## CPU虚拟化

CPU虚拟化指的是将单个物理CPU虚拟化成多个虚拟CPU供虚拟机使用。虚拟CPU分时复用物理CPU，由虚拟机监控器（Virtual Machine Monitor，VMM，也叫Hypervisor）对虚拟CPU分配时间片，并同时对虚拟CPU的状态进行管理。

CPU虚拟化的三种实现方式：

* 基于全虚拟化的CPU虚拟化
* 基于半虚拟化的CPU虚拟化
* 基于硬件辅助的CPU虚拟化

### 基于全虚拟化的CPU虚拟化

用软件模拟一个物理主机所需要的硬件设备。当虚拟机要执行一行代码时，VMM会将此指令拦截，拦截之后会将此指令发送至BT翻译器（也叫二进制翻译器）。经过翻译的指令这样一来，每个虚拟机的特权指令都需要VMM进行翻译，虚拟机的隔离和封装性得到了保证。

优点：不需要修改Guest OS，指令在虚拟机层次上得到了保证；对VMM和虚拟机之间的内存空间进行保护，实现了虚拟机之间的隔离

缺点：存在频繁的VMM与虚拟机之间的上下文切换开销，性能大打折扣。翻译过程会很耗费物理机的计算资源，效率很低。

图示

描述已自动生成

### 基于半虚拟化的CPU虚拟化

半虚拟化在虚拟机操作系统和虚拟层之间增加了一个特殊指令的过渡模块，通过修改虚拟机操作系统的内核，将执行特殊敏感指令替换为对虚拟化层进行超级调用(Hypercall)的方式来达到此目的。除CPU之外，虚拟化层也对内存操作诸如内存管理、中断处理、时间同步提供了超级调用的接口。

选用基于半虚拟化的CPU虚拟化，能使虚拟机使用CPU如同在物理机上使用CPU一样。对于开源的系统我们可以对其内核指令替换，那么对于Windows操作系统，这样闭源的操作系统，这种方式就吃不开了。

优点：使Guest OS的相关操作都可以在底层硬件上直接实现，获得与本地物理CPU相近的性能。

缺点：但是需要开源操作系统的支持，安全性，可移植性较差。

图示

描述已自动生成

### 基于硬件辅助的CPU虚拟化

当虚拟机要执行一条特权指令时，会引发一个中断，虚拟机会被挂起并且CPU会分配给VMM。接下来会检查引发虚拟机中断的指令，并根据被挂起时所保存的信息，CPU硬件模仿虚拟机的状态并执行相应特权指令，操作完毕后VMM会恢复虚拟机的状态并继续执行。原本是软件进行的操作，通过硬件模拟实现，极大的分担了虚拟化层的压力。

优点：不需要修改客户机的OS，在大部分情况下也不需要进行二进制翻译。大大减少了开销，指令集的变革也使得虚拟机之间的资源共享更加流畅。可以提供非常接近于原生环境的虚拟环境性能

缺点：需要硬件设备支持，成本比较高

图示

描述已自动生成

## 内存虚拟化

## IO虚拟化

## 网络虚拟化卸载加速技术

在传统的应用场景中，服务器资源过剩情况普遍，为了充分利用服务器资源，产生了虚拟化技术。虚拟化技术以牺牲部分效率为代价提升了资源的使用率，将原来需要硬件完成的工作，通过软件模拟的方式，满足多个云租户的需要。随着5G、8K等新型业务的不断涌现，应用系统对性能提出了更高的要求，云服务也出现了满足不同QoS要求的分化，有（物理机、裸金属）等不同产品形态。各个功能模块的衔接配合，各租户之间的信息交互都需要高效的网络系统来实现。

### 网络虚拟化

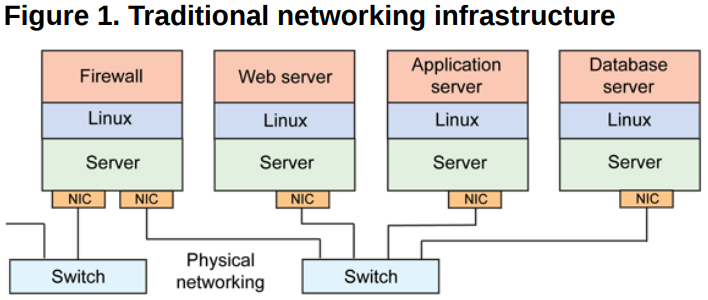
传统数据中心的服务器使用硬件网卡收发包，硬件网卡连接硬件交换机完成跨服务器的通信。硬件场景下，主机发送数据包时：应用程序将数据写入内核分配的内存中，硬件网卡通过DMA将要发的数据从内存搬运到物理网卡的队列中。

主机接收数据包时：硬件网卡收到数据包时按照CPU分配的硬件网卡内存地址，通过DMA功能把数据包写入对应内存中，然后中断CPU进行后续处理。硬件交换机通过MAC进行二层转发。

在云计算场景下，对计算资源进行了切分，[物理服务器](https://cloud.tencent.com/product/cpm?from_column=20065&from=20065)上运行多个虚拟机，虚拟机之间通过虚拟网卡实现互通，虚拟网卡连接到虚拟交换机上,实现同一个服务器内不同虚拟机之间以及不同服务器上虚拟机之间的流量转发。

#### 传统网络架构

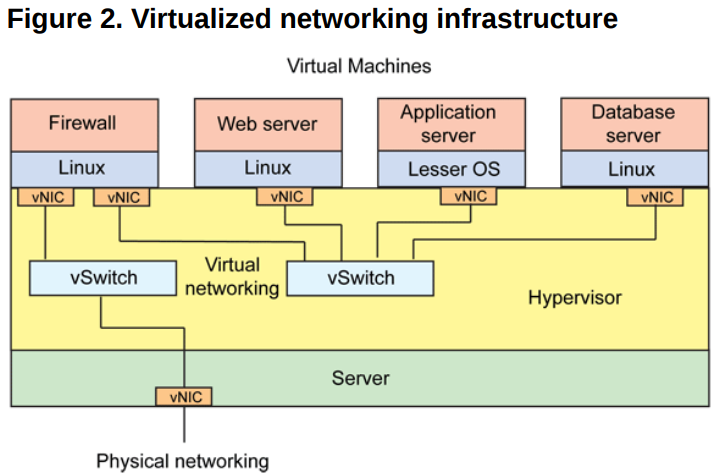
在传统网络环境中，一台物理主机包含一个或多个网卡（NIC），要实现与其他物理主机之间的通信，需要通过自身的 NIC 连接到外部的网络设施，如交换机上，如下图所示。



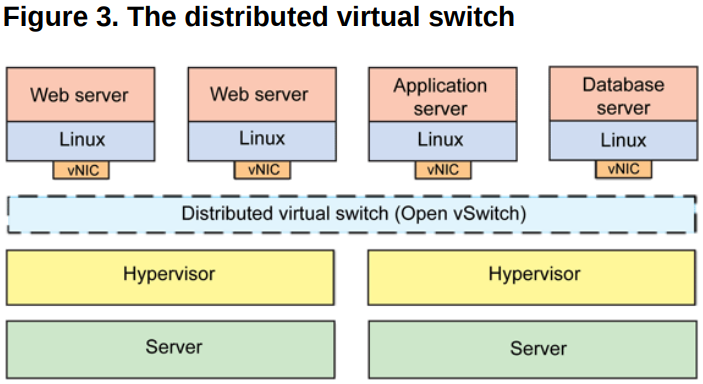
这种架构下，为了对应用进行隔离，往往是将一个应用部署在一台物理设备上，这样会存在两个问题，1）是某些应用大部分情况可能处于空闲状态，2）是当应用增多的时候，只能通过增加物理设备来解决扩展性问题。不管怎么样，这种架构都会对物理资源造成极大的浪费。

#### 虚拟化网络架构

为了解决这个问题，可以借助虚拟化技术对一台物理资源进行抽象，将一张物理网卡虚拟成多张虚拟网卡（vNIC），通过虚拟机来隔离不同的应用。 这样对于上面的问题 1），可以利用虚拟化层 Hypervisor 的调度技术，将资源从空闲的应用上调度到繁忙的应用上，达到资源的合理利用；针对问题 2），可以根据物理设备的资源使用情况进行横向扩容，除非设备资源已经用尽，否则没有必要新增设备。这种架构如下所示。



虚拟机与虚拟机之间的通信，由虚拟交换机完成，虚拟网卡和虚拟交换机之间的链路也是虚拟的链路，整个主机内部构成了一个虚拟的网络，如果虚拟机之间涉及到三层的网络包转发，则又由另外一个角色——虚拟路由器来完成。 一般，这一整套虚拟网络的模块都可以独立出去，由第三方来完成，如其中比较出名的一个解决方案就是 Open vSwitch（OVS）。 OVS 的优势在于它基于 SDN 的设计原则，方便虚拟机集群的控制与管理，另外就是它分布式的特性，可以「透明」地实现跨主机之间的虚拟机通信，如下是跨主机启用 OVS 通信的图示。



总结下来，网络虚拟化主要解决的是虚拟机构成的网络通信问题，完成的是各种网络设备的虚拟化，如网卡、交换设备、路由设备等。

### 网卡虚拟化

虚拟网卡模拟硬件网卡，工作机制类同。虚拟网卡包括e1000，virtio等实现技术。virtio是目前最为通用的技术框架。virtio提供了虚拟机和物理服务器数据交换的通用机制，得到了大多数hypervisor的支持，成为事实上的标准。

virtio是一种半虚拟化的解决方案。半虚拟化方案中，Guest OS知道自己是虚拟机，通过前端驱动和后端模拟设备互相配合实现IO虚拟化。这种方式与全虚拟化相比，可以大幅度提高虚拟机的IO性能。virtio分为四层，包括guest OS中各种前端驱动模块，hypervisor上的后端驱动模块，中间两层包括前后端通信的virtio层和virtio-ring层。其中，virtio层是虚拟队列接口，负责前后端之间的通知机制（kick，notify）和控制流程。virtio-ring实现了两个环形虚拟队列，分别用于保存前端驱动程序和后端处理程序执行的信息。

## VirtIO

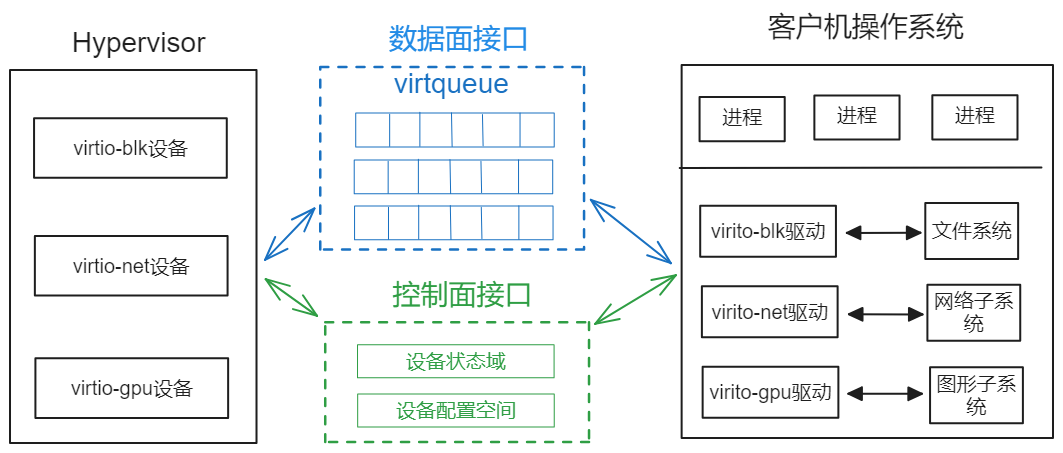
### Virtio简介

VirtIO 由 Rusty Russell 开发，最初是为了支持自己开发的 lguest Hypervisor，其设计目标是在虚拟化环境下提供与物理设备相近的 I/O 功能和性能，并且避免在虚拟机中安装额外的驱动程序。基于这一目标，后来通过开源的方式将 VirtIO 延伸为一种虚拟化设备接口标准，并广泛地支持 KVM、QEMU、Xen 和 VMware 等虚拟化解决方案。

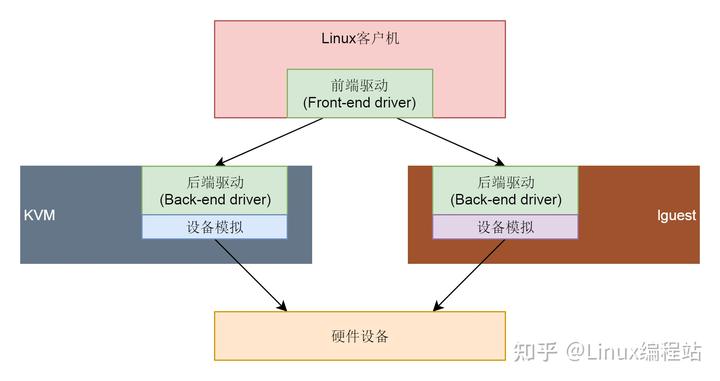
VirtIo是一种I/O虚拟化解决方案，一套通用I/O设备虚拟化的程序，是对半虚拟化Hypervisor中的一组通用I/O设备的抽象。提供了一套上层应用与各Hypervisor虚拟化设备(KVM、Xen、VMware等)之间的通信框架和编程接口，减少跨平台所带来的兼容性问题，大大提高驱动程序开发效率。

图示

中度可信度描述已自动生成



* **高性能**：VirtIO 省去了 Full-virtualizaiton 模式下的 Traps（操作捕获）环节，GuestOS 通过 VirtIO Interfaces 可以直接与 Hypervisor 中的 Device Emulation 进行交互。
* **低开销**：VirtIO 优化了 CPU 在内核态和用户态之间频繁切换，以及 CPU 在 VM Exit 和 VM Entry 之间频繁陷入陷出所带来的性能开销。
* **标准化**：VirtIO 实现了统一的虚拟设备接口标准，可以应用在多种虚拟化解决方案中。



在 VirtIO 提出的虚拟化设备接口标准中，包括多个子系统，每个子系统都定义了一组虚拟设备类型和协议。例如：

* **VirtIO-Block**（块设备）：提供虚拟磁盘设备的接口。
* **VirtIO-Net**（网络设备）：提供虚拟网络设备的接口。
* VirtIO-Serial（串口设备）：提供虚拟串口设备的接口。
* VirtIO-Memory（内存设备）：提供虚拟内存设备的接口。
* VirtIO-Input（输入设备）：提供虚拟输入设备的接口。

此外还有 VirtIO-SCSI、**VirtIO-NVMe**、VirtIO-GPU、VirtIO-FS、VirtIO-VSock 等等虚拟设备类型和协议。

表格

中度可信度描述已自动生成

Virtio的虚拟设备接口标准的架构：

* Front-end：前端驱动
* Back-end：后端驱动，设备模拟程序
* Transport：一组标准的传输层接口，基于环形队列的方式来批量处理I/O请求

图示

描述已自动生成

Virtio-ring是virtio的核心，包括三个部分：

* Desc table：描述符数组，用于存储一些关联的描述符
* Available ring：用于guest端表示当前有哪些描述符是可用的
* Used ring：表示host端哪些描述符已经被使用

对于采用virtio协议进行通信的CPU和外设，其抽象原理如下图所示。CPU与外设可以共同访问内存(例如外设以DMA方式访问内存)；内存中存在一个称为环形队列(IO RING)的数据结构，根据存放对象不同，该队列可分成由IO请求组成的请求队列(Avail Queue)和由IO响应组成的响应队列(Used Queue)。一个IO的处理过程可以分成如下四步：

* 第一步，应用程序下发IO时，CPU将IO请求放入环形结构(IO RING)的请求队列(Avail Queue)中并通知设备；
* 第二步，设备收到通知后从请求队列中取出IO请求并在内部进行实际处理；
* 第三步，设备将IO处理完成后，将结果作为IO响应放入响应队列(Used Queue)并以中断通知CPU；
* 第四步，CPU从响应队列中取出IO处理结果并返回给应用程序。

图示

描述已自动生成

抛开总线协议所决定的通知机制及访存方式，virtio协议定义了明确的队列结构及操作流程。Virtio-blk设备举例：

virtio-blk是一种存储设备，CPU发起的IO请求包含操作类型(读或写)、起始扇区(一个扇区为512节节，是块设备的存储单位)、内存地址、访问长度；请求处理完成后返回的IO响应仅包含结果状态(成功或失败)。如下示例图中，系统产生了一个IO请求(an example of IO)，它在内存上的数据结构分为三个部分：Header，即请求头部，包含操作类型和起始扇区；Data，即数据区，包含地址和长度；Status，即结果状态。

virtio-blk设备使用一个环形队列结构(IO RING)，它由三段连续内存组成：Descriptor Table、Avail Queue和Used Queue：

队列结构：

图示

AI 生成的内容可能不正确。

* Descriptor Table由固定长度(16字节)的Descriptor组成，其个数等于环形队列(IO RING)长度，其中每个Descriptor包含四个域：addr代表某段内存的起始地址，长度为8个字节；len代表某段内存的长度，本身占用4个字节(因此代表的内存段最大为4GB)；flags代表内存段读写属性等，长度为2个字节；next代表下一个内存段对应的Descpriptor在Descriptor Table中的索引，因此通过next字段可以将一个请求对应的多个内存段连接成链表。
* Avail Queue由头部的flags和idx域及entry数组(entry代表数组元素)组成：flags与通知机制相关；idx代表最新放入IO请求的编号，从零开始单调递增，将其对队列长度取余即可得该IO请求在entry数组中的索引；entry数组元素用来存放IO请求占用的首个Descriptor在Descriptor Table中的索引，数组长度等于环形队列长度(不开启event\_idx特性)。
* Used Queue由头部的flags和idx域及entry数组(entry代表数组元素)组成：flags与通知机制相关；idx代表最新放入IO响应的编号，从零开始单调递增，将其对队列长度取余即可得该IO响应在entry数组中的索引；entry数组元素主要用来存放IO响应占用的首个Descriptor在Descriptor Table中的索引(还有一个len域，virtio-blk并不使用)， 数组长度等于环形队列长度(不开启event\_idx特性)。
* 环形队列结构(IO RING)被CPU和设备同见。仅CPU可见变量为free\_head(空闲Descriptor链表头，初始时所有Descriptor通过next指针依次相连形成空闲链表)和last\_used(当前已取的used元素位置)。仅设备可见变量为last\_avail(当前已取的avail元素位置)。

针对示例图中的IO请求，处理流程分析如下：

* 第一步，CPU放请求。由于示例IO请求在内存中由Header、Data和Status三段内存组成，因此要从Descriptor Table中申请三个空闲项，每项指向一段内存，并将三段内存连接成链表。这里假设我们申请到了前三个Descriptor(free\_head更新为3，表示下一个空闲项从索引3开始，因为0、1、2已被占用)，那么会将第一个Descriptor的索引值0填入Aail Queue的第一个entry中，并将idx更新为1，代表放入1个请求；
* 第二步，设备取请求。设备收到通知后，通过比较设备内部的last\_avail(初始为0)和Avail Queue中的idx(当前为1)判断是否有新的请求待处理(如果last\_vail小于Avail Queue中的idx，则有新请求)。如果有，则取出请求(更新last\_avail为1 )并以entry的值为索引从Descriptor Table中找到请求对应的所有Descriptor来获知完整的请求信息。
* 第三步，设备放响应。设备完成IO处理后(包括更新Status内存段内容)，将已完成IO的Descriptor Table索引放入Used Queue对应的entry中，并将idx更新为1,代表放入1个响应；
* 第四步，CPU取响应。CPU收到中断后，通过比较内部的last\_used(初始化0)和Used Queue中的idx(当前为1)判断是否有新的响应(逻辑类似Avail Queue)。如果有，则取出响应(更新last\_used为1)并将Status中断的结果返回应用，最后将完成响应对应的三项Descriptor以链表方式插入到free\_head头部。

### Virtio-mmio

Virtio是一个半虚拟化标准(协议)，协议本身实现的载体和总线并无绑定。virtio协议实现过程中，CPU与外设之间的通知机制以及外设访问内存方式由实际连接CPU与外设的总线协议决定

图示

AI 生成的内容可能不正确。

Virtio协议可以基于多种不同的总线协议来实现。虚拟化场景中，主要采用PCI总线协议和MMIO总线协议：

采用PCI总线协议的virtio设备交virtio-pci设备，它可以支持virtio设备的热插拔特性(基于PCI总线的设备热插拔机制)，并可应用于真是物理外设。

采用MMIO总线协议的virtio设备交virtio-mmio设备，它完全是针对虚拟机涉及的，是一种轻量的虚拟总线机制，支持快速设备发现，但是无法使用在真是物理外设中。

MMIO设备无法像PCI设备一样支持动态发现，guest os需要知道MMIO设备使用的具体内存地址和中断号。这种静态的设备发现机制不像PCI设备的总线枚举机制那么灵活，因此不支持设备热插拔等高级特性。

MMIO设备需要类似PCI设备配置空间一样实现一些寄存器来存放virtio设备的基本信息

MMIO寄存器表单：

表格

AI 生成的内容可能不正确。

### Virtio-pci

### Split模式

Split virt queue的问题在于desc table、used ring、available ring以一种非常稀疏的方式使用内存，导致CPU需要频繁地访问内存，给CPU的cache会造成较大的压力，cache miss的概率比较大，从硬件的角度来看意味着每个descriptor的读写操作都需要几个PC事务

### Packed模式

Packed模式的ring将desc、used、available合成了一个queue来改善split的问题，在descriptor里的flag域增加AVAIL、USED取值，driver和device双方通过本地维护的一个warp bit状态与AVAIL、USED的值综合对比来知晓对方的buffer使用情况

### Vhost-net

Virtio的内核态卸载

图示

描述已自动生成

### Vhost-user

Virtio的DPDK卸载

图示

描述已自动生成

### Virtio-net

VirtIO Networking 是一种高功能、高性能、可扩展的虚拟化网络设备，支持多种网络功能和虚拟网络技术，并可以通过多种实现方式进行部署。

* 网络功能：MAC 地址、流量限制、流量过滤、多队列收发等；
* 虚拟网络技术：VLAN、GRE、VXLAN 等；
* 多种实现方式：Kernel、DPDK、硬件网卡等。

virtio-net 是 VirtIO Networking 的默认实现。其中，Front-end 是 virtio-net Driver（虚拟网卡驱动程序）；Back-end 是由 QEMU 软件模拟的 virtio-net Device。

### Virtio-blk

虚拟化生态中的虚拟化块存储的一种实现方式，利用virtio共享内存机制，提供了一种高效的块存储挂载的方法，guest-OS的内核通过加载virtio-blk驱动，实现块存储的读写。Virtio-blk设备在虚拟机中以一个磁盘的方式呈现，是目前应用最广泛的虚拟存储控制器。

  virtio-blk设备从功能上来看，核心功能就是实现虚拟机内外的事件通知和数据传递：虚拟机内部的前端驱动准备好待处理的IO请求和数据存放空间并通知后端；虚拟机外部的后端程序获取待处理的请求并交给真正的IO子系统处理，完成后将处理结果通知前端。实际上，除了虚拟磁盘，虚拟网卡也完全可以复用这套机制，从而实现半模拟的网络前后端(virtio-net)。如果将virtio-blk或virtio-net设计成不同类型的PCI设备，那么前端驱动中会存在大量关于事件通知和数据传递的重复代码。

### Virtio的DPU卸载

### SR-IOV

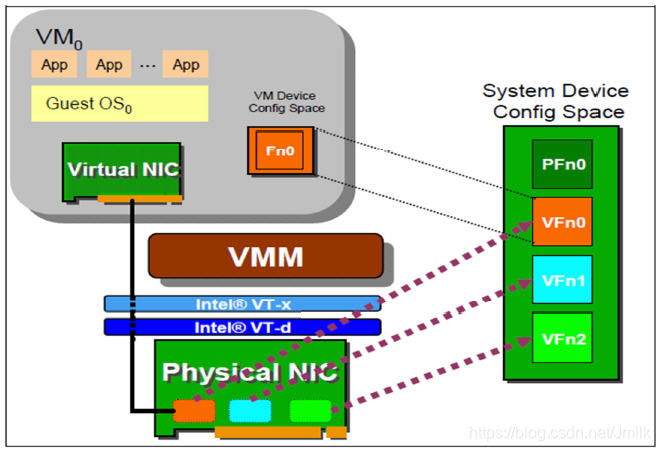
Single-Root I/O Virtualization，单根I/O虚拟化，PCI-SIG推出的一项标准，定义了一种PCIe设备虚拟化技术的标准机制，是“虚拟通道”的一种技术实现，用于将一个PCIe设备虚拟成多个PCIe设备，每个虚拟PCIe设备都具有自己的PCIe配置空间，可以像物理PCIe设备一样为上层软件提供服务。

SR-IOV 技术是一种基于物理硬件的虚拟化解决方案，pass-through方案，将物理设备(通常是网卡的VF)直接分配给虚拟机，其优点在于数据平面是在虚拟机和硬件之间直通的。可以提高物理 I/O 设备（常见的是网络适配器）的性能与可扩展性。SR-IOV 技术允许在虚拟机之间高效共享 PCIe 设备，由于 SR-IOV 技术是基于硬件实现的，可以使虚拟机获得与宿主机媲美的 I/O 性能。

SR-IOV 虚拟出来的通道分为两个类型：

* **PF（Physical Function，物理功能）**：管理 PCIe 设备在物理层面的通道功能，可以看作是一个完整的 PCIe 设备，包含了 SR-IOV 的功能结构，具有管理、配置 VF 的功能。
* **VF（Virtual Function，虚拟功能）**：是 PCIe 设备在虚拟层面的通道功能，即仅仅包含了 I/O 功能，VF 之间共享物理资源。VF 是一种裁剪版的 PCIe 设备，仅允许配置其自身的资源，虚拟机无法通过 VF 对 SR-IOV 网卡进行管理。所有的 VF 都是通过 PF 衍生而来，有些型号的 SR-IOV 网卡最多可以生成 256 个 VF。

简而言之，每个 VF 就像是物理网卡硬件资源的一个切片，而 PF 则是对所有物理网卡硬件资源的统筹者，包括管理众多 VF 可以协同工作。



目前性能最好的 Data Plane 无疑是 SR-IOV VF Passthrough，但原生的 SR-IOV 缺少控制面逻辑，在实际使用中多有不便。

### vDPA

virtio的半硬件卸载，virtio设备虚拟化的功能一直都是由软件来实现，在Host上因虚拟化而产生的额外开销无法避免。为了提升云端服务的性能，可以将virtio功能offload，将于业务无关的任务绕开系统、CPU交给专用硬件来执行。

vDPA(vhost Data Path Acceleration)，是一种基于virtio标准的硬件加速技术，通过将控制面软件模拟和数据面硬件实现相结合，提高了虚拟机的性能。

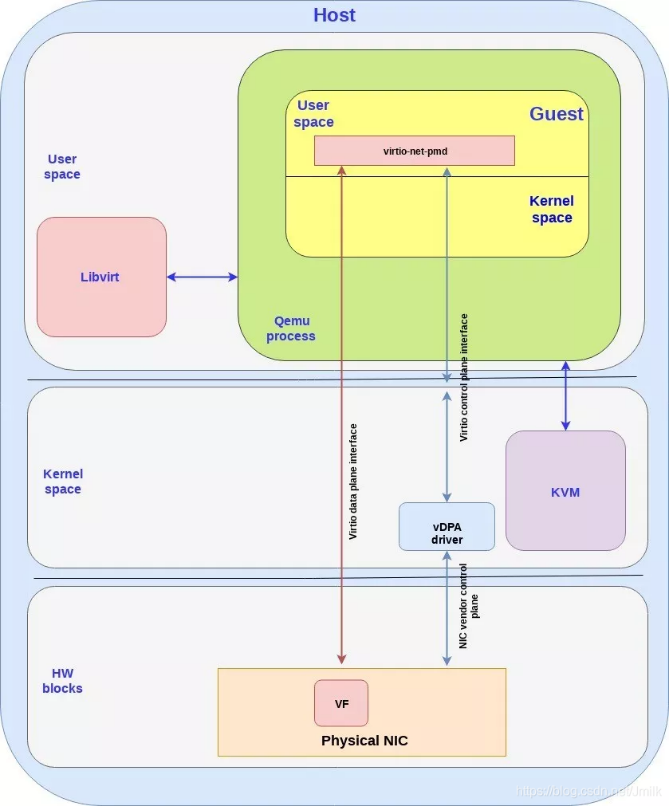
vDPA是virtio的半硬件虚拟化实现，该框架有Redhat提出，实现了virtio数据平面的硬件卸载。控制平面仍然采用原来的控制平面协议，当控制信息被传递到硬件中，硬件完成数据平面的配置之后，数据通信过程由硬件设备(智能网卡)完成，虚拟机和网卡之间直通。中断信息也由网卡直接发送至虚拟机不需要主机的干预。这种方式控制面比较复杂，硬件难以实现。

vDPA的工作原理是将虚拟机的设备驱动程序分为控制面和数据面两部分。控制面负责虚拟机的配置和管理，数据面负责数据的传输和处理。控制面采用软件模拟的方式实现，数据面则采用硬件实现。这样，控制面的复杂度降低，数据面的处理效率提高，从而提高了虚拟机的性能。

vDPA的应用场景主要是针对需要大量IO操作的虚拟机，如[数据库](https://cloud.baidu.com/solution/database.html" \t "_blank)、Web服务器等。在这些场景下，VDPA能够显著提高虚拟机的性能，减少虚拟机间的竞争和冲突。

虽然vDPA提高了虚拟机的性能，但也存在一些缺点。首先，vDPA的实现需要针对不同的硬件平台进行优化和适配，增加了开发的难度和成本。其次，vDPA对虚拟机的隔离性有一定影响，因为虚拟机间的[数据传输](https://cloud.baidu.com/product/dts.html" \t "_blank)和处理可能会存在一定的耦合性。最后，vDPA对虚拟机的可移植性也有一定影响，因为不同的硬件平台可能支持不同的vDPA技术，导致虚拟机在不同的平台上运行时性能存在差异。

HW vDPA（Hardware vhost Data Path Acceleration，硬件 vhost 数据面加速）就是一种使用 SR-IOV VF 来充当 Data Plane、使用 vDPA Framework 来充当控制面的软硬件融合加速技术，能够同时兼具软件的可管理性、灵活性以及硬件的高性能。同时也解决了 VF Passthrough 虚拟机不支持热迁移的难题。



基于vDPA架构的virtio-blk硬件卸载

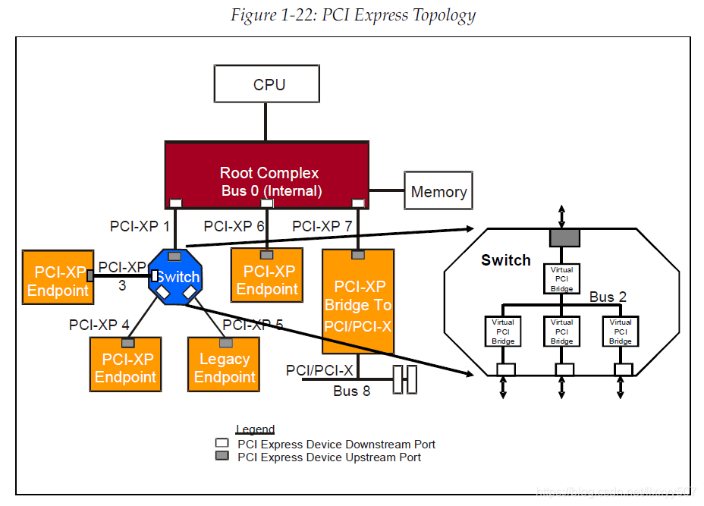
图形用户界面

描述已自动生成

## PCIe

### PCI和PCIe设备

* **PCI（Peripheral Component Interconnect，外设组件互连）**：符合 PCI 总线标准的设备就被称为 PCI 设备，PCI 总线架构中可以包含多个 PCI 设备。
* **PCIe（Peripheral Component Interconnect Express，快速外设组件互连）**：PCI Express，简称 PCIe，是电脑总线 PCI 的一种，它沿用了现有的 PCI 编程概念及通讯标准，但建基于更快的串行通信系统。是 Intel 提出的新一代的总线接口，PCI Express 采用了目前业内流行的点对点串行连接，比起 PCI 以及更早期的计算机总线的共享并行架构每个 PCIe 设备都有自己的专用连接，不需要向整个总线请求带宽，而且可以把数据传输率提高到一个很高的频率，达到 PCI 所不能提供的高带宽。



EP模式

RC模式

### SR-IOV