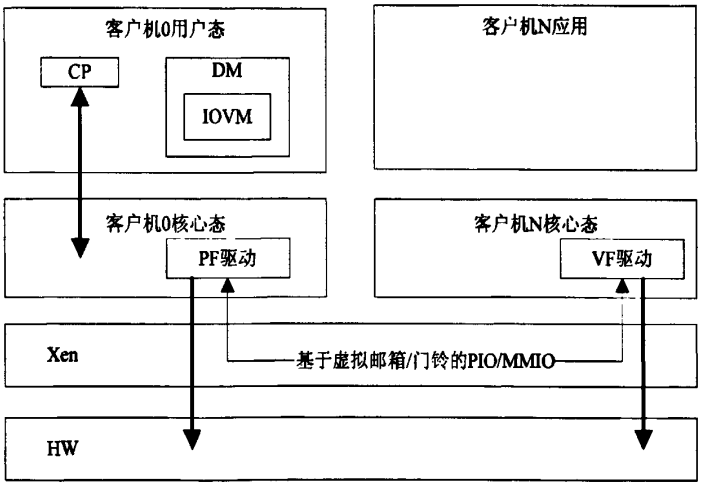

图 4.1 虚拟 PF/VF 通信方式

控制面板主要提供了用户接口和客户机的管理功能，需用增加对 VF 管理和配置。设备模型向虚拟机上的设备提供软件支持，这些软件模块本质上是通过模拟所有的硬件级编程接口来实现的，而设备驱动程序则会使用这些接口来进行 I/O 操作。设备模型中需要增加 SR-IOV 设备的模拟。VMM 基于 PIO 或 MMIO 重映射机制需要提供虚拟的 PF/VF 通信方式。

我们在 VMM 中采用了虚拟的邮箱/门铃机制实现 PF 和 VF 驱动之间的通信。即发送方将消息写入邮箱，然后按响“门铃”，这将会产生一个中断并通知接收方消息已经发送。接收方接收信息后，将事件中的 pending 位进行设置，表明信息

已接收。该模型的涉及 VF 向 PF 发送消息，PF 向 VF 发送消息两个方面：

（1）VF 向 PF 发送消息

SR-IOV 设备的 VF 属于“轻量级”PCIe 功能，具有重置、发送和接收数据的功能，除此之外的功能必须通过对 PF 的设置完成。VF 向 PF 进行通信的流程为：VF 驱动发送请求，进入到 VMM 中，VMM 将该请求记录到邮箱中，通过门铃通知给宿主机的 DM，DM 接收请求，对 PF 进行设置，得到结果返回给 DM，DM 将结果写入邮箱中，并产生门铃通知客户机 N。

（2）PF 向 VF 发送消息

由于 PF 对 SR-IOV 设备的操作通常具有全局效应，因此 PF 需要向所有 VF 发出通知，其流程为：PF 将设备结果返回给 DM，DM 将结果分别写入与 VF 对应的邮箱中，并产生门铃通知所有客户机的 VF。

4.3 基本实现机制

4.3.1 虚拟邮箱/门铃的实现

虚拟邮箱/门铃机制的实现涉及两个内容：邮箱的实现和门铃的实现。我们利用 Xen 中已有的机制提供解决方法。

虚拟邮箱的实现采用共享 I/O 页的方式，共享 I/O 页是一个队列，客户机操作系统与 Xen 虚拟机管理器之间数据异步传输的数据结构。共享 I/O 页由客户机分配，Xen 也可以访问，因此是一种共享资源。共享 I/O 页包括数据传输的缓冲区以及描述符。对于每个共享 I/O 页的访问是基于两组“生产者-消费者”指针。客户机把数据及请求描述符放入 I/O 页上，同时更新请求生产者指针；当 VMM 取出数据和请求后更新关联的请求消费者指针。同样，当 VMM 准备好响应数据之后，也把数据及相应描述符放到 I/O 页上并更新响应生产者指针；客户机可以取出响应数据交给应用程序，并更新响应消费者指针。请求或响应进入共享 I/O 页的顺序没有特殊规定。当客户机发出请求时会分配一个唯一的标识符，VMM 的响应中也带有该标识符。通过标识符，VMM 可以根据任务调度和 I/O 优先级进行重新排序 I/O 操作。请求和响应之间没有紧耦合。客户机可以在产生合理数目的请求后再通知 VMM，VMM 也可以在响应一定数目的请求后再通知客户机。

虚拟门铃的实现利用 Xen 的事件通道方式，事件通道是通过客户机与 VMM 之间的协商创建，通过彼此间的合作来完成的。事件可以是同步的，也可以是异步的。同步事件是由客户机操作所生成并进入 VMM 中，如运行 I/O 指令，访问 MMIO 等，这些都是通过事件通道发送到设备模块以进行进一步的处理。异步事件则包括由设备模块产生的外部中断。这些异步事件将被作为一个外部中断插入

到客户域中。事件通道发送消息会在每一个虚拟域的内部字段（pending）进行置位，以便客户机处理。

虚拟邮箱/门铃机制产生了虚拟中断，需要根据端口 I/O（PIO）或内存映射 I/O（MMIO）中的 BAR 进行绑定。在完成 VF 直接分配的主要功能后，Passthrough 设备把 BAR 映射的信息传递给 VMM，由 VMM 通知客户机建立相应的端口 I/O 或内存映射 I/O 转换表，创建虚拟邮箱/门铃（如图 4.2 所示）。抽象 SR-IOV 的 VF 设备模块和客户机 N 的 VF 驱动初始化将在下面分别说明。

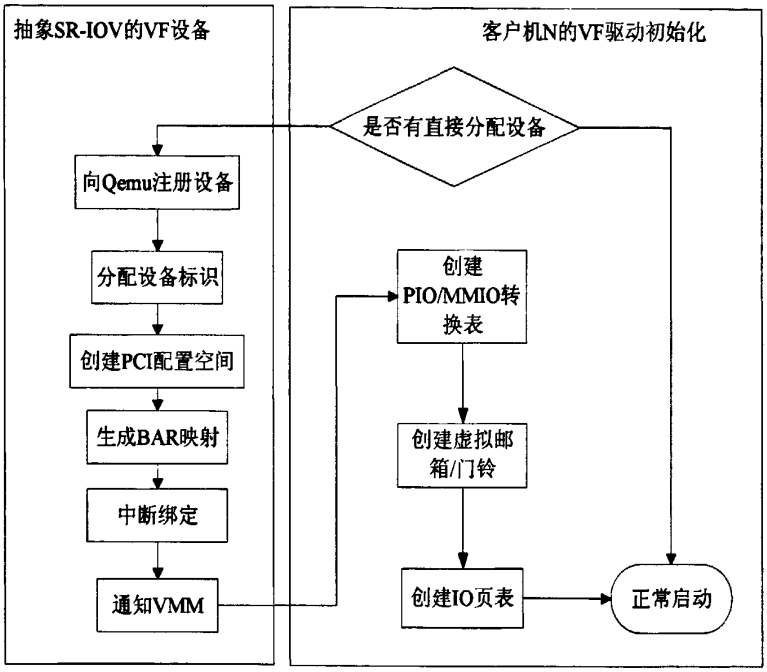

图 4.2 基于虚拟邮箱/门铃的 SR-IOV 实现

4.3.2 抽象 SR-IOV 的 VF 设备模块

客户机采用Passthrough方式访问VF，这需要将VF设备注册到QEMU[38]。在客户机直接分配物理设备的时候，使用设备的BDF号作为参数传递给QEMU，以获取设备的PCI配置空间信息。对于PCI设备，一旦设置了PCI配置空间信息，就对外部描绘出了整个PCI设备。因此，抽象VF的工作重点是配置设备的PCI配置空间。

Passthrough 模型的初始化函数为 pt_init。该函数首先分配 pci_access 结构，这个数据结构描述了平台上所有 PCI 硬件设备的信息。pci_access 结构中的两个重要成员为 device 和 method。指针 device 指向一个 PCI 设备链表，该链表保存着系统中所有的 PCI 设备。指针 method 指向一个函数指针，该结构体封装了对 PCI 设备进行操作的常用函数。之后，通过 pci_bus_scan 函数扫描 PCI 总线，枚举系统中所有的 PCI 设备，并填入 device 指向的链表中。

通过 pt_init，已经获得了系统中所有的 PCI 设备的信息，存放在 device 链表中。用户在配置文件中指定的分配的设备，必须包含在该链表中。在向 QEMU 注册前，要对该设备进行验证。在进行验证时，使用配置文件中的 BDF 信息和 device 链表中 PCI 设备的 BDF 匹配，如果找到所分配的设备，则创建 PCI 配置空间，生成 PCI BAR 映射，进行中断绑定。图 4.3 展示了采用 Passthrough 方式的 VF 注册流程，其函数为 register_real_device。

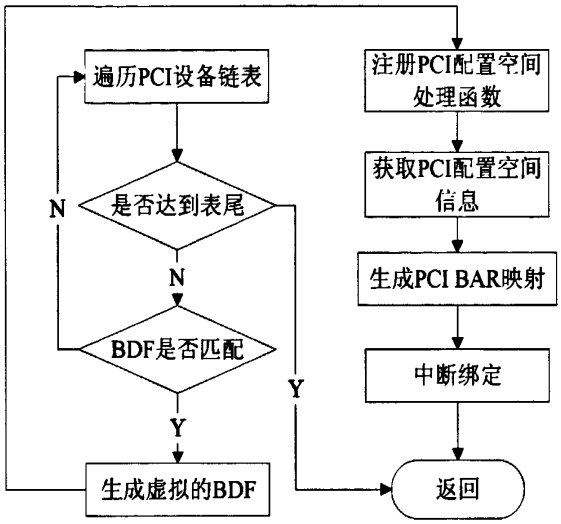

图 4.3 采用 Passthrough 方式的 VF 注册

在设备注册 QEMU 的过程中，对于 SR-IOV 的 VF 来说，涉及到两个处理函数：pt_pci_read_config 和 pt_pci_write_config，当客户操作系统对设备 PCI 配置空间进行修改时，这两个函数会作为回调函数（Callback）被调用。

生成 PCI Bar 映射是对所分配设备 PCI 配置空间中的 PCI BAR 使用 QEMU 分配的虚拟信息。在 PCI 设备中，PCI BAR 用于指出设备端口 I/O 寄存器、MMIO 寄存器，以及扩展存储器的在处理器物理地址空间中的基地址。驱动程序通过这些基地址，读写设备的寄存器与硬件进行交互。PCI BAR 映射过程负责把 QEMU 产生的 PCI BAR 与设备真实 PCI BAR 间的映射关系通知 VMM，以便于建立端口 I/O 映射表或 MMIO 映射表。

在抽象 VF 设备就向 QEMU 注册成功之后，当客户机启动时会收到 QEMU 程序报告的 Passthrough 的 VF 设备信息，虚拟客户机操作系统在扫描到该 VF 设备时，会根据其 PCI 配置空间信息识别出设备类型，并加载相应驱动（如图 4.4 所示）。

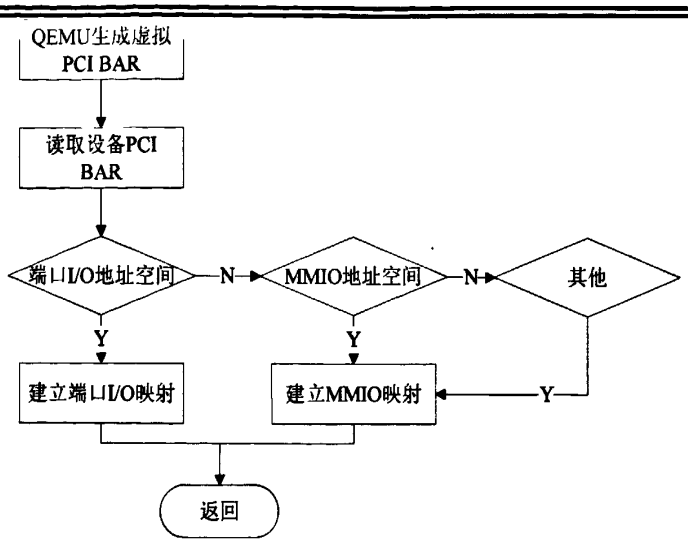

图 4.4 生成 PCI BAR 映射

4.3.3 客户机 N 的 VF 驱动初始化

客户机 N 的 VF 驱动初始化需要为客户机 N 创建端口 I/O 或 MMIO 转换表和 I/O 页表，

创建端口 I/O 转换表，其实是在客户机物理地址到机器物理地址转换表中设置地址映射的一个过程。Passthrough 设备用于建立端口 I/O 映射的 Hypercall 为 xc_domain_ioport_mapping。创建 MMIO 转换表同样是设置客户机物理地址到机器物理地址转换表的过程，其过程与端口 I/O 类似。对于 MMIO，Passthrough 设备用于创建 MMIO 映射调用的 Hypercall 为 xc_domain_memory_mapping。客户机 N 创建 I/O 页表的过程与宿主机相同。在目前 Xen 的实现中，宿主机从地址空间 0 开始，因此 I/O 页表的映射也从地址 0 开始，覆盖的地址范围为宿主机的地址宽度。

对于端口 I/O，Passthrough 设备用于通知 VMM PCI Bar 映射的 Hypercall 具有以下原型：

```
int xc_domain_ioport_mapping(
int xc_handle,
uint32_t domid,
uint32_t first_gport,
uint32_t first_mport,
uint32_t nr_ports,
uint32t add_mapping)
```

其中 xc_handle 是用户态描述客户机的一个句柄，domid 是客户机号，first_gport 为 QEMU 创建的虚拟端口 I/O PCI Bar 的起始端口，first_mport 是设备真实的 P 端口 I / O PCI Bar 起始端 CI，nr_ports 是端口数量，add_mapping 标识此次操作是创

建映射还是解除映射。在创建 Port FO 映射转换表时，first_gport、first_report、nr__ports 三个参数最重要。

对于 MMIO，Passthrough 设备创建 PCI Bar 映射调用的 hypercall 具有如下原型：

```
int xc_domain_memory_mapping(
int xc_handle,
uint32_t domid,
unsigned long first_gfn,
unsigned long first_mfn,
unsigned long nr_mfns,
uint32 t add_mapping)
```

其中 xc_handle、domid、add mapping 和 xc_domain_ioport_mappin()相应参数意义相同。这里 first_gfn 代表虚拟 MMIO PCI Bar 的基地址对应的页帧号。

4.4 虚拟 PF/VF 通信的基本流程

SR-IOV 设备的 PF/VF 通信建立在虚拟邮箱/门铃机制上，VF 向 PF 发送消息的基本流程如图 4.5 所示。

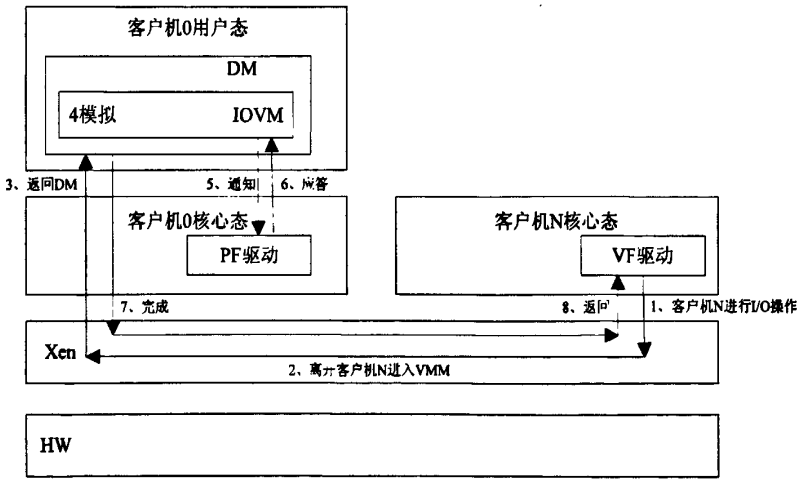

图 4.5 VF 向 PF 发送消息的基本流程

在该模型下，VF 对 PF 的 I/O 请求是由 VF、PF、VMM、宿主机和 DM 配合共同完成的。VF 特定 I/O 请求的处理流程如下：

（1）客户机内核的 VF 驱动执行特定 I/O 指令，触发 VM Exit，处理器调用 VMM 设置的 VM Exit 的处理函数。

（2）VMM 将 I/O 指令的具体信息写入该客户机与 DM 共享的虚拟邮箱（I/O 共享页）中，并通过虚拟门铃（事件通道）通知宿主机，VMM 然后阻塞该客户机。

（3）VMM 恢复宿主机的状态，并把执行 I/O 控制交给 DM。

（4）DM 通过 SR-IOV 设备配置管理 IOVM 对该 I/O 操作进行模拟。

（5）DM 向 PF 驱动发出 I/O 请求，PF 驱动对 SR-IOV 设备进行直接的 I/O 操作。

（6）PF 驱动将操作结果返回给 DM。

（7）DM 的回调函数把虚拟外设的状态写回邮箱中，通过门铃通知客户机处理完毕，并进入到 VMM 中。

（8）VMM 通过 VM Resume 解除客户机的阻塞。未来某一时刻，该客户机可以再次被调度，并从邮箱中获得 I/O 请求的结果。

PF 向 VF 发送消息的基本流程与前者类似，其过程如图 4.6 所示。但由于 PF 的一些 I/O 操作对 SR-IOV 设备具有全局效应，这需要 PF 向所有 VF 发送相应的消息。

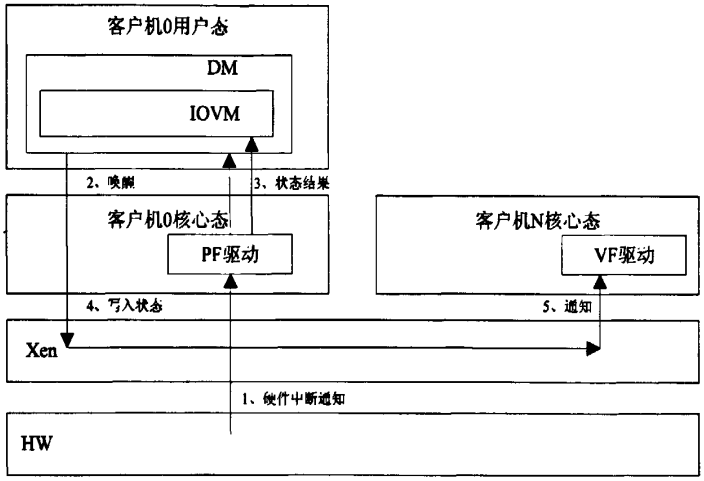

图 4.6 PF 向 VF 发送消息的基本流程

（1）SR-IOV 设备访问完成后，PCI 控制器触发硬件中断，该中断由 VMM 的处理函数接管，接着 VMM 通知宿主机，调用 PF 驱动中的中断处理函数。

（2）宿主机的中断处理函数处理完毕后，唤醒正阻塞于 I/O 等待队列中的 DM。

（3）PF 驱动将操作结果应答返回给 DM。

（4）DM 的回调函数把 SR-IOV 设备的状态写回到邮箱中，将硬件中断通过门铃通知所有客户机处理完毕，并进入到 VMM 中。

（5）VMM 通过 VM Resume 解除客户机 N 的阻塞，该客户机可以再次调度后从其邮箱中 SR-IOV 设备的状态。

4.5 小结

本章针对目前 SR-IOV 技术面临的挑战，提出了虚拟 PF/VF 通信模型，用以

解决 SR-IOV 技术的硬件依赖性和虚拟机跨物理平台迁移的问题，阐述了实现原理、关键方法和实现过程。

通过本章提出的虚拟 PF/VF 模型，可以在 VMM 软件层模拟 PF 与 VF 之间的通信，确保在缺乏 SR-IOV 硬件设备支持的情况下，依然能够充分利用 SR-IOV 机制，使得基于 SR-IOV 技术实现的虚拟机能够顺利迁移到没有 SR-IOV 硬件的物理平台上，增强了 SR-IOV 技术的实用性。

第五章 测试与分析

SR-IOV 实现模型在理论上具有 Split I/O 模型的共享性和 Passthrough I/O 模型的高性能。本章通过对 SR-IOV 实现模型在网络方面进行性能测试，从量化的角度分析了 SR-IOV 实现模型的优势。本章介绍了评测的软硬件实验环境，以及第三方测试工具，针对 SR-IOV 实现模型进行性能和共享性的测试；最后，对测试结果进行了分析。实验结果表明 SR-IOV 实现模型能够有效减少 CPU 开销，提高 I/O 利用率，具有良好的共享性和接近于实际机器的性能。

5.1 测试环境

5.1.1 硬件环境

本地主机与远端主机都是基于 Intel X86 体系结构的 PC 机。其中本地主机是部署 SR-IOV 实现模型 Intel 82576 千兆网卡,并且支持虚拟化的 E8600 处理器。远端主机是进行模型性能测试过程中所需要用到的服务器端。SR-IOV 实现模型进行网络测试的硬件环境如表 5.1 所示。

表 5.1 硬件测试环境

配置	本地主机	远端主机
CPU	Intel(R) Core(TM)2 Duo CPU E8600 3.33GHz 3.32GHz	Intel(R) Pentium(R) Dual CPU E2180 2.00GHz
内存	2048M	1024M
网卡	Intel 82576 千兆网卡	Marvell Yukon 88E8056 千兆网卡

5.1.2 软件环境

宿主域为 RedHat Enterprise 5.4，支持 VT，VT-d，SR-IOV 设备，其内核版本为修改后的 Linux 2.6.34，并提供对修改后的 Xen 的支持。Xen-3.1.0 是当前流行的一个相对稳定的版本。SR-IOV 实现模型进行网络测试的软件环境如表 5.2 所示。在测试的过程中，我们将 SR-IOV 实现模型网卡的带宽平分给三个客户机。客户机的实际 CPU 利用率是通过检测物理 CPU 的利用率的增值计算得到的。

表 5.2 软件测试环境

配置	本地主机	远端主机
Xen	Xen-3.1.0	
客户机	RedHat Enterprise 5.4 +Netperf	
宿主机	RedHat Enterprise 5.4 +Netperf	RedHat Enterprise 5.4 +Netperf

5.1.3 测试软件

Netperf 是一种网络性能的测量工具，主要针对基于 TCP 或 UDP 的传输。Netperf 根据应用的不同，可以进行不同模式的网络性能测试，即批量数据传输（bulk data transfer）模式和请求/应答（request/reponse）模式。Netperf 测试结果所反映的是一个系统能够以多快的速度向另外一个系统发送数据，以及另外一个系统能够以多块的速度接收数据。

Netperf 工具以 client/server 方式工作。server 端是 netserver，用来侦听来自 client 端的连接，client 端是 netperf，用来向 server 发起网络测试。在 client 与 server 之间，首先建立一个控制连接，传递有关测试配置的信息，以及测试的结果；在控制连接建立并传递了测试配置信息以后，client 与 server 之间会再建立一个测试连接，用来来回传递着特殊的流量模式，以测试网络的性能。

由于 TCP 协议能够提供端到端的可靠传输，因此被大量的网络应用程序使用。但是，可靠性的建立是要付出代价的。TCP 协议保证可靠性的措施，如建立并维护连接、控制数据有序的传递等都会消耗一定的网络带宽。

Netperf 可以模拟三种不同的 TCP 流量模式：

1） 单个 TCP 连接，批量（bulk）传输大量数据

2） 单个 TCP 连接，client 请求/server 应答的交易（transaction）方式

3） 多个 TCP 连接，每个连接中一对请求/应答的交易方式

UDP 没有建立连接的负担，但是 UDP 不能保证传输的可靠性，所以使用 UDP 的应用程序需要自行跟踪每个发出的分组，并重发丢失的分组。

Netperf 可以模拟两种 UDP 的流量模式：

1） 从 client 到 server 的单向批量传输

2） 请求/应答的交易方式

由于 UDP 传输的不可靠性，在使用 netperf 时要确保发送的缓冲区大小不大于接收缓冲区大小，否则数据会丢失，netperf 将给出错误的结果。因此，对于接收到分组的统计不一定准确，需要结合发送分组的统计综合得出结论。

5.1.4 测试目标

随着 Internet 网络技术的普遍应用，绝大部分网络采用 TCP/IP 协议，TCP/IP 协议网络采用的是一种“尽力而为”的传输机制，它不能保证分组顺序的正确性、传输的及时性等因素，因此数据业务的网络流量特征、性能特征、可靠性和安全性都是人们非常关心的问题。决定网络系统整体性能好坏的参数很多，但主要的指标有以下几个：

响应时间——数据分组在网络传输中的延迟时间，也就是 ping 命令的 echo request/reply 一次往返所花费的时间。有很多因素会影响到响应时间，如网段的负荷、网络主机的负荷、广播风暴、网络设备工作不正常等。

吞吐量——单位时间内传送通过网络中给定节点的数据量，网络吞吐量可以帮助寻找网络路径中的瓶颈。网络吞吐量非常依赖于当前的网络负载情况。因此，为了得到正确的吞吐量，需要在不同的时间点进行测试。

CPU 利用率——CPU 利用率是指网络传输中 CPU 被使用的时间占总时间的比例，其中总时间是 CPU 被使用的时间和空闲时间的总和。CPU 利用率反应了网卡自主工作及操作系统的支持的程度。

5.2 测试结果

TCP Stream 批量数据传输典型的例子有 FTP 和其他类似的网络应用（即一次传输这个文件）。测试过程中 Netperf 向服务器发送批量的 TCP 数据分组，以确定数据传输过程中的吞吐量。在 Netperf 向远端发送数据时，远端系统使用大小为 87380 字节的 socket 接收缓冲区，本地系统使用大小为 16384 字节的 socket 发送缓冲区，本地系统向远端系统发送的测试分组大小为 16384 个字节，测试经历的时间为 60 秒，测试结果如表 5.3 所示。

表 5.3 TCP stream 测试结果

	吞吐率（M/s）	CPU 利用率
本机	941.41	7.36
虚拟机	941.36	8.98
客户机 1	322.96	7.27
客户机 2	322.23	7.97
客户机 3	296.86	7.85

UDP Stream 用来测试进行 UDP 批量传输时的网络性能。需要特别注意的是，此时测试分组的大小不得大于 socket 的发送与接收缓冲大小，否则 Netperf 会报出错提示：为了避免这样的情况，可以通过命令行参数限定测试分组的大小，或者增加 socket 的发送/接收缓冲大小。测试结果如表 5.4 所示。

表 5.4 UDP Stream 测试结果

	吞吐率（M/s）	CPU 利用率
本机	962.0	8.45
虚拟机	961.5	11.1
客户机 1	324.1	7.77
客户机 2	322.8	7.18
客户机 3	294.3	7.36

一种典型的网络流量类型是应用在 client/server 结构中的请求/应答模式。在每次的交易 (transaction)中，client 向 server 发出小的查询分组，server 接收到请求，经处理后返回大的结果数据。如下图 5.1 所示：

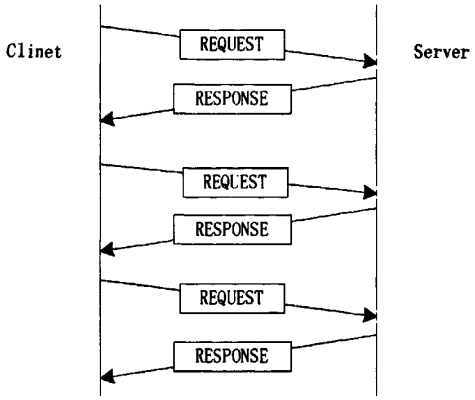

图 5.1 client/server 交易示意图

TCP RR 的测试对象是多次 TCP 请求和应答的交易过程，但是它们发生在同一个 TCP 连接中，这种模式常常出现在数据库应用中。数据库的 client 程序和 server 程序建立一个 TCP 连接以后，就在这个连接中传送数据库的多次交易过程。在本次测试中，远端系统使用大小为 87380 字节的 socket 发送缓冲区和接收缓冲区，本地系统使用大小为 16384 字节的 socket 发送缓冲区和接收缓冲区，本地系统向远端系统发送的请求大小为 1 个字节，远端系统发回的应答大小为 1 个字节，测试经历的时间为 60 秒，测试结果如表 5.5 所示。

表 5.5 TCP RR 测试结果

	交易量	CPU 利用率
本机	6513.71	4.75
虚拟机	6217.90	6.42
客户机 1	1929.20	3.76
客户机 2	1959.25	3.53
客户机 3	1946.27	3.70

UDP RR 方式使用 UDP 分组进行 request/response 的交易过程。由于没有 TCP 连接所带来的负担，所以其交易率一定会有相应的提升。

表 5.6 UDP RR 测试结果

	交易量	CPU 利用率
本机	6831.98	3.41
虚拟机	6689.67	3.65
客户机 1	1965.88	3.78
客户机 2	1964.45	3.43
客户机 3	1965.62	3.03

TCP CRR 为每次交易建立一个新的 TCP 连接，进行一次 TCP 请求和应答的交易过程后，断开此次连接，当有新的交易时，重新连接，最典型的应用就是 HTTP 服务。在本次测试中，远端系统使用大小为 87380 字节的 socket 接收缓冲区和发送缓冲区，本地系统使用大小为 16384 字节的 socket 发送缓冲区和接收缓冲区，本地系统向远端系统发送的请求大小为 1 个字节，远端系统发回的应答大小为 1 个字节，测试经历的时间为 60 秒，测试结果如表 5.7 所示。

表 5.7 TCP CRR 测试结果

	交易量	CPU 利用率
本机	3044.32	3.98
虚拟机	2850.66	4.85
客户机 1	924.13	9.19
客户机 2	913.13	9.50
客户机 3	948.83	9.82

5.3 测试结果分析

本机与宿主机之间的吞吐量、交易量比较分别如图 5.2 和 5.3 所示，从结果中我们可以看出本文实现的 SR-IOV 模型能够使虚拟机充分利用设备资源，几乎达到与本机相同的性能。

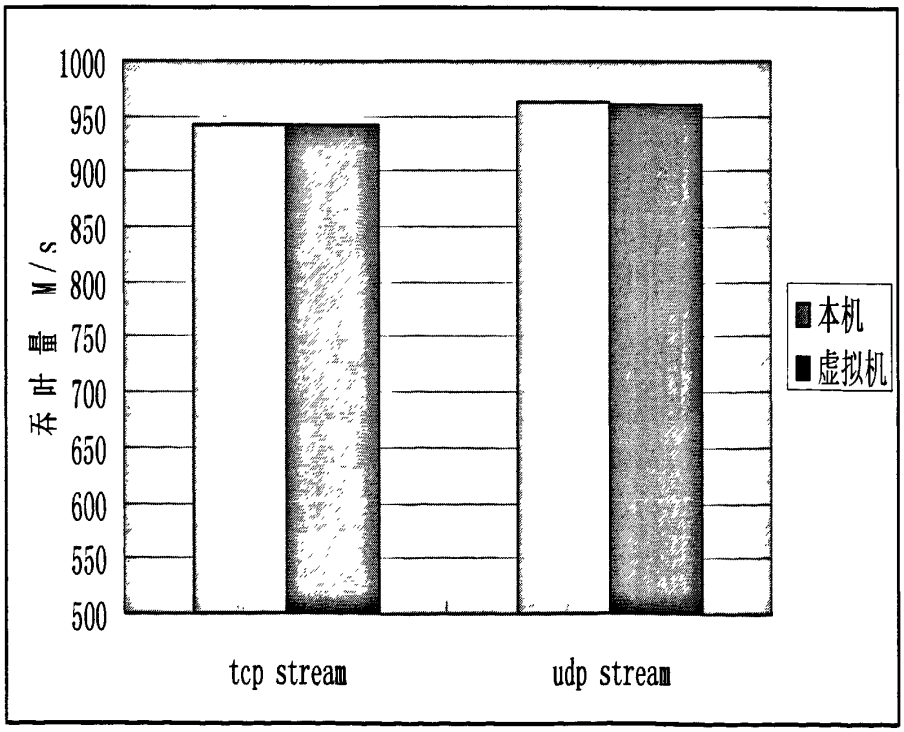

图 5.2 本机与虚拟机的吞吐量比较

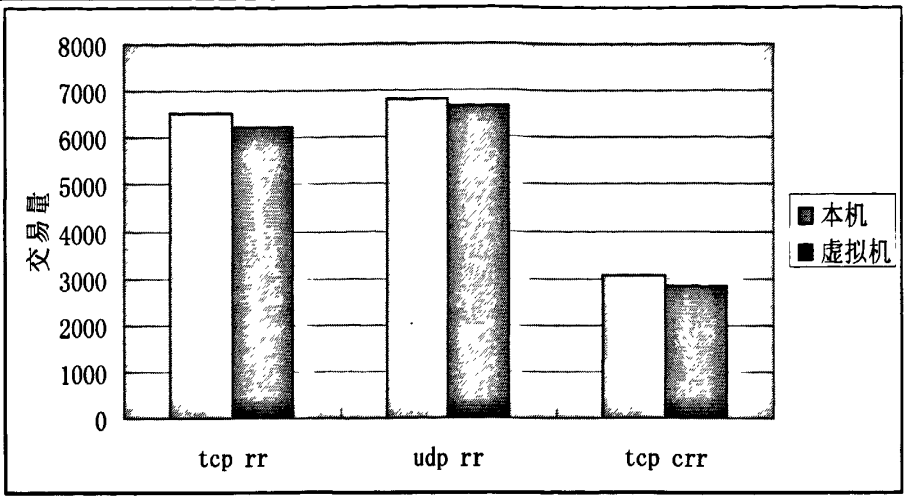

图 5.3 本机与虚拟机的交易量比较

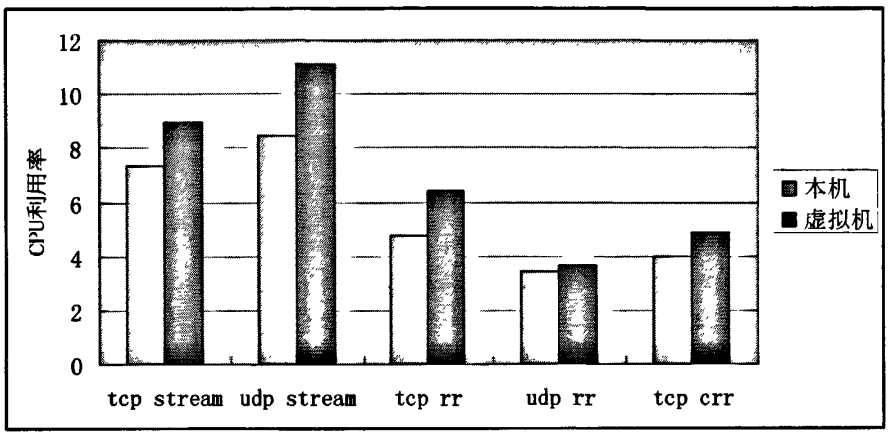

图 5.4 本机与虚拟机的 CPU 利用率比较

本机与宿主机之间的 CPU 利用率比较如图 5.4 所示，通过对的测试结果，我们发现使用 SR-IOV 实现模型的虚拟机采用了 Passthrough 模型直接访问硬件，并没有给 CPU 增加过多的额外负担。

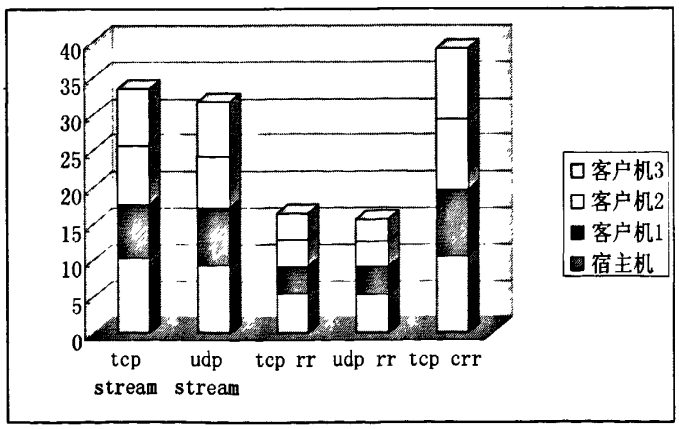

图 5.5 虚拟机 CPU 利用率的增量

在使用 SR-IOV 实现模型的情况下，虚拟机整体的 CPU 利用率增长情况如图

5.5 所示。与 Split I/O 和 Direct I/O 技术相比，SR-IOV 实现模型在设备具有相同吞吐率的情况下，其 CPU 的增加量是很少的。

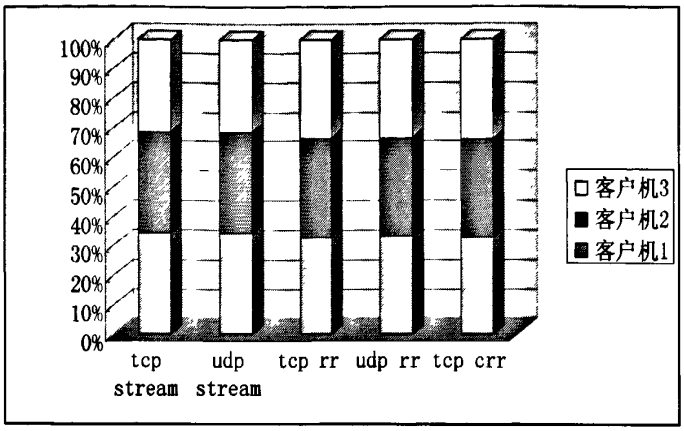

图 5.6 VF 对网卡的使用率比较

虚拟机中 VF 对网卡的使用率如图 5.6 所示，从结果中我们可以看出三个客户机共享同一个 Intel 82576 千兆网卡的资源，并获得了几乎与本机相同的性能。因此，SR-IOV 实现模型具有良好的可扩展性和共享性。

5.4 小结

本章通过网络性能测试工具 NetPerf 对本文实现的 SR-IOV 模型进行评测。在网络性能测试方面，整体性能与本机的性能十分接近，而且通过限制 VF 的数量，采用本文实现的扩展 VF 模型，客户机可以高效共享 SR-IOV 实现模型的物理资源。测试结果分析表明本文实现的 SR-IOV 模型能够可以达到几乎本机的性能，并具有良好的可扩展性。

第六章 总结与展望

前面的章节详细描述了课题研究中所做的工作，本章中将进一步总结，客观评价课题研究中取得的成果，分析存在的不足，并对下一步工作进行展望。

6.1 本文总结

虚拟化技术发展的早期主要致力于CPU和内存的虚拟化，注重于它们性能的提高。然而在I/O虚拟化方面，I/O设备的性能折损却很大。虽然近几年各大IT厂商纷纷提出了多个高效设备访问模型，在一定程度上提高了I/O设备的性能，但是由于它们体系机构的设计原因，它们本身也存在着很大的缺陷。对I/O性能进行优化仍然是目前虚拟环境下迫切需要解决的主要问题之一。SR-IOV技术能够有效解决现有虚拟化I/O的不足，减少CPU开销，提高I/O利用率，具有可扩展、易共享等特点，具有广泛的应用前景。

本文对Intel最新基于I/O虚拟化的研究成果SR-IOV技术进行了介绍，并对其原理进行了分析，加以改进，进行测试，最后做出了评价。主要由以下几个部分组成：

1 研究了SR-IOV技术的产生背景、技术原理以及该技术面临的机遇和挑战

传统的基于软件和硬件辅助的虚拟化I/O方法虽然能够从不同角度提高虚拟化I/O的能力，但无法同时获得I/O设备的高性能和共享性。SR-IOV技术规范正是针对这一问题提出了相应的解决方法。

2 分析了SR-IOV在Linux和Xen平台上的实现机理

通过分析Linux内核和Xen的源代码，说明了SR-IOV设备使用的整体框架，并分别对SR-IOV硬件初始化模块、客户机0对VF的设置、抽象SR-IOV的VF设备和客户机N的VF驱动初始化进行了阐述。Xen通过启动了SR-IOV功能并创建多个VF，利用Passthrough方式将VF分配给客户机，从而达到共享SR-IOV设备和获得高性能的目标。

3 提出了一种扩展VF共享模型，用以解决了VF数量的局限性

通过在宿主机的IOVM中增加VFM模块（Virtual Function Manager），对VF进行实时管理，并根据客户机的优先级动态地进行分配。该模块能够有效地共享SR-IOV设备的VF，充分利用SR-IOV设备的资源，使得SR-IOV设备提供高性能和良好的可扩展性

4 提出了一种虚拟PF/VF通信模型，用以解决SR-IOV技术的硬件依赖性和客户机跨物理硬件平台的迁移问题，有效扩展了SR-IOV虚拟化技术的实用性。

本文通过对 VMM 进行修改，支持虚拟的 PF/VF 通信方式。由于 Xen 大体上是由用户端的控制面板（Control Panel）和设备模型（Device Model）以及下层的 Hypervisor（VMM）组成，因此该模型对这三部分进行扩展，避免了特殊的 SR-IOV 硬件支持。

5 对本文实现的 SR-IOV 模型进行评测

利用标准的网络性能测试工具 Netperf 对本文实现的 SR-IOV 模型进行了测试，包括 TCP_STREAM、TCP_RR、TCP_CRR 在本机以及客户机的测评结果。并得出相应结论：SR-IOV 实现模型能够可以达到几乎本机的性能，并具有良好的可扩展性。

6.2 工作展望

经过对 SR-IOV 的分析与改进，本文实现的 SR-IOV 技术具备 I/O 设备虚拟化的高性能和良好的可扩展性。本文的研究工作还存在以下不足:

1、实现尚不充分

论文提出了扩展 VF 模型和虚拟 PF/VF 通信模型，并基于 Linux 操作系统和 Xen 虚拟机管理器进行了基本实现，但很多扩展功能和实现的优化尚不充分，如结合 Split I/O 模型和 Passthrough I/O 模型，对 VF 动态分配算法的进一步改进，虚拟 PF/VF 驱动之间高效通信机制的进一步优化等。

2、测试尚不全面

论文对 SR-IOV 改进模型的实验还有些不足，仅对客户机使用 SR-IOV 改进模型的网络流量，网络应答，CPU 占用率进行了测试，这只能笼统的表现该模型的设备使用性能，未能表现该模型对 CPU 工作周期的占用情况和中断的使用情况。此外，在虚拟 PF/VF 通信方面，需要在真实虚拟机迁移应用场景下对本文的工作进行验证，这部分测试工作未能全面展开。

下一步的工作主要包括以下三个方面：

1、在 Xen 3.2.4 最新的虚拟机管理器平台上进行实现，进一步完善扩展 VF 模型中 VF 的动态分配机制和虚拟 PF/VF 通信机制的优化功能；

2、在虚拟机实际应用中进行网络和其他 I/O 设备访问的针对性测试，作为进一步改进 SR-IOV 模型的依据；

3、结合银河麒麟操作系统的轻量级虚拟机管理器，进行迁移和优化，并实现简单易用的 SR-IOV 配置和管理界面。

致 谢

在我的课题和硕士论文完成之际，谨向在我攻读硕士学位的过程中曾经指导过我的老师，关心过我的朋友，关怀过我的领导，以及所有帮助过我的所有人致以崇高的敬意和深深的感谢！

首先， 要衷心感谢我的导师戴华东教授！本次课题的选择和展开是和戴老师的指导分不开的。戴老师经常从自己繁忙的工作中抽出时间和我探讨课题的研究和具体的实现问题；在我的课题遇到问题时，总是能给我有益的启示，使我得以顺利的完成课题任务。戴老师系统的知识、丰富的工程经验和包容的人格魅力对我完成学业有很大帮助。

感谢吴庆波研究员，吴所长在百忙之中抽出时间听取我的研究情况，并给出了指导性的建议，对本课题的展开有很大的帮助。

感谢谭郁松老师和刘晓建老师，谭老师总是严格要求我们，并在课题的进程中对我们给予了必要的指导、帮助和督促。刘晓建老师工作细致，对我的问题总是悉心解答，使得我的课题可以顺利进行。

感谢颜跃进老师和任怡老师，颜老师在百忙之中抽出时间听取我的研究情况，并给出了建议，完善了本课题的研究。任老师在我前期学习中给与了很大的帮助。

感谢 620 教研室的所有老师以及各位工作人员，他们工作细致周密，为人温和，在课题的研究过程中给予我很多关心和帮助。

感谢杨昭君、舒畅、董青、姚远、孟凯凯、王春光、王文竹等同学，在过去的一年多时间里大家一起工作、学习，既有技术上的支持，又能感受到大家庭的温暖。

感谢王伟、朱浩、卢雪山、魏辉以及张钰森、汤慧明、林彬、李晓等各位师弟，在我的课题中给予的必要支持。

感谢学院各位师长，学院的领导，是他们在日常生活中的关心、教育使我顺利地完成了研究生阶段的学业；感谢继教二队的全体同学和我一起同甘共苦走过了研究生阶段的美好时光，给我留下了许多美好的回忆。

感谢我的父母在生活中给予我的关怀。他们对我的鼓励是促使我不断进取的巨大精神动力。

最后，再次向所有在我攻读硕士学位的过程中曾经指导我的老师，关怀我的领导，关心我的朋友和帮助过我的人致以真挚的谢意！

参考文献

[1] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In Proceedings of the ACM Symposium on Operating Systems Principles, pages 164–177, October 2008.

[2] Mendel Rosenblum, Tal Garfinkel. Virtual Machine Monitors Current Technology and Future Tends[J]. IEEE, 2005, 38(5): 39-47.

[3] Himanshu Raj and Ivan Ganev and Karsten Schwan and Jimi Xenidis. Self-Virtualized I/O: High Performance, Scalable I/O Virtualization in Multi-core Systems. Technical Report GIT-CERCS-06-02, CERCS, Georgia Tech, 2006.

[4] http://software.intel.com/file/1024

[5] Ariel Cohen。I/O Virtualization for Next-Generation Datacenters。Windows Hardware Engineering Conference 2007

[6] Paul Barham, Boris Dragovic, Keir Fraser. Xen and the art of virtualization[J]. ACM Press, 2003: 164-177.

[7] D Abramson,J Jackson，S Muthrasanallur,and et al. Intel Vtmmfization Technology for Directed I/O. Intell Technology Journal，Volume 10，Issue 3，2006

[8] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor[C]. In Proceedings of the USENIX Annual Technical Conference, 2007.6.

[9] J. Liu,W. Huang, B. Abali, and D. Panda.High performance vmm-bypass i/o in virtual machines[C]. In Proceedings of the USENIX Annual Technical Conference, 2006.5.

[10] P. Willmann, S. Rixner, and A. L. Cox, Protection Strategies for Direct Access to Virtualized I/O Devices, Proceedings of the USENIX Annual Technical Conference, Boston, MA, 2008, 15-28

[11] M Ben-Yehuda,J D Mason,O Krieger,J Xenidis. Xen/IOMMU：Breaking IO in. New and Interesting Ways[C]. Austin，Xen Summit,2006.

[12] Intel Corporation: Intel Virtualization Technology for Directed I/O. http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art02.pdf

[13] Sunay Tripathi , Nicolas Droux , Thirumalai Srinivasan , Kais Belgaied, Crossbow: from hardware virtualized NICs to virtualized networks, Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures,

August 17-17, 2009, Barcelona, Spain

[14] Intel VT Technology[EB/OL]. http://www.intel.com/technology/advanced_comm/virtualization.html.

[15] http://en.wikipedia.org/wiki/X86_virtualization#AMD_virtualization_.28AMD-V.29

[16] 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

[17] Intel Corporation. Intel virtualization technology for direct I/O: Intel technology Journal 10(03). September, 2008.

[18] Intel Virtualization Technology. http://www.intel.com/technology/itj/2006/v10i3/1-hardware/6-vt-x-vt-isolutions.Htm

[19] PCI SIG: PCI-SIG Single Root I/O Virtualization. http://www.pcisig.com/specifications/iov/single_root/

[20] PCI Special Interest Group, http://www.pcisig.com/home

[21] Joshua LeVasseur, Ramu Panayappan, Standardized but Flexible I/O for Self-Virtualizing Devices。In WIOV '08: Proceedings of the 1stWorkshop on I/O Virtualization, December 2008.

[22] I Pratt，K Fraser,S Hand,C L Limpach，A Warfiel&Xen 3．0 and the Art of Wmualization．Proceedings of the 2005 Ottawa Linux Symposium，2007

[23] K. Fraser, S. Hand , R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the Xen virtual machine monitor[C]. In 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure(OASIS), 2004.10.

[24] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing Performance Overheads in the Xen Virtual Machine Environment[C]. In First ACM/USENIX Conference on Virtual Execution Environments(VEE'05), 2005.6.

[25] G. Neiger, A. Santoni, F. Leung, D. Rodgers, R. Uhlig. Intel virtulization technology: Hardware support for efficient processor virtualization[C]. Intel Technology Journal, 2006.

[26] Barham P, Dragovic B, Frase K, Hand S, Harris T, Ho A, Neugebauer R, Pratt L, Warfield A. Xen and the art of virtualization: proceedings of 19th ACM Symposium on Operating Systems Principles, October, 2003.

[27] Liu Jx, Huang W, Abali B, K.Panda D. High performance vmm-bypass i/o in virtual machines: proceedings of the USENIX Annual Technical Conference, May, 2006.

[28] ZHAI, E., CUMMINGS, G. , AND DONG, Y. Live migration with pass-through device for linux vm. In Open LInux Symposium (2008), pp. 261–267.

[29] 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.

[30] Y. Dong, J. Dai, et al. Towards high-quality I/O virtualization. Proceeding of the Israeli Experimental Systems Conference (SYSTOR),Haifa, Israel 2009.

[31] PCI Special Interest Group, http://www.pcisig.com/home.

[32] PCI SIG: PCI-SIG Local Bus Specification 3.0.http://www.pcisig.com/members/downloads/specifications/conventional/PCI_LB3.0-2-6-04.pdf

[33] Intel® 82576 Gigabit Ethernet Controller. http://download.intel.com/design/network/ProdBrf/320025.pdf

[34] L. Cherkasova and R. Gardner. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. In USENIX Annual Technical Conference, Apr 2005.

[35] Aravind Menon , Alan L. Cox , Willy Zwaenepoel, Optimizing network virtualization in Xen, Proceedings of the annual conference on USENIX '06 Annual Technical Conference, p.2-2, May 30-June 03, 2006, Boston, MA

[36] 胡冷非,李小勇.基于 Xen 的 I/O 准虚拟化驱动研究. 计算机工程, V.35 No.23,2009 年 12 月

[37] Y. Dong, et al. SR-IOV Networking in Xen: Architecture, Design and Implementation. 1st Workshop on I/O Virtualization, San Diego, CA, 2008

[38] Fabrice Bellard, QEMU, a fast and portable dynamic translator, Proceedings of the annual conference on USENIX Annual Technical Conference, p.41-41, April 10-15, 2005, Anaheim, CA

作者在学期间取得的学术成果

[1] 李超,董青,戴华东.基于 SR-IOV 的 IO 虚拟化技术,电脑与信息技术,2010,10(已录用)

SR-IOV虚拟化技术的研究与优化

作者： 李超
学位授予单位： 国防科学技术大学

本文链接：http://d.g.wanfangdata.com.cn/Thesis_Y1795829.aspx