云数据中心 I/O 资源池化

王 展 安学军

（中国科学院计算技术研究所 北京 100190）

摘 要 近年来，云计算的发展为数据中心带来了新的应用场景和需求。其中，虚拟化作为云服务的重要使能技术，对数据中心服务器 I/O 系统的性能、扩展性和设备种类多样性提出了更高的要求，沿用传统设备与服务器紧耦合的 I/O 架构将会导致资源冗余，数据中心服务器密度降低，布线复杂度增加等诸多问题。因此，文章围绕 I/O 资源池化架构的实现机制和方法展开研究，目标是解除设备与服务器之间的绑定关系，实现接入服务器对 I/O 资源的按需弹性化使用，从根本上解决云计算数据中心的 I/O 系统问题。同时，还提出了一种基于单根 I/O 虚拟化协议实现多根 I/O 资源池化的架构，该架构通过硬件的外设部件高速互连接口多根域间地址和标识符映射机制，实现了多个物理服务器对同一 I/O 设备的共享复用；通过虚拟 I/O 设备热插拔技术和多根共享管理机制，实现了虚拟 I/O 资源在服务器间的实时动态分配；采用现场可编程门阵列（Field-Programmable Gate Array）构建了该架构的原型系统。结果表明，该架构能够为各个共享服务器提供良好的 I/O 操作性能。

关键词 云计算；数据中心；虚拟化；资源池；I/O 共享；动态分配
中图分类号 TP 334 **文献标志码** A

I/O Resource Pooling for Cloud Datacenter

WANG Zhan AN Xuejun

(*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190, *China*)

Abstract The emergence of cloud computing brought datacenter to new application environments and new requirements. Server virtualization, as the main enabling technology for cloud services, needed datacenter to increase the number, performance and variety of I/O devices to support more virtual machines, for which the traditional server I/O architecture could result in waste of resources, increasing of server density and rise of wiring complexity. I/O resource pooling was a promising architecture to solve these problems, by separating devices and servers and enabling elastic on-demand provisioning of I/O resources to servers. We designed and implemented a single root I/O virtualization based multi-root I/O resource pooling architecture. By using a hardware-based peripheral component interface express identifier re-mapping, different physical servers can share the same physical devices, and through a hot-plug based allocation method, virtual I/O device could be dynamically reallocated between servers. The correctness and performance of our pooling architecture was proved by a Field-Programmable Gate Array prototype in this paper.

收稿日期：2015-09-11 **修回日期**：2015-11-19
作者简介：王展（通讯作者），助理研究员，研究方向为高性能互连网络、虚拟化技术，E-mail：wangzhan@ncic.ac.cn；安学军，正研级高级工程师，研究方向为计算机系统结构、高性能互连网络。

Keywords cloud computing; datacenter; virtualization; resource pooling; I/O sharing; dynamical allocation

1 引 言

1.1 云计算改变传统信息技术模式

随着社会经济不断发展，用户的需求日益多样化，快速地响应市场的变化成为信息服务企业经营成功的关键因素。过高的初始投入和运维成本，以及相对固化的架构，使传统的信息化方法成为企业发展的桎梏[1]。

云计算的产生为企业信息化带来了新的思路。“云”是位于互联网络上的信息技术(Information Technology，IT)资源(如服务器、存储、操作系统或中间件、应用程序等)，由云计算服务提供商统一部署和管理。使用云计算的企业不需要再购置和维护自己的 IT 系统，而是通过网络向服务提供商以按需、自助的方式获得 IT 资源和服务，企业可以按照经营的需求便捷地更改其 IT 系统的配置[2]。同时，受益于规模经济和资源的多用户复用，云计算底层基础设施的单位资源成本仅为传统企业自建数据中心的 1/5～1/7[3]，这使得云计算服务的价格大幅降低，进而帮助企业降低了信息化成本。

云计算对传统 IT 使用模式的改变并不局限于企业信息化领域。越来越多的个人应用如视频播放、文字处理、文件存储等，也从传统的桌面模式转向了云模式；政府、大学等社会机构也逐渐将 IT 业务交付云计算平台完成以节省开支[4]；近年来因互联网的快速发展而催生的大数据问题，也依靠包含大规模硬件资源的云计算平台予以解决。云计算被认为是继个人电脑及互联网以来，第三次的 IT 浪潮，引领着信息产业的发展[5]：继 2006 年谷歌和亚马逊推出云计算服务，IBM、微软、EMC、Oracle 等 IT 巨头也纷纷跟进，中国企业也先后推出了阿里云、百度云、腾讯云等云计算服务；许多国家也都将云计算作为国家战略性新兴产业的重要组成部分，予以大力支持。

1.2 云计算的虚拟化特征

按照提供服务内容的不同，云计算可以分为基础架构即服务(Infrastructure as a Service，IaaS)、平台即服务(Platform as a Service，PaaS)和软件即服务(Software as a Service，SaaS)[5]。其中，IaaS 提供虚拟基础资源，如虚拟主机、存储等，用户以此为基础从底层搭建自定义的 IT 环境，典型代表包括 Amazon EC2[6]、Rackspace[7]、阿里云[8]等；PaaS 提供构建应用所需的套件，如编程工具、测试平台、运行时库等，使用户可以构建自定义应用而不需要购买和部署相关工具，典型代表包括 Google App Engine[9]、Microsoft Windows Azure[10]、Sina App Engine[11]等； SaaS 向用户提供远程的应用程序直接运行，典型代表包括 Google Apps[12]、Microsoft Online[13]、Oracle CRM on Demand[14]等。三类云计算服务之间及其与底层硬件设施之间的层次关系如图 1 所示。

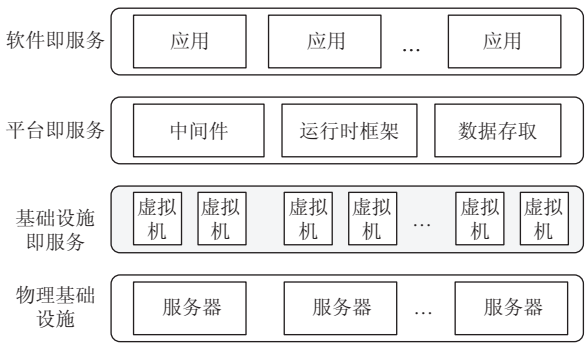


图 1 云计算不同类型服务和底层硬件设施的层次关系

Fig. 1 System hierarchy of cloud computing and physical infrastructure

可以看到，无论何种类型的云计算，都使用虚拟化＋数据中心的组合作为其服务平台搭建的

基础。其中，虚拟化部署于数据中心服务器机群之上，将单个物理服务器抽象成为多个虚拟机，为上层用户提供一套以虚拟机为中间媒介的硬件资源使用方式。

虚拟化[15,16]对云计算的基础硬件设施管理和服务体系建设都至关重要。对于前者，虚拟化可以提高硬件资源利用率，降低系统成本和能耗。如果将操作系统和应用直接部署在服务器硬件上，硬件就需要冗余配置以满足应用的峰值性能。相关统计显示，在应用的平均运行状态下，这种部署方式的硬件资源利用率仅有5%～20%[17,18]，大量部件都处于上电却空闲的状态，造成成本和功耗的浪费。使用虚拟化技术，原部署于不同服务器上的多个应用就可以混合部署到单个物理服务器内的多台虚拟机上，通过多虚拟机对底层硬件的共享复用，可以将硬件资源（主要是 CPU 和内存，后面会看到 I/O 资源的利用率仍然很低）利用率提高到 60%[19]以上，从而大大降低云计算的基础设施成本和能耗。

对于后者，虚拟化可以向用户提供：（1）弹性化的资源使用模式，用户可支配的资源就由虚拟机的数量和虚拟机的虚拟硬件配置决定，可以动态改变并与等量的底层硬件实时绑定；（2）良好的应用隔离，各个应用在不同的虚拟机操作系统中独立运行，即使某个应用出错导致其操作系统崩溃，所产生的负面效果也会被虚拟化软件所拦截，不会影响到其他应用[21]；（3）屏蔽硬件差异的应用部署环境，虚拟化技术向用户屏蔽底层基础设施由于不同批次采购或扩容产生的硬件差异，降低了用户软件的部署难度，提高了系统对用户的易用性。

1.3　虚拟化下传统 I/O 架构的缺陷

虚拟化的使用对数据中心服务器的 I/O 系统结构设计提出性能、扩展性和设备多样性三方面需求。首先，在性能方面，虚拟机作为物理服务器的替代，当其通过软件完全模拟或前/后端驱动的方式使用宿主服务器的物理 I/O 设备时，虚拟软件处理带来的 I/O 性能损失需要通过物理 I/O 设备数量或性能的提升来弥补[30,31]；其次，在扩展性方面，直接 I/O 虚拟化技术允许虚拟机通过独占宿主机物理设备来提高 I/O 访问效率，但该技术需要 I/O 设备的数量与虚拟机个数匹配，即对服务器的 I/O 系统扩展性提出需求[32]；最后，在设备多样性方面，虚拟化技术允许多种应用整合部署在单个物理服务器之上，这就要求宿主物理服务器配备更多类型的 I/O 设备以支持多种应用所对应的不同的存储、计算以及通信协议。

目前，云数据中心服务器仍然沿用着传统的计算机 I/O 系统结构，每台服务器都通过 I/O 总线与一定数量的设备形成排他连接，设备只能被其连接的服务器独占使用。在这种架构下，服务器只能以增加其机箱内物理设备的性能、数量以及种类来应对虚拟化的需求，这将产生如下问题：

（1）I/O 资源冗余：多虚拟机混合部署可以提高单台服务器的硬件资源利用率，为了满足虚拟的特殊 I/O 需求而增加设备的性能、数量、种类反而会进一步加重 I/O 资源的冗余；

（2）计算密度降低：虚拟化环境下，为了在单台服务器内容纳更多的 I/O 设备并保证良好的散热效果，服务器机箱需要变得更大，从而导致数据中心单位空间内的计算密度降低，提供同等计算能力所需的场地租用费用增加；

（3）布线复杂度增加：通信、存储、管理等多套网络的存在已经使传统数据中心的布线日趋复杂[33]，而虚拟化环境下网络接口卡的增加将使布线复杂度进一步提高。

近年来，超大规模集成电路（Very Large Scale Integration）技术的发展使得单个设备的能力越来越强，如果将原多个 I/O 设备的硬件逻辑集成在单个设备上，就可以大大降低服务器机箱内的 I/O 设备数量。这种思想推动了 I/O 聚合和 I/O 融合两类技术的产生和发展。I/O 聚合是通过设备

本身的硬件辅助虚拟化技术，将直接 I/O 虚拟化所需的多个物理 I/O 设备聚合转换为单个物理设备内的多个虚拟功能，以减少单个物理服务器所需配备的 I/O 设备数量，如 PCI-SIG 提出的 PCIe 单根虚拟化(Single Root I/O Virtualization，SR-IOV)技术[34]。I/O 融合是通过设备本身的硬件辅助协议转换，实现单个物理 I/O 设备支持多种协议数据包的处理和传输，从而降低数据中心所需配备的 I/O 设备和互连网络的多样性，如以太网光纤通道(Fiber Channel over Ethernet)技术[35]。

然而，这种单纯的设备端仅能减少单个服务器内的设备数量，由于服务器仍然通过排他的总线连接独占其机箱内的 I/O 设备，应用需求降低时产生的空闲 I/O 资源仍然得不到利用，而应用需求增加时需要的额外 I/O 资源也得不到补充。因此，应该从服务器与设备的连接关系，到设备的使用方式，对传统服务器的 I/O 系统结构做整体改变。

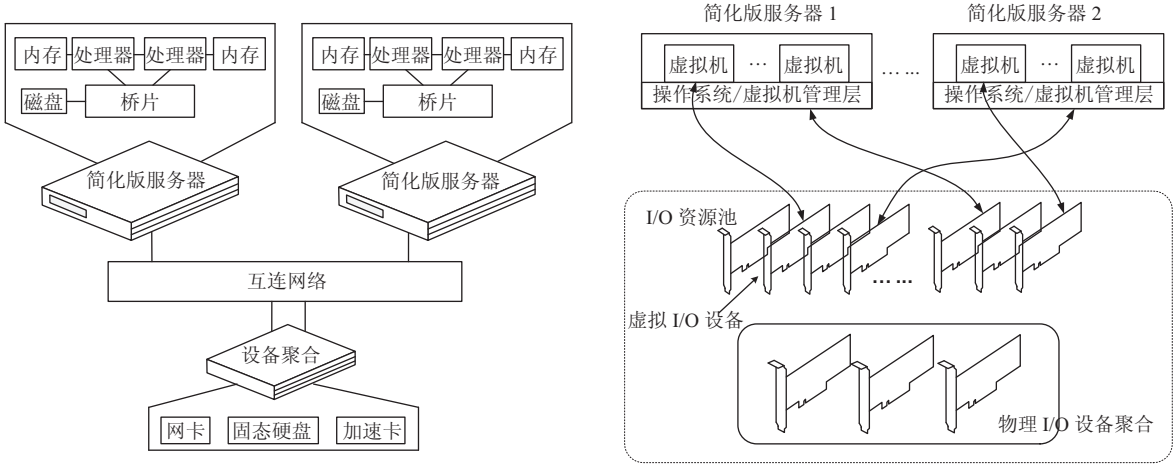
2 面向数据中心的 I/O 资源池化

2.1 I/O 资源池化架构概述

图 2 给出了一种全新的服务器 I/O 系统结构方案。其中，图 2(a)描述其物理结构：每台服务器的硬件被拆分为两部分，CPU 和内存仍保留在服务器机箱内，I/O 设备分离出来并统一归置；统一归置的 I/O 设备通过主机间互连网络与“简化版”的服务器连接，为任意服务器到任意设备的访问提供数据通路基础；图 2(b)描述该架构下的 I/O 资源使用方式：单个设备不再被某个服务器独占使用，通过架构的虚拟化技术，单个设备被抽象为多个虚拟设备，每个虚拟设备都可被某一服务器当作同功能的物理设备而使用，进而形成多台服务器对同一设备的共享复用。通过虚实的设备映射关系动态调整，架构可进一步实现 I/O 资源对服务器的按需分配使用。由于这种架构对 I/O 资源的聚合和再分配像水池对水的作用一样，我们就称其为面向多服务器的 I/O 资源池化架构。

I/O 资源池化架构是对传统服务器 I/O 系统结构的全面改进，其不但可以良好地解决传统结构的资源利用率问题和设备数量问题，还能为云计算平台带来如下两方面优势：

(1)硬件系统的高效升级和扩展。I/O 设备与 CPU 和内存的分离，使它们各自的升级和扩展都更加高效和灵活。在传统 I/O 架构下，升级多个



(a)资源池架构的物理结构 (b)资源池架构的 I/O 模式

图 2 面向多服务器的 I/O 资源池化架构

Fig. 2 Multi-server I/O resource pooling architecture

服务器的 I/O 硬件性能需要为每台服务器都配置高端设备，花费高昂；而在 I/O 资源池架构下，只需要购买少量的高端 I/O 设备就可以满足所有服务器性能升级需求。

(2) 虚拟机 I/O 资源的高效纵向扩展。云计算环境下，用户应用程序对虚拟机资源量的需求是动态变化的，传统的应对方法是增加或减少虚拟机的数量，被称为横向扩展(Horizontal Scaling)[36]。横向扩展的资源利用率还不高，粒度较大。Galante 等[37]指出，对于科学计算类应用，粗粒度的虚拟机数量增加仅能提高应用的并行性，而应用实际需要的是单个虚拟机处理能力的增强，也就是纵向扩展(Vertical Scaling)。同时，由于虚拟机的启动时间通常要花费数十到数百秒，横向扩展对一些要求资源动态变化响应时间在秒以内的应用是不适用的，这类应用也需要虚拟机的纵向扩展。传统数据中心服务器架构使虚拟机的纵向扩展受限于服务器本身资源量，而在 I/O 资源池架构下，I/O 资源量的限制得到了解除，虚拟机纵向扩展所需的额外 I/O 资源都可以从资源池中动态地获取。

目前，学术界和工业界对 I/O 资源池化架构并没有统一的结论，本节开始的描述也只是对其架构的一种概念上的说明，具体的设计和实现需要进一步的探索和研究。

2.2　I/O 资源池化的关键问题

实现数据中心内多服务器对底层物理 I/O 资源的池化使用，需要解决如下三方面的关键问题。

(1) 多根 I/O 互连：传统数据中心服务器使用 PCIe 总线连接处理器和 I/O 设备，根据标准 PCIe 协议的规定，服务器内所有组件必须互连为单根树形拓扑结构，作为树形拓扑端点的 I/O 设备也就只能与唯一的服务器根联合体形成连接通路，导致服务器与设备间的物理绑定。I/O 资源池化系统提供多个服务器对设备的按需共享，首先就要打破将这种单根互连结构，实现单个 I/O 设备到多个服务器根联合体的数据通路，即多根 I/O 互连。同时，为了兼容已有的处理器和 I/O 设备，多根互连结构需要兼容现有单根协议数据包的传输。

(2) 多服务器 I/O 共享：将传统仅用于单根环境中的 I/O 设备置于多根互连环境下，如果允许不同的服务器使用其各自的 PCIe ID 和内存地址空间地址对设备直接进行访问，不但会引起设备本身硬件逻辑的混乱，还有可能导致服务器操作系统的崩溃。因此，需要在多根 I/O 互连的基础上，为不同服务器共享使用同一 I/O 设备提供支持。

(3) I/O 资源的动态分配：在通常的共享环境下，资源由管理框架通过提前映射的方式静态部署，特别地，在本文所提及的多服务器间 I/O 共享环境下，资源需要由服务器启动设备枚举来发现并实用，如果仅支持静态部署，资源仍无法获得按需配置。因此，实现构建共享 I/O 资源对服务器的动态按需分配框架，最终达到“资源池化”的使用效果。

3　I/O 资源池系统的设计和实现

3.1　多根 PCIe 交换机

现有针对多根 I/O 互连的研究主要分为两类：基于系统域网络(如以太网、Infiniband)的间接扩展和基于 PCIe 总线的直接扩展。前者可以做到良好的软硬件兼容，系统可扩展以及 I/O 操作传输可靠性保证，但其性能无法满足云计算环境下的 I/O 资源池化需求；而后者虽然具备更优的互连性能，但对现有服务器硬件的改动较大，难以被工业界所接受。本节以后者为基础，提出一种基于 PCIe 总线协议扩展的多根交换机设计，在兼容现有处理器、设备接口以及数据包格式的基础上，实现多服务器与同一设备间的数据通路。

多根 PCIe 交换机的结构如图 3 所示，

基本组成部分包括：多个上游端口(Upstream Port，UP)，多个下游端口(Downstream Port，DP)，PCIe 配置空间(图中的 P2P)和交叉开关(Crossbar)。其中，每个上游端口通过标准 PCIe 接口接入独立的主机系统，端口中实现了完整的 PCIe 三层协议处理逻辑，其中包含了原 PCIe 的整套数据可靠性措施。此外，上游端口还在事务层额外增加了多根路由标识添加和删除功能。多根路由标识是代表主机连接的交换机上游端口号的标签，该标签在 PCIe 数据包从主机进入上游端口后添加进事务层包头，在数据包从上游端口进入主机前从事务层包头清除。

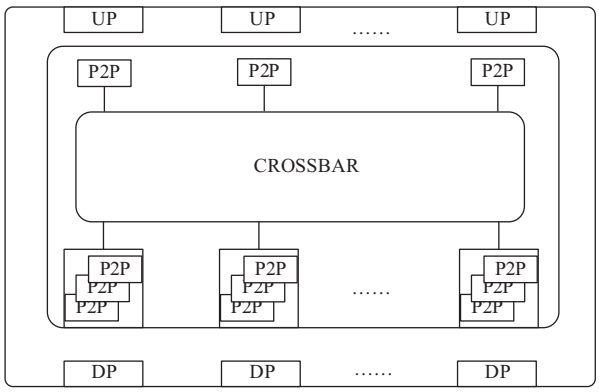


图 3 多根 PCIe 交换机结构

Fig. 3 Multi-root PCIe switch architecture

每个下游端口连接一个 I/O 虚拟化装置，该虚拟化装置又通过标准 PCIe 接口与一个真实的物理 I/O 设备相连；虚拟化装置主要用于解决多物理主机共享同一设备时的各种冲突，这将在下一节介绍。单纯的下游端口功能较为简单，主要充当了虚拟化装置与交换机内部交叉开关之间的数据通道，传输 PCIe 事务层数据包。

PCIe 配置空间实现两方面的功能：(1)兼容上游端口连接的物理主机操作系统的单根 PCIe 拓扑映射；(2)根据主机到设备方向的 PCIe 数据包携带的 PCIe ID 和地址信息，为其生成交叉开关的目的端口。配置空间的具体实现是多份且分布式的：(1)每个上游端口绑定一个 PCIe 配置空间，当端口连接的主机对其 PCIe 系统进行枚举时，该配置空间通过头类型寄存器和设备类型寄存器的值向主机表明自己是一个 PCI-PCI 上游桥设备，从而使主机认定这是系统 PCIe 树的一个分支，并以深度优先的方式对该分支继续枚举；(2)每个下游端口绑定了与上游端口数目相同的多个配置空间，与上游端口形成一一对应关系，其在主机枚举时都会通过头类型寄存器和设备类型寄存器的值向主机表明自己是一个 PCI-PCI 下游桥设备，从而使主机认定其下仍连接有 PCIe 组件，并以深度优先的方式继续枚举，进而使主机按照传统的单根方式枚举到下游端口连向的真实物理设备。

交叉开关负责各个端口之间数据的快速传递。为了消除队头阻塞，最大限度提高交换能力，采用商用交换机中常用的非阻塞交换结构，在此不作赘述。

3.2 多根 I/O 共享设计

服务器使用 PCIe I/O 设备，首先需要通过 BIOS 和操作系统对设备进行枚举并配置(即配置 PCIe 配置空间)，将设备映射到主机的内存地址空间，再通过设备驱动程序以内存地址读写的方式发起设备访问。如果不加处理地允许多个服务器使用同一设备，就会在设备枚举配置(Device Enumeration)和枚举配置完成后的内存映射 I/O 访问两处产生冲突，如图 4 所示。

本文在硬件的 PCIe 事务层对 I/O 操作过程中的数据包实施监控，采用 PCIe ID 映射用于解决多服务器 I/O 共享过程中的 ID 冲突。参与 I/O 共享服务器被划分为两类：Owner 和 User；与其对应，PCIe 事务层数据包的传输空间也可划分为两个域，分别为 Owner 域和 User 域，如图 5 所示。在 Owner 域，PCIe 事务层数据包将直接接触 Owner 服务器和 I/O 设备，因此必须使用 Owner 服务器为设备功能分配的 PCIe ID 和内存地址空间地址；而在 User 域，PCIe 数据包直接

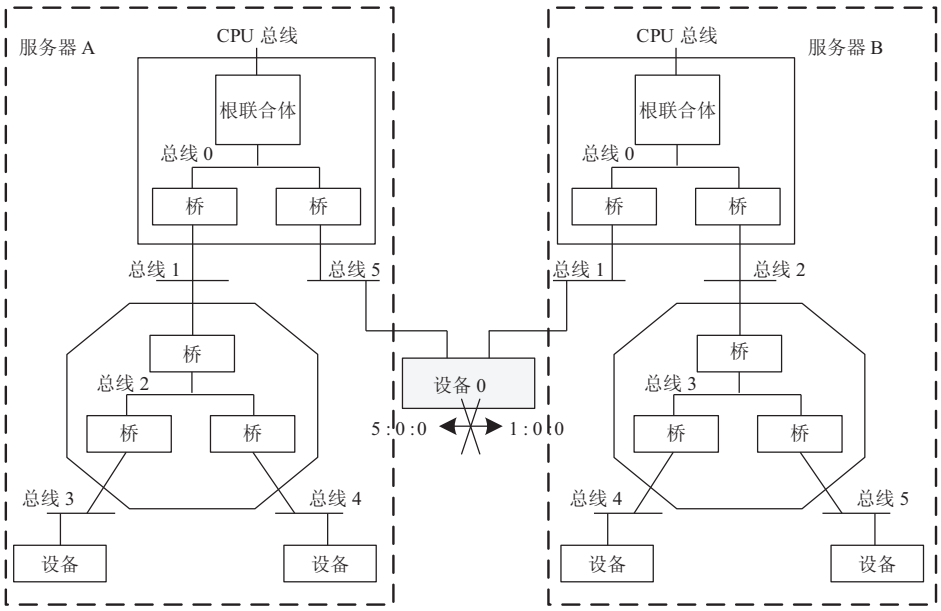


图 4　两个物理主机枚举同一设备时的冲突

Fig. 4　The conflict happens when two hosts enumerate the same device

接触的是 User 服务器，因此必须使用各个 User 服务器为其共享的设备功能分配的 PCIe ID 和内存地址空间地址。PCIe 事务层数据包在两个域之间通过，对其携带的 ID 和地址进行相互的转换。

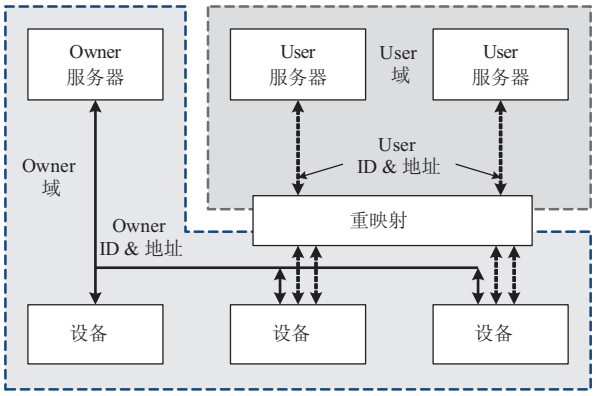


图 5　PCIe 事务层数据包在 Owner 域和 User 域间的划分

Fig. 5　Division of transaction packets between owner and user domain

PCIe ID 映射的逻辑结构设计如图 6 所示，分为下行 ID 映射和上行 ID 映射两部分。PCIe 事务层数据包从 User 域进入 Owner 域的 ID 转换称为下行 ID 映射。所有 User 服务器对共享设备功能进行访问的事务层数据包，包括 PCIe 配置空间读写包和内存映射 I/O 读写包都携带 User 服务器的 PCIe ID，都需要做下行 ID 映射。PCIe 事务层数据包从 Owner 域进入 User 域的 ID 转换称为上行 ID 映射。所有共享设备对 User 服务器的请求响应包，和共享设备读取 User 服务器内存数据的 PCIe 读写包初始都携带 Owner 服务器的 PCIe ID，都需要进行做上行 ID 映射。

在 PCIe ID 映射的基础之上，配置空间模拟帮助 User 服务器在对设备功能的 PCIe 配置空间读写过程中获得正确的结果。配置空间模拟在设备与 User 服务器之间创造了一个配置空间虚拟层，该虚拟层将代表真实设备功能的配置空间对 User 服务器的配置读写操作做出正确的响应，如图 7 所示。

内存地址映射用于解决地址冲突问题，当不同 User 服务器使用各自的内存地址空间访问同一 I/O 设备时，会引发地址冲突问题。下行内存地址映射使用了内容寻址存储器（Content Addressed Memory，CAM）和随机读写存储器

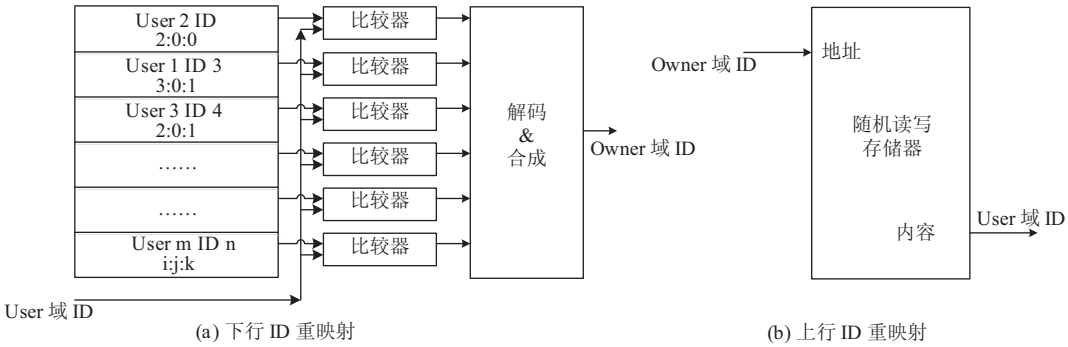


图 6 PCIe ID 映射逻辑结构

Fig. 6 PCIe ID mapping structure

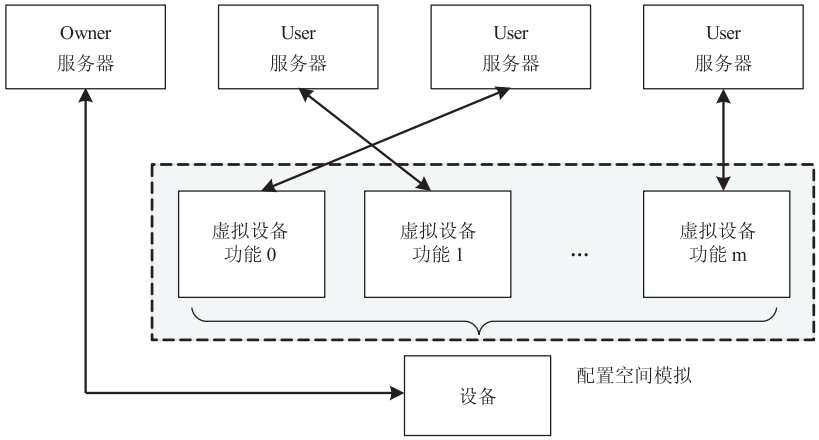


图 7 设备配置空间模拟

Fig. 7 Configure space emulation structure

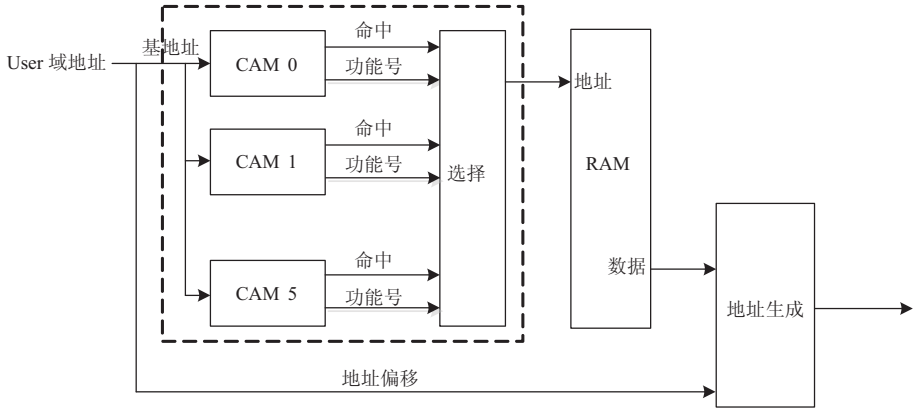


图 8 下行地址映射逻辑结构

Fig. 8 Downstream address mapping structure

(Random Accessed Memory，RAM)结合的设计，如图 8 所示，6 个并联 CAM 分别单独对应一个基址寄存器(PCIe 配置空间有 6 个基址寄存器)：CAM 中存储的是设备功能通过相应基址寄存器在各个 User 服务器中映射内存空间的基地址；与 CAM 串联的 RAM 中则存储设备功能通过各基址寄存器在 Owner 服务器中映射内存空间的基地址。下行 PCIe 事务层数据包携带的 User 服务器的内存地址空间地址：首先与基址掩码做与操作得到基地址和地址偏移，基地址作为并联 CAM 的输入，比对获得该数据包指向的真实设备功能号，该功能号与命中的 CAM 号的联合作为地址索引 RAM，获得设备功能在 Owner 域的基地址，该基址又与原偏移一起合成数据包进入 Owner 域需要的内存地址空间地址。

3.3　I/O 资源动态分配

在本文已设计的多服务间 I/O 共享框架下，各参与共享的服务器对真实物理资源的取用，是以虚拟设备功能作为中间媒介而完成的，如图 9 所示。因此，实现物理资源对各共享服务器的按需动态分配，本质上就是要实现虚拟设备功能对不同服务器操作系统的动态绑定。

为了做到对操作系统和上层软件的完全透明，虚拟设备功能兼容现有服务器 BIOS 和操作系统的设备枚举配置过程。该过程将虚拟设备功能加入到系统的逻辑 PCI 树中，并为其分配用于内存映射 I/O 操作的地址资源，从而实现功能与服务器操作系统的绑定；而执行相应的逆过程，清除虚拟设备功能在逻辑 PCI 树中的占位和系统内存地址空间中的映射，就能解除功能与服务器操作系统的绑定。当前，设备枚举配置过程及其逆过程的动态执行只能通过 PCIe 标准热插拔机制来触发。

本文设计虚拟热插拔引擎实现虚拟功能对不同服务器的动态分配，虚拟热插拔引擎与每个物理 PCIe 设备的多根 I/O 虚拟化引擎以及相关下游端口内的所有 PCI-PCI 下游桥绑定，并在热插拔过程中配合修改多根 I/O 虚拟化引擎中相关功能模块的硬件状态和 PCI-PCI 下游桥配置空间的相关寄存器。

具体而言，在热插过程中，引擎的主要功能包括：(1)根据被插入虚拟功能在 Owner 域中的 PCIe 功能号，向多根 I/O 虚拟化引擎的 PCIe ID 映射表相关表项中该功能在 User 服务器内 PCIe ID，从而为即将使用该虚拟功能的 User 服务器提供正确的 PCIe ID 映射；(2)向虚拟功能插入的

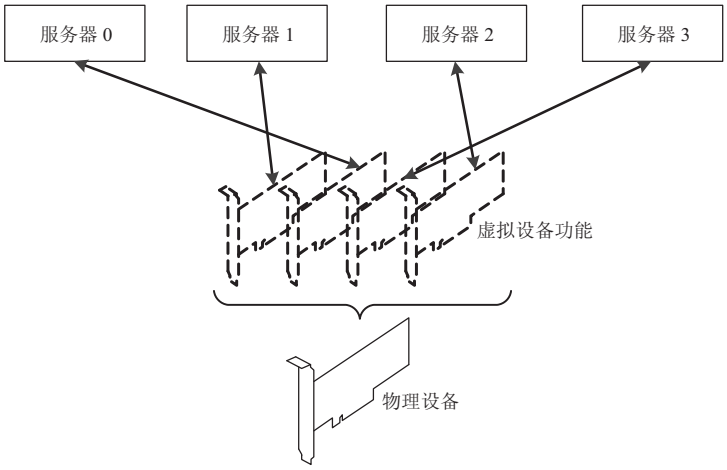


图 9　共享服务器通过虚拟设备功能实现对物理资源的取用

Fig. 9　Different servers share the same physical device through virtual device function

User 服务器所对应的 PCI-PCI 下游桥配置空间的插槽状态寄存器中写入热插事件，该事件将被对应 User 服务器操作系统内核捕捉，触发内核对插入的虚拟设备功能进行枚举和配置。

在热拔过程中，虚拟热插拔引擎的主要功能包括：(1) 向虚拟功能被拔出的 User 服务器所对应的 PCI-PCI 下游桥配置空间的插槽状态寄存器中写入热拔事件，该事件将被对应 User 服务器操作系统内核捕捉，触发内核清除被拔出虚拟设备功能占用的 PCIe ID 以及内存地址空间，该过程处理完成后，User 服务器操作系统将改变插槽状态寄存器的值以使热插拔控制器获得完成通知；(2) 清除被拔出的虚拟功能在多根 I/O 虚拟化引擎中占用的 PCIe ID 映射表项，内存地址映射表项以及配置空间模拟结构体，以腾出硬件资源为其他执行热插操作的虚拟功能所使用。

4 原型系统实现和评测

本文使用 Xilinx Virtex6 ff365t FPGA (Field-Programmable Gate Array， 现场可编程门阵列) 实现了设计的原型系统，如图 10 所示，其向外部呈现 3 个 PCIe 接口。其中，标注为 CPU Port 0 和 Port 1 的 PCIe Gen2 x8 金手指接口和 PCIe Gen2 x8 线缆接口，分别用于连接 Owner 服务器和 User 服务器；标注为 IO Port 的 PCIe Gen2 x8 的插槽接口用于连接设备。

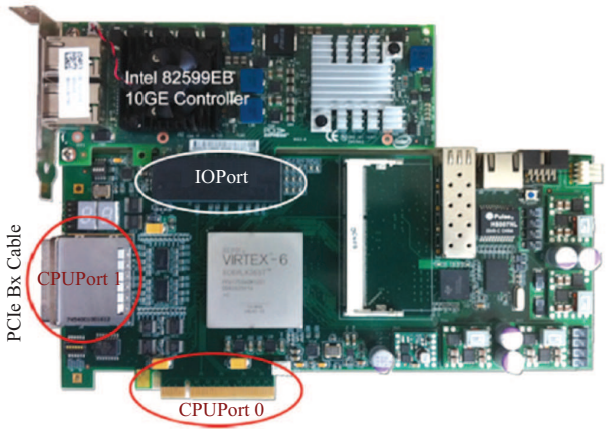


图 10 I/O 资源池化架构的 FPGA 原型系统

Fig. 10 FPGA prototype of our I/O resource pooling architecture

测试平台搭建如下，两台物理服务器通过原型系统共享同一块 Intel 82599 万兆以太网卡，该网卡支持 PCIe SR-IOV 协议。服务器 1 是 I/O 资源池化系统中的 Owner，服务器 2 则是 User。服务器 3 是测试服务器，它通过 Intel 82599 万兆以太网卡与 I/O 资源池系统直接互连，形成本地局域网络，以测试 I/O 资源池化系统的网络 I/O 功能和性能。图 11 给出了测试平台的搭建框图。

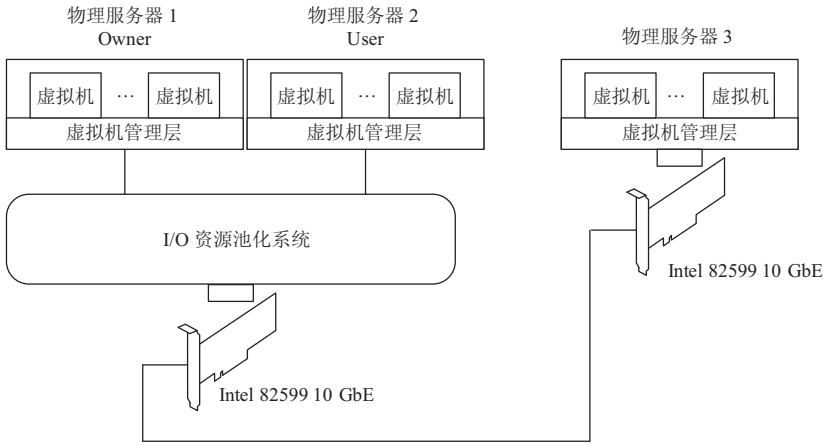


图 11 测试平台搭建框图

Fig. 11 Testbed for our prototype

表 1 给出了测试平台服务器的软硬件配置。

表 1　测试平台服务器软硬件配置

Table 1　Hardware & software configuration of test servers

硬件	型号
OS	CentOS 6.0(内核版本 2.6.32)
CPU	Intel i5-3470@3.20GHz
Memory	8G
Chipset	Intel Q77
PCIe Slot	Gen2 x4

4.1　功能测试

首先验证原型系统功能的正确性和完整性，而根据本文 I/O 资源池化架构的概念定义，评测包括三方面：

(1) Owner 服务器和 User 服务器都能够正确发现并配置原型系统为其分配的网络 I/O 资源。具体来说，Owner 服务器可以直接发现并配置物理 82599 网卡，并为其正确加载驱动，而 User 服务器可以发现由原型系统为其呈现的虚拟网卡，并将该网卡识别为 Intel 82599 SR-IOV 虚拟功能，为其加载 Intel 82599 虚拟功能驱动。图 12 和 13 分别给出了 Owner 服务器和 User 服务器上的 PCIe 设备加载结果，显示该项功能正确。

(2) 在设备枚举配置过程完成之后，Owner 和 User 服务器各自承载的应用可以正确调用 Intel 82599 网卡驱动发起网络通信操作，并获得正确的操作结果。图 14(a) 给出了在 User 服务器内 ping 服务器 3 的 IP 地址 169.254.9.153 得到的结果，(b) 则给出了在 User 服务器和服务器 3 之间运行更复杂的网络应用，如 ssh 和 scp 得到的结果。

```
[****@Owner ~]$ lspci -vt
-[0000:00]-+-00.0  Intel Corporation Sandy Bridge DRAM Controller
           +-01.0-[01-04]--+-00.0  Xilinx Corporation Device 8086
           |               \-00.1-[02-04]----00.0-[03-04]--+-[0000:04]-+-10.0  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-10.2  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-10.4  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-10.6  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-11.0  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-11.2  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-11.4  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-11.6  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-12.0  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-12.2  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-12.4  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-12.6  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-13.0  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-13.2  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-13.4  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               |          +-13.6  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                               -[0000:03]-+-00.0  Intel Corporation 82599EB 10 Gigabit TN Network Connection
           |                                                          -00.1\Intel Corporation 82599EB 10 Gigabit TN Network Connection
           +-02.0  Intel Corporation Integrated Graphics Controller
           +-19.0 …
           +-1a.0 …
           +-…
           \-1f.5 …
```

图 12　Owner 服务器内的 PCIe 逻辑映射信息

Fig. 12　The PCIe tree in Owner server

```
[****@User ~]$ lspci -vt
-[0000:00]-+-00.0  Intel Corporation Sandy Bridge DRAM Controller
           +-01.0-[01-04]--+-00.0  Xilinx Corporation Device 8086
           |               \-00.1-[02-03]----00.0-[03]--+-00.0  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                           +-00.1  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                           +-00.2  Intel Corporation 82599 Ethernet Controller Virtual Function
           |                                           -00\3  Intel Corporation 82599 Ethernet Controller Virtual Function
           +-02.0  Intel Corporation Integrated Graphics Controller
           +-19.0 …
           +-1a.0 …
           +-…
           \-1f.5 …
```

图 13　User 服务器内的 PCIe 逻辑映射信息

Fig. 13　The PCIe tree in User server

(3)User 服务器可使用的虚拟功能的个数可以通过原型系统提供的虚拟热插拔功能灵活动态地改变。具体来说，可以通过在 Owner 或 User 服务器内以软件命令的形式启动虚拟热插拔过程，在一定时间后，User 服务器内可用的虚拟功能个数就会根据命令的设定自动改变。

4.2 性能测试

性能测试首先关注系统在多服务器 I/O 资源共享方面的性能表现，主要包括应用软件层面的 I/O 带宽和延迟性能，以及共享环境下 I/O 资源分配的公平性。

本文使用网络延迟测试程序 ping 来测试 I/O 资源池化原型系统的应用层网络 I/O 延迟。在表 2 给出的测试结果中，每个表项中的前后两个数字分别代表服务器在不使用和使用本文 I/O 资源池化原型系统的情况下，执行 ping 操作的延迟。由于每次 ping 操作都是单向的，所以会有类似 Server 1/2→Server 3，Server 3→Server 1/2 这样的两组数据。

不使用 I/O 资源池化原型系统时，Server 1 和 Server 2 分别装备一个 82599 网卡与 Server 3 直接通信；而使用原型系统时，Server 1 和 Server 2 共享一块 82599 网卡与 Server 3 通信。表中的数据显示，共享带来的延迟增量约为 3～6 μs，与延迟总量相比，仅占 1.5%～3%。因此，在实际应用运行中，I/O 资源池化的共享处理带来的额外延迟基本可以忽略不计。

本文使用网络性能测试工具 iperf 来测试 Server 1 和 Server 2 通过 I/O 资源池化原型系统共享使用网卡时的 TCP 带宽性能，包含两个测试场景：

```
[****@User ~]$ ping 169.254.9.153
PING 169.254.9.153 (169.254.9.153) 56(84) bytes of data .
64 bytes from 169.254.9.153: icmp_seq=1 ttl=64 time=0.606 ms
64 bytes from 169.254.9.153: icmp_seq=2 ttl=64 time=0.191 ms
64 bytes from 169.254.9.153: icmp_seq=3 ttl=64 time=0.196 ms
64 bytes from 169.254.9.153: icmp_seq=4 ttl=64 time=0.188 ms
64 bytes from 169.254.9.153: icmp_seq=5 ttl=64 time=0.199 ms
64 bytes from 169.254.9.153: icmp_seq=6 ttl=64 time=0.243 ms
--- 169.254.9.153 ping statistics ---
6 packets transmitted , 6 received , 0% packet loss , time 5331ms
rtt min/avg/max/mdev = 0.188/0.270/0.606/0.152 ms
```

(a) 在 User 服务器内 ping 服务器 3 得到的结果

```
[****@User ~]$ ssh ****@169.254.9.153
****@169.254.9.153's password :
Last login: Thu Aug ** **:**:** 2013 from 169.254.6.193
[****@Server3 ~]$ scp iperf.iso ****@169.254.6.193:/home/****/
The authenticity of host  '169.254.6.193 (169.254.6.193)' can't be established .
RSA key fingerprint is  43:42:e8:aa:68:cf:bb:20:dd:c4:d6:7e:1b:30:99:23.
Are you sure you want to continue connecting   (yes/no)? yes
Warning: Permanently added  '169.254.6.193' (RSA) to the list of known hosts .
****@169.254.6.193's password :
iperf.iso    100%  430KB 430.0KB/s   00:00
```

(b) 在 User 服务器内 ssh 登陆服务器 3 并执行远程文件拷贝的结果

图 14 在 User 服务器内执行复杂网络操作得到的结果

Fig. 14 Network operation result in User server

表 2 测试服务器之间的 ping 操作延迟

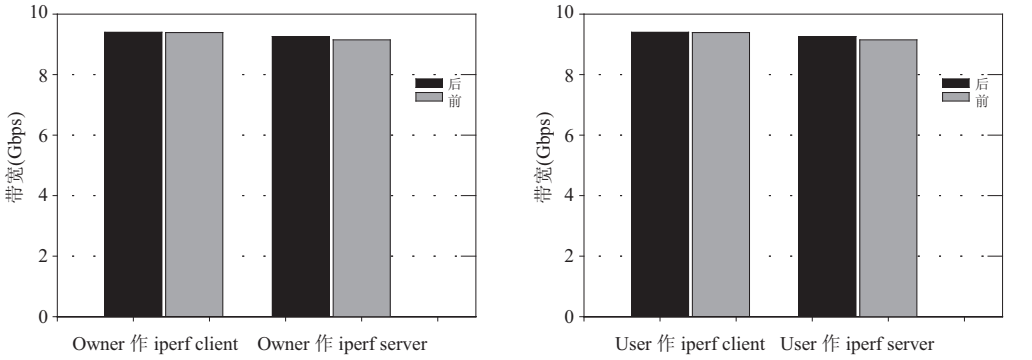
Table 2 The ping delay between different test servers

	Server 1 (Owner)	Server 2 (User)	Server 3
Server 1 (Owner)	— / —	— / 116 μs	202 μs / 206 μs
Server 2 (User)	— / 115 μs	— / —	203 μs / 208 μs
Server 3	201 μs / 204 μs	201 μs / 207 μs	— / —

场景一：Owner 服务器(Server 1)和 User 服务器(Server 3)分别独立通过原型系统调用网卡与 Server 3 进行 iperf 测试的 TCP 带宽数据，与之对比的是 Owner 服务器和 User 服务器分别独立使用各自机箱内的直连网卡与 Server 3 进行 iperf 测试的 TCP 带宽数据。由于 iperf 测试是双向的，即要求测试环境中有一端作为 iperf server 监听到达的数据传输请求，另一端作为 iperf client 发起传输会话，server 端和 client 端的网络操作不同，测试得到的带宽也会不同，所以会有服务器分别作为 iperf server 和 client 的区分。

在图 15 给出的测试数据中，服务器直接使用 82599 网卡作为 iperf client 时的 TCP 带宽为 9.4 Gbps，作为 iperf server 时的 TCP 带宽为 9.26 Gbps；而服务器通过原型系统以 Owner 的身份共享使用网卡时，在 iperf client 端测得 TCP 带宽为 9.39 Gbps，在 iperf server 端测得 TCP 带宽为 9.15 Gbps。使用原型系统前后的 TCP 带宽性能几乎相同；当服务器通过原型系统以 User 的身份共享使用网卡时，测试结果也与 Owner 基本一致。这就证明：I/O 资源池化在提供多服务器 I/O 资源共享特性的同时，还能保证与原 I/O 架构几乎相同的应用层 I/O 带宽性能；同时，在 I/O 资源池化架构设计中，Owner 和 User 服务器在共享设备方面的性能基本相同。

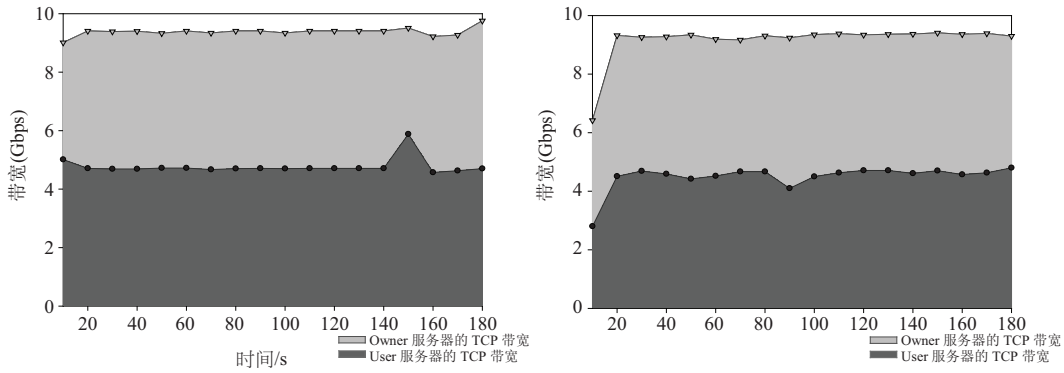
场景二：Owner 服务器(Server 1)和 User 服



(a) 使用本文原型系统前后的 Owner 服务器 TCP 带宽对比　(b) 使用本文原型系统前后的 User 服务器 TCP 带宽对比

图 15　Owner 和 User 服务器独立使用原型系统的 TCP 带宽对比

Fig. 15　TCP throughput comparison between Owner & User



(a) Owner 和 U ser 服务器作为 iperf client 时的 TCP 带宽　(b) Owner 和 U ser 服务器作为 iperf server 时的 TCP 带宽

图 16　Owner 和 User 同时使用共享网卡做 iperf 测试时的 TCP 带宽情况

Fig. 16　Sharing NIC TCP throughput comparison between Owner & Usera

务器(Server 2)同时共享使用网卡与 Server 3 进行 iperf 通信测试。其中，Owner 服务器使用 82599 网卡的物理功能；User 服务器使用资源池系统为其分配的 SR-IOV 虚拟功能；Owner 和 User 服务器作为 iperf client 和 iperf server 时的 TCP 带宽分别如图 16(a)和(b)所示：

Owner 服务器(Server 1)与 User 服务器(Server 2)作为 iperf client 时的平均 TCP 带宽分别为 4.78 Gbps 和 4.61 Gbps，作为 iperf server 时的平均 TCP 带宽分别为 4.66 Gbps 和 4.49 Gbps，它们的带宽总和基本与独立物理网卡的带宽峰值相当。该结果表明在本文的 I/O 资源池化架构下，服务器之间对共享设备的竞争不会导致设备性能的下降，而且当两台服务器都通过单个设备功能(Owner 使用单个物理功能，User 使用单个 SR-IOV 虚拟功能)使用共享设备时，可以获得基本公平的带宽分配。

5 相关研究

面向泛型 PCIe I/O 设备服务器间共享的相关研究可以分为基于软件和基于硬件两类。其中，软件类以本地 I/O 操作通过网络发送至远程执行的方法为主，如 IBM 海法研究实验室的 Satran 等[38]提出的名为 Scalable I/O 的基于软件操作代理的多主机间 I/O 共享方法，一台配备 I/O 设备的服务器通过类远程过程调用向其他服务器提供 I/O 操作服务。然而，软件类方法由于经过较多的软件层次间的上下文切换以及数据拷贝，将导致共享 I/O 主机的 I/O 性能较低。

硬件类主要以 I/O 总线协议扩展以实现不同服务器间的 I/O 共享。NEC 的 ExpEther[39]使用以太网将 PCIe I/O 设备的共享范围从单个服务器内部扩展到多个服务器之间，以太网的两端使用 PCIe-以太网协议转换桥分别连接服务器和 I/O 设备，该协议转换桥负责实现多个服务器对同一设备的共享使用。Suzuki 等[40]在 ExpEther 的基础之上，支持以 SR-IOV 设备的虚拟功能作为多服务器间 I/O 共享的资源分配实例，兼顾了 SR-IOV 技术的 I/O 聚合特性。文献[39]事实上是设备网络化技术，由于服务器与 I/O 设备之间的数据交换需要经过 PCIe 和以太网之间的协议转换，增加了系统 I/O 操作的开销和延时。PCI-SIG 提出了 MR-IOV[41](多根虚拟化)技术，通过对标准 PCIe 协议修改和扩展，使得支持 MR-IOV 协议的处理器和设备可原生支持 I/O 的多服务器间共享。 但该技术要求计算机 I/O 系统的各个部分(包括芯片组和设备)都按照 MR-IOV 协议进行修改实现，较大的系统改动，导致其未得到普遍应用。NextIO 公司关注于数据中心单机架内所有服务器间的 I/O 设备共享，其设计的 vNET I/O[42]将单个机架所需的以太网卡和 HBA 卡整合部署在一个 I/O 箱(I/O Box)内。该 I/O Box 可将单个物理设备虚拟为多个虚拟设备，供多个服务器共享使用。服务器只需使用普通的 PCIe 扩展适配卡连接到该箱，无需修改上层操作系统和设备驱动。vNet I/O 可以简化数据中心 I/O 设备部署和内部以太网及光纤网络的连线数量，提高系统的 I/O 管理效率。但是，vNet I/O 没有利用 SR-IOV 设备本身即可提供多个虚拟设备的特性，技术设计存在冗余。其他如 Xsigo[43]、Virtensys[44]公司也有类似的产品，但要么存在类似文献[39,40]的协议转换开销问题，要么限定可共享 I/O 设备种类，无法实现设备多样性需求。

本文提出 I/O 资源池化方法克服了上述问题，并利用 SR-IOV 设备自身的虚拟化特性优化了整个 I/O 资源池化共享框架，不引入任何协议转换开销，不限制 I/O 设备的功能类型。

6 总 结

为了提高云计算虚拟化环境下数据中心 I/O

系统的灵活性，提高资源利用率，本文提出了一种面向云数据中心的 I/O 资源池化架构。该架构充分考虑了对现有服务器软硬件和 I/O 设备的兼容性，采用原型系统对该架构进行了验证。结果表明，以标准 PCIe 体系结构为基础，在 PCIe 事务层对传统 I/O 架构的互连模式和操作模式实现了原生扩展；基于 PCIe 事务层处理的硬件化实现，也使该架构能够为各个共享服务器提供良好的 I/O 操作性能。

参 考 文 献

[1] Adamides ED, Karacapilidis N. Information technology support for the knowledge and social processes of innovation management [J]. Technovation, 2006, 26(1): 50-59.

[2] Buyya R, Yeo CS, Venugopal S. Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities [C] // Proceedings of IEEE International Conference on High Performance Computing and Communications, 2008: 5-13.

[3] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing [J]. Communications of the ACM, 2010, 53(4): 50-58.

[4] Rosenberg J, Mateos A. The Cloud at Your Service [M]. New York: Manning Publications, 2010.

[5] NIST Special Publication 800-145. The NIST definition of cloud computing [S]. 2009.

[6] Amazon elastic compute cloud (EC2) [EB/OL]. [2015-11-20]. http://aws.amazon.com/ec2/.

[7] Dedicated server, managed hosting, web hosting by rackspace hosting [EB/OL]. [2015-11-20]. http://www.rackspace.com/.

[8] 阿里云-打造数据分享第一平台 [EB/OL]. [2015-11-20]. http://www.aliyun.com/.

[9] Google App Engine [EB/OL]. [2015-11-20]. http://code.google.com/appengine/.

[10] Microsoft Windows Azure Platform [EB/OL]. [2015-11-20]. http://www.microsoft.com/windowsazure/.

[11] Sina App Engine [EB/OL]. [2015-11-20]. http://sae.sina.com.cn/.

[12] Google Apps [EB/OL]. [2015-11-20]. http://www.google.com/apps/.

[13] Microsoft business productivity online services and software (BPOS) [EB/OL]. [2015-11-20]. http://www.microsoft.com/online/.

[14] Oracle CRM on demand [EB/OL]. [2015-11-20]. http://www.oracle.com/us/products/applications/crmondemand/index.html.

[15] Rosenblum M, Garfinkel T. Virtual machine monitors: Current technology and future trends [J]. Computer, 2005, 38(5): 39-47.

[16] Smith JE, Nair R. Virtual Machines: Versatile Platforms for Systems and Processes [M]. Amsterdam: Elsevier, 2005.

[17] Rangan K, Cooke A, Post J, et al. The Cloud Wars: $100+ Billion at Stake [Z]. Merrill Lynch, 2008.

[18] Siegele L. Let it Rise: A Special Report on Corporate IT [M]. London: Economist Newspaper, 2008.

[19] Padala P, Shin KG, Zhu X, et al. Adaptive control of virtualized resources in utility computing environments [C] // Proceedings of the 2nd ACM SIGOPS Operating Systems Review, 2007, 41(3): 289-302.

[20] OpenStack. Cloud software [EB/OL]. [2015-11-20]. http://www.openstack.org.

[21] Matthews JN, Hu W, Hapuarachchi M, et al. Quantifying the performance isolation properties of virtualization systems [C] // Proceedings of the 2007 Workshop on Experimental Computer Science, 2007: 289-302.

[22] Rochwerger B, Breitgand D, Levy E, et al. The reservoir model and architecture for open federated cloud computing [J]. IBM Journal of Research and Development, 2009, 53(4):1-11.

[23] VMWare Server [EB/OL]. [2015-11-20]. http://www.vmware.com.

[24] VirtualBox [EB/OL]. [2015-11-20]. http://www.virtualbox.org/.

[25] Open source processor emulator (QEMU) [EB/OL]. [2015-11-20]. http://wiki.qemu.org/.

[26] Microsoft Hyper-V server [EB/OL]. [2015-11-20].

http://www.microsoft.com/brasil/servidores/hyper-v-server/default.mspx.

[27] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization [C] // Proceedings of ACM SIGOPS Operating Systems Review, 2003, 37: 164-177.

[28] Abramson D, Jackson J, Muthrasanallur S, et al. Intel® virtualization technology for directed I/O [J]. Intel Technology Journal, 2006, 10(03): 179-192.

[29] AMD I/O Virtualization Technology (IOMMU) Specification [Z]. AMD, 2006.

[30] Shafer J. I/O virtualization bottlenecks in cloud computing today [C] // Proceedings of the 2nd Conference on I/O Virtualization, 2010.

[31] Wang GH, Ng TSE. The impact of virtualization on network performance of Amazon EC2 data center [C] // Proceedings of IEEE International Conference on Computer Communications, 2010: 1-9.

[32] Cohen A. I/O virtualization for next-generation datacenters [EB/OL]. [2015-11-20]. download.microsoft.com/.../d/f/6/.../winhec2007_io-virt.doc.

[33] Farrington N, Rubow E, Vahdat A. Data center switch architecture in the age of merchant silicon [C] // Proceedings of the 17th IEEE Symposium on High Performance Interconnects, 2009: 93-102.

[34] PCI-SIG. Single root I/O virtualization and sharing 1.1 specification [EB/OL]. [2015-11-20]. http://www.pcisig.com/specifications/iov/single_root/.

[35] Fibre channel over Ethernet [EB/OL]. [2015-11-20]. http://en.wikipedia.org/wiki/Fibre_Channel_over_Ethernet.

[36] Chieu TC, Mohindra A, Karve AA, et al. Dynamic scaling of web applications in a virtualized cloud computing environment [C] // Proceedings of IEEE International Conference on e-Business Engineering, 2009: 281-286.

[37] Galante G, De Bona LCE, Mury AR, et al. Are public clouds elastic enough for scientific computing? [C] // Service-Oriented Computing–ICSOC 2013 Workshops, 2014: 294-307.

[38] Satran J, Shalev L, Ben-Yehuda M, et al. Scalable I/O-a well-architected way to do scalable, secure and virtualized I/O [C] // Proceedings of the First Conference on I/O Virtualization, 2008: 3.

[39] Suzuki J, Hidaka Y, Higuchi J, et al. Expressether-ethernet-based virtualization technology for reconfigurable hardware platform [C] // Proceedings of the 14th IEEE Symposium on High-Performance Interconnects, 2006: 45-51.

[40] Suzuki J, Hidaka Y, Higuchi J, et al. Multi-root share of single-root I/O vritualization(SR-IOV) compliant PCI express device [C] // Proceedings of 18th IEEE Symposium on High Performance Interconnects, 2010: 25-31.

[41] Multi-root I/O virtualization and sharing 1.0 specification [EB/OL]. [2015-11-20]. http://www.pcisig.com/specifications/iov/multi-root/.

[42] Pettey CJ, Khan A, Pagan A, et al. Apparatus and method for sharing I/O endpoints within a load store fabric by encapsulation of domain information in transaction layer packets: US. 7188209 [P]. 2007-06-03.

[43] Shah S, Vinod S, Anand RK, et al. Resource virtualization switch: US. 7502884 [P]. 2009-10-03.

[44] VirtenSys Inc.. VIO 4000 series I/O virtualization (IOV) switch [EB/OL]. [2015-11-20]. http://searchstorage.techtarget.com/review/VirtenSys-Inc-VIO-4000-Series-I-O-Virtualization-IOV-Switch.