A Comprehensive Implementation and Evaluation of Direct Interrupt Delivery

摘要

随着与CPU和内存虚拟化相关的性能开销在很大程度上可以忽略不计，研究工作致力于减少I / O虚拟化开销，这主要来自两方面：DMA设置和有效负载复制以及中断传递。 SRIOV和MRIOV的出现将DMA相关的虚拟化开销最小化。 因此，最小化虚拟化开销的最后战场是如何在不涉及虚拟机hypervisor的情况下直接将每个中断传递到其目标VM。

本文介绍了一种称为DID的基于KVM的直接中断传递系统的设计，实现和评估。 DID将中断从SRIOV设备，虚拟设备和计时器直接传递到其目标VM，从而完全避免了VM退出。此外，DID不需要修改VM的操作系统，并且在所有情况下都可以在中断之间保留正确的优先级。 我们证明了DID将I / O密集型工作负载的VM Exit数量减少了100倍，将中断调用延迟减少了80％，并将运行Memcached的VM的吞吐量提高了3倍。

1.简介

随着虚拟化对硬件的支持越来越多且更加复杂，与CPU和内存虚拟化相关的性能开销在很大程度上可以忽略不计。剩下的唯一重要的虚拟化开销来自于I / O虚拟化。 I / O虚拟化开销本身主要来自两个方面：设置DMA操作和复制DMA有效负载，以及在I / O操作完成后传送中断。 SRIOV [12]和MRIOV [32]的出现允许VM直接与I / O设备进行交互，从而有效地将与DMA相关的虚拟化开销降至最低[25，27]。因此，最后的I / O虚拟化性能障碍是由于中断传递。 因为中断传递的主要开销是VM Exit，所以减少虚拟化服务器I / O开销的一种关键方法是将发往给VM的中断直接传递到该VM，绕过VM Exit并避免使用hypervisor。直接将中断传递给对应的VM不仅减少了与I/O虚拟化有关的虚拟化的性能开销，还减少了**中断调用延迟**——一个在实时虚拟化计算系统中应重点考虑的点。本论文描述了一个基于KVM的叫做DID的中断直接传递系统的设计、实现和评估，提供了一个综合性解决方法来减少虚拟服务器的中断传递性能开销。

DID解决了直接中断传递的两个关键技术挑战。第一个挑战是如何在传递过程中避免调用hypservisor而直接传递中断给目标VMs。第二个挑战是如何将中断成功完成的信号传给中断控制器硬件而不陷入到hypervisor。在项目开始时设定了以下目标：

1.如果某个虚拟机正在运行，则该虚拟机的所有中断（包括来自仿真设备，SRIOV设备，计时器和其他处理器的中断）都将直接传递。

2.当目标虚拟机未运行时，必须通过虚拟机hypervisor间接传递其关联的中断，但是正确保留所有中断之间的优先级，不论是直接或间接传递的。

3.传递和完成中断所需的VM退出数量为零。

4.无需对虚拟机的操作系统进行半虚拟化或修改。

为了满足这些目标，DID利用了现代Intel x86服务器上可用的几种架构功能。首先，DID利用IOMMU上的中断重映射表，在目标VM运行时将中断直接路由到其目标VM，而在目标VM未运行时将其路由到hypervisor，而无需对guest OS进行任何更改。此外，系统hypervisor可以在任何可用的CPU内核上运行。如果通过hypervisor传递了中断，则在将其传递给目标VM时，它将变为虚拟中断。其次，DID利用**处理器间中断（IPI）机制**将来自hypervisor的虚拟中断直接注入另一个内核上运行的VM。这种虚拟中断传递机制可以有效地将虚拟中断转换回物理中断，从而消除了现有虚拟中断传递机制的一个众所周知的问题，即较低优先级的虚拟中断可能会覆盖较高优先级的直接传递的中断，**因为虚拟中断不会通过本地中断控制器（本地APIC）**。此外，DID对计时器中断进行特殊处理，这种处理不会通过IOMMU，并且在传递计时器中断时，无论将其传递到VM还是hypervisor，都避免了VM Exit。这是通过仔细安装hypervisor在专用内核上设置的计时器中断以及在VM挂起和迁移时VM所设置的计时器中断的迁移来实现的。

解决此中断传递问题的现有方法包括：在客户机OS和主机内核中使用软件补丁程序，以在中断到达时启用hypervisor绕过[18、19、31]或特定于供应商的硬件升级，例如英特尔的中断APIC虚拟化支持（APICv）[1]。 DID采用仅软件的方法，并且与现有的硬件解决方案相比，DID被证明在减少VM退出数量方面更为有效。在仔细研究了现有的基于软件的解决方案后[18，19，31]，我们发现了这些解决方案中的几个主要限制，并在DID中删除了所有这些限制。具体而言，现有解决方案区分分配的中断和未分配的中断，并且能够直接传递通常来自SRIOV设备的分配的中断。此外，这些解决方案具有级联效应——在向VM注入虚拟中断时，和在创建更多的虚拟中断的过程中，管理程序必须关闭VM的直接中断机制。最后，传统或闭源OS不能享受这些解决方案的优势，因为它们需要修改客户机OS。

当前的DID原型内置于KVM管理程序[24]中，它支持SRIOV设备的直接直通。 对于配备SRIOV NIC的虚拟化x86服务器，我们演示了DID的以下性能优势：
•循环测试基准的中断调用延迟减少了80％，从14µs降至2.9µs。
•基于TCP的计算机内iperf吞吐量提高了21％。
•由于将VM退出速率从每秒97K降低到每秒小于1K，因此，按照每秒有限延迟请求的速度，Memcached吞吐量提高了330％。

2.背景

在Intel x86服务器上，系统软件执行以下步骤来与I / O设备（例如NIC）进行事务。 首先，系统软件发出I / O指令以设置DMA操作，以将数据从内存复制到I / O设备。 然后，I / O设备上的DMA引擎通过向CPU发送中断来执行实际的复制并发出完成信号。 最后，系统软件中的相应中断处理程序将被调用以处理完成中断，并向中断控制器硬件发送确认。在原本的I / O虚拟化实现中，至少需要三个VM Exit才能执行I / O事务：一个Exit在发出I / O指令时，另一个Exit在完成中断传递时，第三个Exit在中断处理程序完成时 。 如果I / O设备支持**单根I / O虚拟化（single root I/O virtualization, SRIOV）**，则VM能够以与其他VM隔离的方式直接向该设备发布I / O指令，因此避免了发出I / O指令时的VMExit。 但是，尽管有SRIOV，其他两个VM Exit仍然存在。 DID的目标是通过直接向其VM发送完成中断并允许VM直接确认中断来消除与每个I / O事务相关的两个VM退出，这两种情况均不涉及hypervisor。

2.1 Intel x86's 中断体系结构

在x86服务器上，中断是由I / O设备等外部组件生成的异步事件。 当前正在执行的代码被中断，并且控制跳至预定义的处理程序，该处理程序在被称为**IDT（interrupt description table,中断描述符表）**的内存表中指定。 x86体系结构最多定义256个中断向量，每个中断向量对应一个中断处理程序函数的地址，该地址将在触发相应的中断时被调用。

以前，I / O设备通过在将自身连接到CPU的**可编程中断控制器（programmable interrupt controller, PIC）**的电线上发送信号来中断CPU。但是，现代x86服务器采用了更灵活的中断管理架构，称为**消息信号中断（message signal interrupt, MSI）及其扩展MSI-X。** I / O设备通过对特殊地址执行内存写操作来向CPU发出一条消息，告知中断消息，这会导致将物理中断发送给CPU。服务器启动时，系统软件负责为服务器中检测到的每个I / O设备分配MSI地址和MSI数据。 MSI地址是从分配给**本地APIC（local advanced programmable interrupt controller, LAPIC）**的地址范围中分配的，MSI数据是在存储写操作中使用的有效载荷，可触发一个发出中断信号的消息。中断的MSI地址指定了中断的目标CPU内核的ID，其MSI数据包含中断的向量号和传送guest mode。

MSI与**PCIe**兼容，PCIe是Intel x86服务器上使用的主要I / O互连体系结构。用于触发MSI中断的每个内存写操作都是一个PCIe内存写请求，该请求由PCIe设备发出，并将PCIe层次结构遍历到根联合体[7]。 x86服务器为每个CPU内核使用一个LAPIC，为每个I / O子系统使用一个IOAPIC，以及一个**IOMMU**，来将PCIe地址空间与服务器的物理内存空间隔离开。 IOAPIC支持I / O重定向表，而IOMMU支持中断重映射表。这两个表均允许系统软件为每个PCIe设备中断指定目标ID，触发guest mode和传送guest mode。中断的触发guest mode指定中断向CPU的信号是边沿触发还是电平触发。中断的可能传递guest mode是（1）固定guest mode，其中将中断传递给目标ID字段中指示的所有CPU，（2）最低优先级guest mode，其中将中断仅传递给以最低优先级执行的目标CPU （3）**NMI（Non-Maskable Interrupt不可屏蔽中断）**guest mode，其中中断以最高优先级传递到目标CPU内核，并且不能被屏蔽。

IOMMU是现代x86服务器[6，20]中内置的I / O虚拟化技术的重要组成部分，可确保仅允许来自授权PCIe设备的授权中断进入系统。 每个**中断重新映射表（interrupt remapping table,IRT）**条目都指定与MSI地址相关联的中断信息，包括称为SID的源ID字段。 当IOMMU的中断重新映射机制打开时，MSI地址中的字段用于引用IRT中的条目。 未经授权的MSI中断指向无效的IRT条目或SID不匹配的有效IRT条目，因此被IOMMU阻止[34]。

当MSI中断到达其目标CPU时，将调用IDT中的相应中断处理程序。 具体来说，x86 CPU维护两个256位位图：**中断请求寄存器（interrupt request register,IRR）**和**服务中寄存器（in-service register,ISR）**。 带有向量v的中断X的到来设置IRR的第v位（即IRR [v] = 1）。 一旦调用X的中断处理程序，就会清除IRR [v]并将ISR [v]设置为指示X当前正在被服务。 与X关联的中断处理程序完成后，它将写入相应LAPIC的**中断结束（end-of-interrupt,EOI）寄存器**，以向硬件确认中断X。 通常，对EOI的写入不包含向量信息，因为它隐式假定当前最高中断已完成。 中断控制器依次清除ISR中的相应位，并在当前未决的中断（如果有）中传递最高优先级的中断。

最后，x86 CPU内核可以通过称为**处理器间中断（inter-processor interrupt,IPI）**的特殊类型的中断将中断发送到另一个CPU内核。 IPI的应用包括启动，唤醒或关闭另一个CPU内核以实现更节能的资源管理，以及刷新另一个CPU内核的**TLB**(translation lookaside bufffer, 地址转换后援缓存，简称“快表‘)以保持TLB一致性。 当CPU内核发送IPI时，它会将包含IPI参数（例如，传递guest mode，触发guest mode，中断向量，目标ID，优先级等）的有效负载写入其LAPIC的中断命令寄存器（ICR）。 CPU内核能够将IPI发送到其自己的目标ID，从而触发自身IPI，即发送内核上的中断。

2.2虚拟中断

当管理程序在x86 CPU内核上运行时，x86 CPU内核处于host mode；在虚拟机上运行x86 CPU内核时，其处于guest mode。 CPU内核将保持guest mode，直到配置为强制转换为host mode的任何事件为止。 过渡到host mode时，hypervisor将接管，处理触发事件，然后重新进入guest mode以恢复VM的执行。 从guest mode到host mode的过渡称为VM退出，从host mode到guest mode的过渡称为VM进入。 VM退出/进入的性能开销在于执行管理程序代码时保存和还原执行上下文所花费的周期以及与CPU缓存相关的污染。

x86体系结构中的VT支持[33]使hypervisor可以在**VMCS（Virtual Machine Control Structure,虚拟机控制结构）**中设置一个控制位，称为外部中断退出（EIE）位，该位指定一个硬件中断事件是否触发VM退出。 更具体地说，如果清除了EIE位，则在运行VM的情况下到达CPU内核的中断会导致直接调用VM中的中断处理程序地址，而不会导致VM退出。 设置EIE后，该中断将强制VM退出并由管理程序进行处理。 x86体系结构的VT支持还支持VMCS中的另一个控制位，称为**NMI**(不可屏蔽中断)退出位，它指定当NMI中断传递到运行VM的CPU内核时，是否触发VM退出；或者也直接传送到VM中。

当将中断直接传递给VM时，CPU内核使用与host mode下使用的IDT不同的中断描述符表（IDT）。 **另一方面，当发往虚拟机的中断触发VM Exit并由hypervisor传递时，hypervisor负责将此中断转换为虚拟中断，并在虚拟机恢复执行时将其注入目标虚拟机。** 请注意，VM退出并不总是导致虚拟中断注入。 例如，如果VM退出是由目标不是正在运行的VM 的中断引起的（例如，由管理程序设置的计时器中断），则此中断不会转换为虚拟中断，因此不会执行虚拟中断注入。

KVM通过使用内存中的数据结构模拟LAPIC寄存器，将虚拟中断注入VM，通过在恢复VM之前设置模拟寄存器（例如IRR和ISR）来模拟硬件LAPIC。 恢复虚拟机后，它会检查IRR，并通过查找虚拟机的IDT并调用相应的中断处理程序来处理优先级最高的挂起中断。 中断处理程序完成后，它将通过写入（模拟的）EOI寄存器来确认虚拟中断，这将触发另一个VM退出至管理程序，以更新软件模拟的IRR和ISR寄存器。 这种设计有两个缺点。 首先，虚拟中断可能会以更高的优先级覆盖直接中断的服务。 其次，每个EOI写入除了引起原来触发的那个中断传递，还会导致VM退出。

2.3虚拟设备

如果VM是SRIOV设备，则它直接与I / O设备进行交互；如果它是虚拟设备，则通过管理程序间接与I / O设备进行交互。对于服务器上部署的SRIOV设备，服务器上的每个VM均分配有SRIOV设备的虚拟功能。当SRIOV设备上的虚拟功能发出中断时，管理程序将处理该中断，然后将相应的虚拟中断注入目标VM。现代管理程序将虚拟设备驱动程序分为驻留在客户机中的前端驱动程序和驻留在管理程序中的后端驱动程序。当VM与虚拟设备执行I / O事务时，hypervisor会在虚拟设备的后端驱动程序处完成事务，并通过IPI向请求的VM注入完成中断，因为VM及其后端驱动程序通常在不同的CPU内核。异步地，管理程序与相应的物理设备执行请求的事务，并以正常方式处理来自物理设备的完成中断。

来自SRIOV设备和虚拟设备的完成中断由hypervisor处理，并作为虚拟中断进行转换并传递到其目标VM。 此外，当前用于处理虚拟中断的EOI写入的机制需要管理程序的参与。 结果，来自I / O设备的每个完成中断都需要至少两个VM退出。

2.4APIC虚拟化

由于退出虚拟机的主要原因之一是由于hypervisor保持了虚拟机仿真的LAPIC的状态，因此最近发布的英特尔CPU功能APICv旨在通过在处理器中虚拟化LAPIC来解决该问题。 通常，APICv虚拟化VMCS中与中断相关的状态和APIC寄存器。 APICv模拟APIC访问，以便APIC读取请求不再导致退出，并且APIC写入请求从fault-like的VM退出转换为trap-like VM退出，这意味着指令在VM退出之前完成，并且处理器状态将根据指令被更新 。APICv通过它的**posted interrupt**机制优化了虚拟中断传递处理，这一机制允许hypervisor在guest mode中通过program VMCS中与posted interrupt相关的数据结构将虚拟中断注入。通常，传递虚拟中断需要VM退出到host mode，因为由VMCS维护的数据结构不允许在guest mode中修改。然而，有了APICv，就没有了这一限制，而且在VM运行时，hypervisor就可以更新VM的中断状态寄存器，如IRR (Interrupt Request Register，中断请求寄存器)和ISR (In-Service Register，服务中寄存器)。

特别地，APICv能够通过增加两个寄存器作为客户机中断状态——RVI(Requesting Virtual Interrupt，请求虚拟中断)和SVI(Servicing Virtual Interrupt，服务虚拟中断)，来在VM不退出的情况下传递虚拟中断，并且允许在guest mode对它们更新。APICv的虚拟中断，或者posted 中断，是通过设置256位的PIR(Posted Interrupt Requestr，posted中断请求)和ON(Outstanding Notification，重要通知？)来传递的。PIR寄存器显示了posted-interrupt的数量，而ON位则表明了一个posted-interrupr被悬挂中。

posted interrupt在guest mode被传递给正在运行的VM而RVI和SVI的相应状态，在未牵涉hypervisor的情况下就被改变了。在处理posted interrupt的后期，APICv的EOI（end-of-interrupt）虚拟化维护了一个256-bit的EOI-Exit位图，允许Hypervisor启用相应的posted interrupt的矢量号的trap-less EOI写入。最后，posted interrupts可以在中断重映射表中被配置，所以不仅虚拟中断，还有外部的中断都可以被直接注入到客户机中。

3.相关工作

中断和LAPIC已经被识别出是I/O虚拟化开销的主要原因，特别是对于I/O密集型的任务。为了减少VM退出的次数，硬件厂商们正在努力寻求对APIC的硬件虚拟化支持，例如英特尔的APICv，ARM的VGIC。这些技术或许可以作为将来硬件的一个选择，而如今，对于减少VM退出的开销，DID则可以在不需要高级的特定硬件厂商的硬件支持下，达到相同或者更好的目标。Ole Agesen等 [14]提出了一种二进制重写技术来减少VM退出的次数。 该机制通过识别随后会导致VM退出的指令对，并将客户机代码动态转换为会减少VM退出的变体来动态优化VM代码。Jailhouse [4]是一个分区hypervisor，它可以预分配硬件资源并将其专用于客户机系统，以实现裸机性能。 但是，由于所有类型的物理资源都缺乏硬件虚拟化，因此此方法通常需要大量的客户机修改，这就失去了虚拟化的好处。另一方面，NoHype [22, 29] 从安全方面解决了VM退出，因为VM退出发生在控制权从客户机转换到主机hypervisor的时刻。不同于NoHype，DID是为了性能而非安全。

ELI和ELIVS是针对实现直接传递中断的最著名的解决方案。然而ELI只直接传递了SRIOV设备的中断给VM，DID则通过直接传递所有的中断，如计时器的，虚拟化的和半虚拟化的设备中，来断改善了ELI。ELI通过使用一个**shadow IDT**——修改VM的IDT使得所有分配给hypervisor和其他VM的中断向量都无效，（causing the corresponding interrupts to always force the a VM exit. ）解决了mis-delivery问题。在半虚拟化的I/O设备中段的情况下，DID比ELVIS更普适，因为它不需要对客户机OS进行修改,对于使用了闭源的操作系统和二进制操作系统发行版本的VM而言，这是个很大的部署优势。最后，DID利用IPI机制将虚拟中断注入目标VM，从而迫使HW LAPIC以与直接传递中断相同的方式管理虚拟中断。这统一了虚拟和直接中断的传递机制，避免了优先级倒置。

此外，ELI / ELVIS提出的直接中断传递机制仅在其非注入guest mode下才有效，如图1所示。具体来说，ELI将中断源分为可直接传递的**assigned interrupts**和退回到KVM的虚拟中断的**non-assigned interrupts**。随着未分配的中断在t1时刻到达，并且直到t4时刻完成，ELI的直接中断机制完全关闭。即使已分配的中断到达了注入guest mode（t6，t7），ELI / ELVIS也必须将其转换为未分配的中断，从而使直接中断机制部分定向，并且系统在处理传统中断注入时会停留更长的时间。 我们总结了表1中的现有方法，并在下一部分中介绍了DID的设计。

	Virtual Interrupt	External Device Interrupt	Timer Interrupt	End-Of-Interrupt	Guest Modification
ELI/ELVIS	Mixed HW/emulated LAPIC	Partially Direct	Indirect	Partially Direct	No/Yes
Jailhouse	Not Support	Direct	Direct	Direct	Yes
APICv	Posted Interrupt	Indirect	Indirect	Direct	No
DID	HW LAPIC	Direct	Direct	Direct	No

Table 1. *Comparison of the interrupt delivering mechanisms between ELI/ELVIS, Jailhouse, APICv, and DID.*

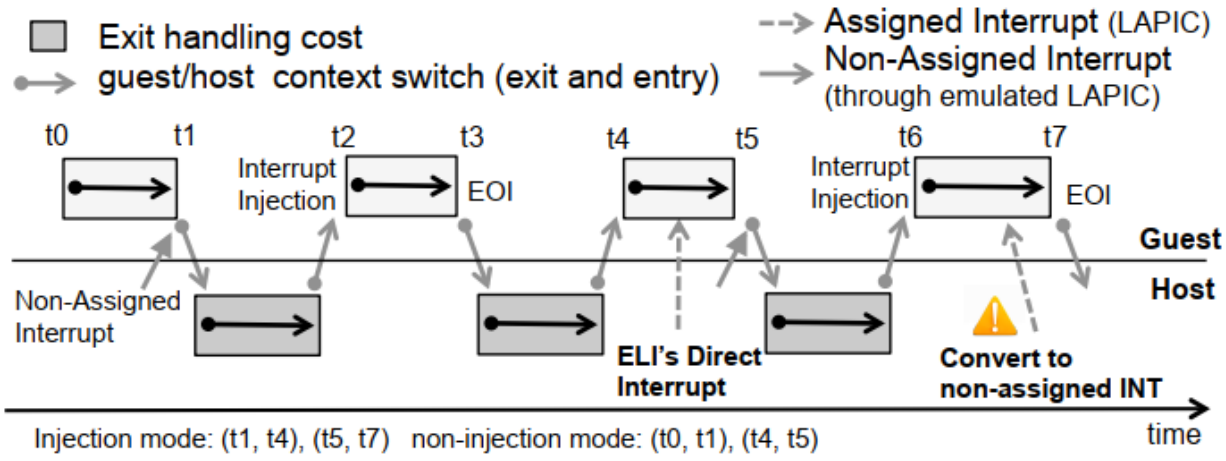

Figure 1. *ELI's mechanism takes effects only at its non-injection mode period, which are between (t0, t1) and (t4, t5), while DID direct delivers all interrupts as long as the CPU is in guest mode.*

4.提出的直接中断传递机制

4.1综述

在x86服务器上支持直接中断传递的主要挑战是避免误传递问题，即，将中断传递给非目标VM的问题。传递错误的问题主要是由以下架构限制引起的。首先，x86服务器体系结构规定，要么每个外部中断都会导致VM退出，要么所有外部中断都不会导致VM退出。此限制使得难以根据其目标VM当前是否正在运行来不同地传递中断。一种可能的解决方案是影子IDT [18，19]。但是，它存在几个安全问题。其次，仅当虚拟机监控程序和VM都在同一CPU内核上运行时，虚拟机监控程序才能够向虚拟机注入虚拟中断。对于虚拟设备，这会导致VM对于后端驱动程序到与其关联的前端驱动程序的每个中断，都会退出，因为这些驱动程序倾向于在不同的CPU内核上运行。第三，LAPIC定时器中断不会通过IOMMU，因此无法从中断重映射表中受益。结果，用于计时器中断传递的现有机制触发了VM退出到管理程序。随着越来越多的应用程序中使用高分辨率计时器，这会导致显着的性能开销。此外，在定时器中断上触发VM退出会增加中断调用等待时间的差异，因为在中断传递中涉及了附加的软件层（即，管理程序）。

DID利用x2APIC [11]提供的灵活性，在对计时器进行编程并发出中断信号时，消除了不必要的VM退出。 借助x2APIC，hypervisor可以指定VM可以直接读取或写入LAPIC区域中的哪些寄存器，而无需触发VM退出。 具体来说，DID向VM公开了两个特定于模型的寄存器，即x2APIC EOI寄存器和TMICT（Initial Timer Count，初始计时器计数）寄存器。 结果，VM可以对LAPIC定时器进行编程并直接写入关联的EOI寄存器，而不会导致VM退出和与此相关的性能开销。

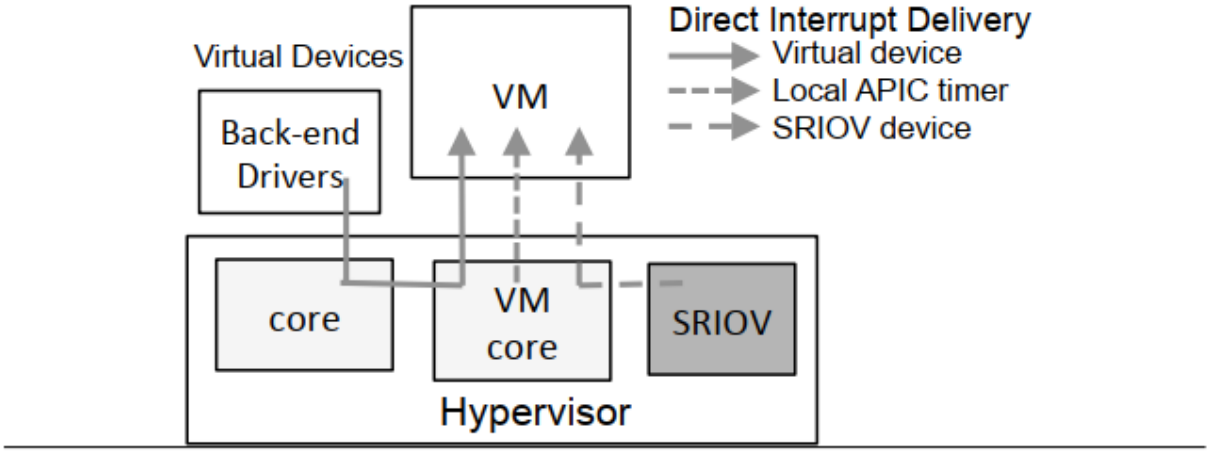

Figure 2. *DID delivers interrupts from SRIOV devices, virtual devices, and timers directly to the target VM.*

在下面的小节中，我们将详细介绍DID如何将中断从SRIOV设备，虚拟设备和计时器直接传递到它们的目标，以及如何支持直接EOI写入，同时保持中断之间的优先级，而不管它们如何传递。

4.2 SRIOV设备中断

当在具有SRIOV设备（例如NIC）的服务器上启动虚拟机 M时，会在SRIOV设备上为其提供虚拟功能F。 一旦建立了M和F之间的绑定，M可以直接向F发出内存映射的I / O指令，而F只能中断M。 在DID中，当F产生中断时，如果M正在运行，则此中断将通过PCIe层次结构（即IOMMU），并最终到达M在guest mode下运行的CPU内核的LAPIC。 否则，DID安排将中断传递给hypervisor，然后hypervisor将虚拟中断注入M。

为了实现上述行为，对于每个VMCS，我们都将EIE位清零，以便向正在运行的VM发送中断不会导致VM退出。我们还设置了NMI退出位，这样即使清除EIE位，NMI中断也会强制VM退出。当我们的DIDhypervisor调度VM M在CPU内核C上运行时，它将修改分配给M的虚拟功能的IOMMU的中断重映射表条目，以使这些虚拟函数生成的中断的目的地为C。这确保了每个当M运行时，M的SRIOV设备中断直接路由到分配给M的CPU内核。此外，当DIDhypervisor对VM M进行调度时，它会修改分配给M的虚拟功能的IOMMU的中断重映射表条目，以便将这些虚拟功能所生成的中断的传递guest mode更改为NMI guest mode。这样可以确保当M不在运行时，M的每个SRIOV设备中断都会导致VM退出，并且作为NMI中断传递给hypervisor。 系统管理程序schedules和deschedules VM时，对中断重新映射表的其他修改仅在中断的目标VM正在运行时才可以直接传递SRIOV设备中断。

当通过DIDhypervisor间接传递SRIOV设备中断时，hypervisor将在最初运行中断目标VM的CPU内核上运行，而不是在专用CPU内核上运行。 这使得我们间接传递的中断的处理开销可以均匀地分布在所有CPU内核上。

在我们的设计中，即使VM M在CPU内核C上运行，当直接传递的SRIOV设备中断到达C时，C实际上也可能处于host mode（即，hypervisor正在运行，而不是在运行 M）。 在这种情况下，DID hypervisor将接收到的中断转换为虚拟中断，并在恢复执行M时将其注入M。

4.3 虚拟设备中断

为了利用物理I / O设备操作和VM执行之间的并行性，现代管理程序（例如KVM）将线程专用于与VM关联的每个虚拟设备。通常，VM的虚拟设备线程在与运行VM的CPU内核不同的CPU内核上运行。在DID系统上，当虚拟设备线程将虚拟设备中断I传递到与其关联的VM M时，虚拟设备线程首先检查M当前是否正在运行，如果是，则向运行M的CPU内核发出IPI，将内核的IPI的中断向量设置为中断I的中断向量。因为我们清除了EIE位，所以此IPI被传递给M而不导致VM退出。最终结果是虚拟设备中断直接传递到与其关联的VM中，而没有VM退出。

尽管DIDhypervisor仅在虚拟机正在运行时才尝试将虚拟设备中断传递给与其关联的虚拟机，但仍存在竞争情况。基于IPI的虚拟设备中断可以只在与之相关的VM运行时被传递给CPU内核，但是在传递中断时，CPU内核有可能处于host mode而不是guest mode。在这种情况下，系统管理程序代表关联的VM接受IPI，将基于IPI的虚拟设备中断转换为虚拟中断，然后在恢复客户机的执行前将其注入到关联的VM中。

4.4 计时器中断

对于SRIOV设备中断的直接传递，我们在DID中利用硬件对中断重映射提供的支持的灵活性来解决误传问题；对于虚拟设备中断的直接传递，DID通过确保在发送IPI（处理器间中断）到对应的内核之前，虚拟机正在目标CPU内核上运行来解决误传问题。然而，在x86服务器上，计时器中断与LAPIC相关联，并且在到达目标CPU内核之前不会通过中断重映射表。 结果，系统管理程序不具有在设置定时器中断后修改如何传递定时器中断的灵活性。 因此，如果不通过hypervisor，直接传递计时器中断，则VM设置的计时器可能会误传。如果目标CPU内核处于主机guest mode下，中断可能被错误地传递至hypervisor，如果计时器到期后，另一个VM正在目标CPU内核上运行，则VM设置的计时器可能会传递至错误的VM。

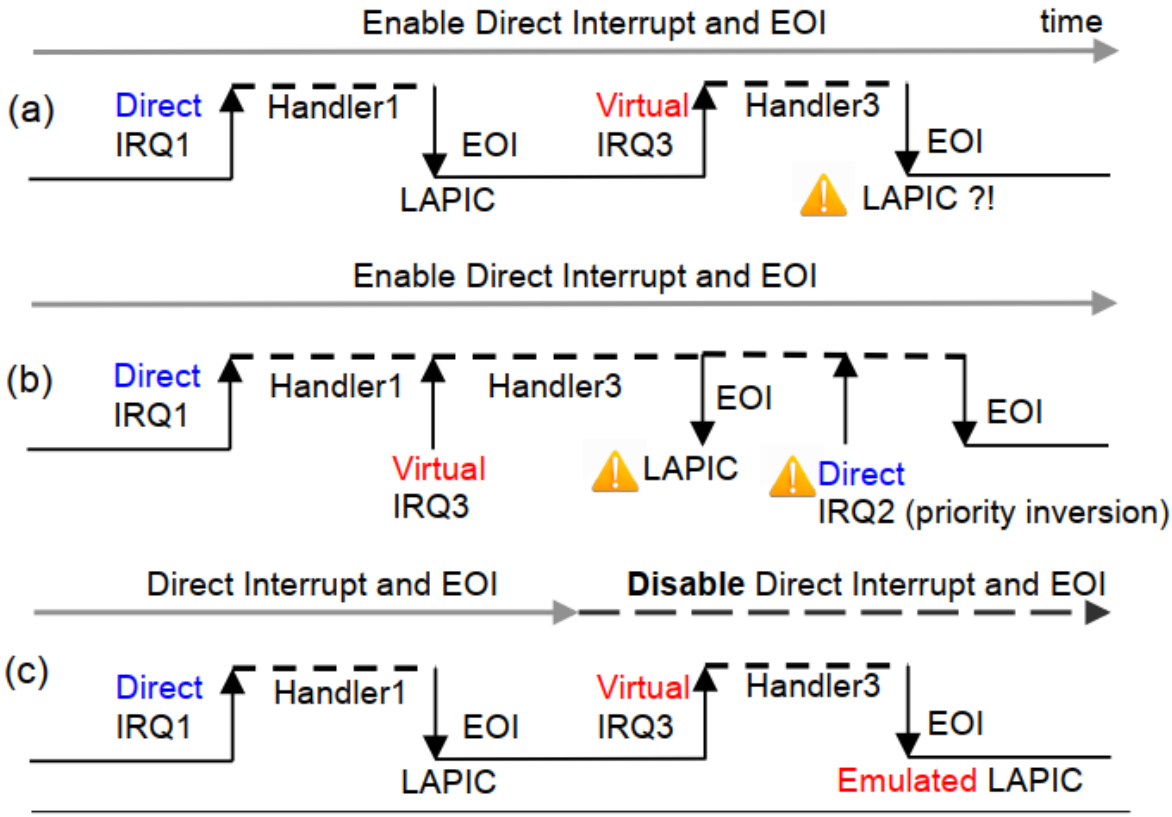

Figure 3. *(a) The LAPIC may receive an EOI write when it thinks there are no pending interrupts. (b) The LAPIC may dispatch an interrupt (IRQ2) when the current interrupt (IRQ1) is not yet done because it receives an EOI write. (c) ELI [18, 19] avoids the confusions caused by direct EOI write by turning off direct interrupt delivery and EOI write whenever at least one virtual interrupt is being handled.*

为了支持定时器中断的直接传递，同时又避免了DID中的传递错误问题，我们将系统管理程序设置的定时器限制到指定的内核。 此外，当系统管理程序在一个CPU内核C上调度虚拟机 M时，M配置的计时器将安装在C的硬件计时器上； 当系统管理程序从CPU内核C调度VM M时，从M的硬件计时器中删除M配置的计时器，并将其安装在指定的CPU内核的硬件计时器上。

我们的设计强制执行以下不变式：除了指定的CPU内核外，CPU内核的硬件计时器上安装的唯一计时器是由该CPU内核上当前运行的VM设置的。 因此，该不变性保证了在直接传递定时器中断时不会发生传递错误的问题。 在指定的CPU内核上，DID hypervisor已准备好服务由hypervisor和当前未运行的那些VM配置的计时器中断。要传递给那些不在运行的VM的计时器中断会在虚拟机恢复时作为虚拟中断传递给它们。

4.5 直接的中断结束写操作

当中断处理程序完成为DID提供的中断服务时，它将写入相关LAPIC上的x2APIC EOI寄存器，以向中断控制器确认当前中断的服务已完成，并且允许中断控制器传递下一个挂起的中断。 x86体系结构允许我们的系统软件选择当VM写入关联的EOI寄存器时是否触发VM退出。 尽管希望在VM写入关联的EOI寄存器时避免VM退出，但是如果写入EOI寄存器不涉及hypervisor，则可能会有不良的副作用，具体取决于hypervisor将虚拟中断注入虚拟机的机制。

由KVM实施的将虚拟中断注入VM的常见方法是在恢复VM之前正确设置VM的VMCS的模拟LAPIC。 但是，这种模拟的LAPIC方法需要EOI写入来触发VM退出，以确保模拟和物理APIC的状态一致。 如果虚拟中断的处理程序直接写EOI，则LAPIC可能会在认为没有挂起的中断时收到EOI通知，如图3（a）所示，或者可能认为当前挂起的中断已经完成，而实际上它仍在进行中，如图3（b）所示。 此外，LAPIC可能会错误地调度优先级较低的中断（例如，图3中的IRQ2）（b）以抢占优先级较高的中断（例如，IRQ1），因为虚拟中断IRQ3的处理程序直接写入EOI寄存器。

此优先级反转问题的根本原因是，通过软件模拟IRR / ISR注入虚拟中断时，LAPIC无法看到虚拟中断。 为了解决这个问题，现有的直接中断传递解决方案[18、19、31]禁止在VM处理任何虚拟中断时对VM进行直接中断传递和直接EOI写入，如图3（c）所示，并且称为 ELI / ELVIS中的注入guest mode。我们在DID中解决此问题的方法不同，因为我们使用self-IPI将虚拟中断注入到VM中。 具体来说，在DIDhypervisor恢复虚拟机之前，它会向其自己的CPU内核发出IPI。 VM恢复后，该IPI然后直接传递到注入的VM。 如果需要将多个虚拟中断注入到VM中，我们的DIDhypervisor将设置多个IPI，每个IPI对应一个虚拟中断。

DID基于IPI的虚拟中断注入机制完全消除了由于直接EOI写入而导致的优先级反转问题。 当虚拟中断以IPI形式交付时，目标CPU内核的LAPIC可以看到它，从而使其能够与其他直接中断和虚拟中断进行竞争。 由于LAPIC会观察传递到其相关的CPU内核的每个中断以及每个EOI写入，因此它使我们的系统不会将服务中的中断误认为实际上已经完成，并且不会过早地传递新的中断。

由于DID使用IPI直接传送虚拟中断，因此常规IPI不再触发我们系统中的VM退出。 对于IPI的原始应用程序，例如关闭CPU内核或刷新远程TLB，我们在DID中使用特殊IPI，其传送guest mode设置为NMI。 NMI设置强制VM在目标CPU内核上退出，从而使DIDhypervisor可以重新获得控制权并采取与特殊IPI相对应的适当操作。

无论DIDhypervisor是否与虚拟中断被注入到的VM在同一CPU内核上运行，我们的DID设计都使用相同的基于IPI的机制（具有适当的中断向量设置）来传递虚拟中断。 我们基于IPI的虚拟中断传递机制有两个关键优势。 首先，当虚拟中断传递中涉及的源和目标在不同的CPU内核上运行时，不需要VM退出。 其次，由于每个虚拟中断均采用硬件中断（即IPI）的形式并经过目标CPU内核的LAPIC，因此无论这些中断是直接还是以其他方式传递的，传递给CPU内核的中断之间的优先级都得以正确保留。

5. 性能评估

5.1 评估方法

为了量化DID的有效性，我们使用各种工作负载测量了每个VM退出的原因和花费的服务时间。 然后，我们通过将每个VM进入和VM退出之间的时间总和作为guest中的总时间，然后将guest中的总时间除以总经过时间，来计算guest时间（TIG）百分比。

评估我们的DID原型时使用的硬件测试平台包括两个Intel x86服务器，它们与两个Intel 10GE 82599 NIC背靠背连接。 DID安装在其中一台服务器上，这是一台Supermicro E3塔式服务器，具有8核Intel Xeon 3.4GHz CPU，具有硬件虚拟化（VT-x）支持和8GB内存。 另一台服务器充当请求生成主机，它配备了8核Intel i7 3.4GHz CPU和8GB内存。 安装了DID的服务器在启用Intel VT-d支持的情况下运行KVM，以便多个虚拟机可以直接访问SRIOV设备而不会受到干扰。

我们在两台服务器上都使用Linux内核版本3.6.0-rc4和qemu-kvm 1.0运行Fedora 15。 我们为每个VM提供一个vCPU，该vCPU固定到特定的内核，1GB内存，Intel SRIOV NIC的一个虚拟功能以及一个使用virtio和vhost [10，26]内核模块的半虚拟化网络设备。

我们以与主机相同的CPU类型设置启动每个VM，并启用x2APIC支持。 由于控制台guest mode（带有-gographic）会由于MMIO触发VM退出而带来额外的性能开销，因此虚拟机开始进入图形用户界面guest mode[23]。 我们还设置了idle = poll以防止HLT指令导致VM退出。 对于计时器实验，我们启用内核参数“ NO HZ”。

我们将所有CPU内核配置为以最大频率运行，因为当CPU内核以节能或按需guest mode运行时，循环测试程序往往会报告更长的延迟。 对于所有网络实验，我们将最大传输单位（MTU）设置为其默认大小1500字节。

对于每种配置，我们都打开和关闭DID以评估DID的好处。本研究使用以下基准程序。

- WhileLoop：运行234个迭代的循环，其中每个迭代执行一个整数加法。
- Cyclictest：用于测量中断调用延迟的程序（生成硬件中断的时间与用户级循环测试程序中的相应处理程序获得控制的时间之间的平均时间间隔）。我们在专用内核上以最高优先级运行循环测试，以每毫秒1个的速率测量100,000个中断。
- PacketGen：基于UDP的程序，以每秒100K，250K，400K和600K数据包的速率向基于UDP的接收器发送128字节UDP数据包，其中发送程序和接收程序均以最低优先级运行。
- NetPIPE [28]：乒乓测试，用于测量两台机器之间的半程往返时间。在我们的实验中，我们将消息大小从32字节更改为1024字节。
- Iperf [30]：用于测量两台计算机之间的TCP吞吐量的程序。我们报告了五次平均100秒的运行。
- Fio [2]：执行4KB随机磁盘读写的单线程程序，对通过禁用了高速缓存的1GB ramdisk通过virtio支持的虚拟磁盘进行写入。
- DPDK l2fwd [21]：支持线速网络数据包转发的用户级网络设备驱动程序和库。
- Memcached [5，17]：键值存储服务器。我们模拟类似Twitter的工作负载，并测量每秒服务的峰值请求（RPS），同时至少对95％的请求保持10ms的延迟。
- SIP B2BUA [9]：SIP（会话发起协议）背对背用户代理服务器软件，可维护完整的呼叫状态和请求。我们使用SIPp [8]每秒建立100个呼叫，每个呼叫持续10秒。

5.2虚拟机退出率降低

在具有VT-x的64位Intel x86架构中，有56个VM退出的可能原因。 每个VM退出均导致了它在hypervisor的退出处理程序，并减少了VM的CPU周期数。 我们将在I / O密集型工作负载下触发VM退出的最常见原因确定为（1）EXTINT：外部中断的到来，其中包括虚拟机管理程序的I / O线程发送的IPI和SRIOV和para的硬件中断虚拟设备，（2）PENDVINT：向虚拟机发出之前不可中断的虚拟中断通知，（3）MSRWR：VM尝试写入模型专用寄存器（MSR）（例如，编程LAPIC寄存器和 EOI寄存器），以及（4）IOINSR：VM尝试执行I / O指令（例如，配置硬件设备）。

为了评估在网络密集型工作负载下DID的有效性，我们测量了正在运行的VM以不同速率接收UDP数据包时，其VM退出率。具体来说，我们在配备SRIOV NIC的测试服务器上针对具有DID的系统对了Vanilla KVM Linux进行了测量。我们使用了在SRIOV NIC上配备了VF的测试VM，并在测试VM中运行了UDP接收程序，使用Linux内核的ftrace工具收集了VM退出统计信息，同时另一个程序将UDP数据包发送到测试VM内部的接收器。如图4所示，当测试VM上的UDP数据包速率达到每秒10万个数据包时，VM退出速率达到每秒28K个退出，其中96.1％的时间花费在guestguest mode（TIG）上。 VM退出的两个主要原因是外部中断（EXTINT）和写入特定于模型的寄存器（MSRWR）。由于此测试中使用的NIC支持SRIOV，因此大多数外部中断来自于在接收UDP数据包时分配给测试VM的VF生成的MSI-X中断。使用半虚拟网络设备时，外部中断退出是由后端驱动程序（通常是QEMU I / O线程）发送的IPI（处理器间中断）引起的。此外，通过分析每个MSR写操作的目标，我们得出结论，写到EOI（中断结束）寄存器占MSR写操作的99％以上。当测试VM每秒接收100K数据包时，仅观察到每秒28K VM退出这一事实，表明NIC支持中断合并。随着数据包速率增加到250K，400K和600K，VM退出速率分别增加到62K，90K和118K，guest时间（TIG）分别减少到87.48％，83.3％和80.42％。由于NIC在更高的数据包速率下更积极地合并中断，因此VM退出速率的增长速度不随数据包速率线性增长。

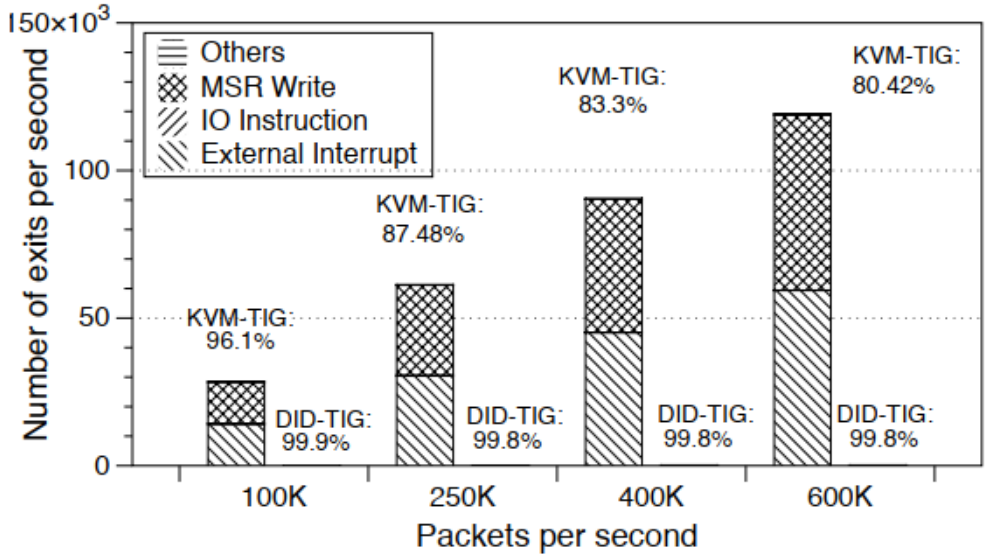

Figure 4. *The breakdown of VM exit reasons for a test VM running on KVM when it is receiving UDP packets through an SRIOV NIC at different rates and when DID is turned on or off*

图4显示DID消除了由于外部中断和EOI写入而导致的几乎所有VM退出，并且无论UDP数据包速率如何，都将VM退出速率降低到每秒1K以下。 使用DID时，VM退出的主要原因是I / O指令（IOINSR），guestVM的驱动程序（即SRIOV VF驱动程序和virtio-net / virtio-blk）用于对分配的VF的配置寄存器进行编程，例如 发送/接收环形缓冲区中的描述符。 当测试VM以每秒600K的速度接收数据包时，DID避免了不必要的VM退出，从而节省了CPU时间的（99.8-80.42 = 19.38％）。

5.3 应用级CPU节省

为了量化DID在应用程序级别上的性能优势，我们在运行Linux（裸机）的物理计算机上，在没有DID的KVM下的Linux VM（原始KVM）和在带有DID的KVM下的Linux VM（KVM）上运行WhileLoop程序+ DID）。 WhileLoop程序不执行任何特权指令，因此在其执行期间不会导致VM退出开销。同时，我们在后台运行UDP接收程序，以不同的速率接收UDP数据包。图5显示，对于所有测试的数据包速率，KVM + DID配置中WhileLoop的总经过时间几乎与裸机配置的相同。这是因为DID消除了几乎所有VM退出开销，从而使绝大多数的CPU时间都可以在执行WhileLoop程序时在guestguest mode下花费。相反，原始KVM配置中WhileLoop的经过时间随着UDP数据包速率的增加而增加，因为较高的数据包速率会导致更多的VM退出开销，从而降低TIG。因此，对于测试的数据包速率，KVM + DID相对于原始KVM的WhileLoop性能提升为3.4％，9.6％，13.4％和18.47％。如图5所示，性能提升与启用的DID TIG降低密切相关，分别为3.8％，11.42％，16.6％和19.38％。

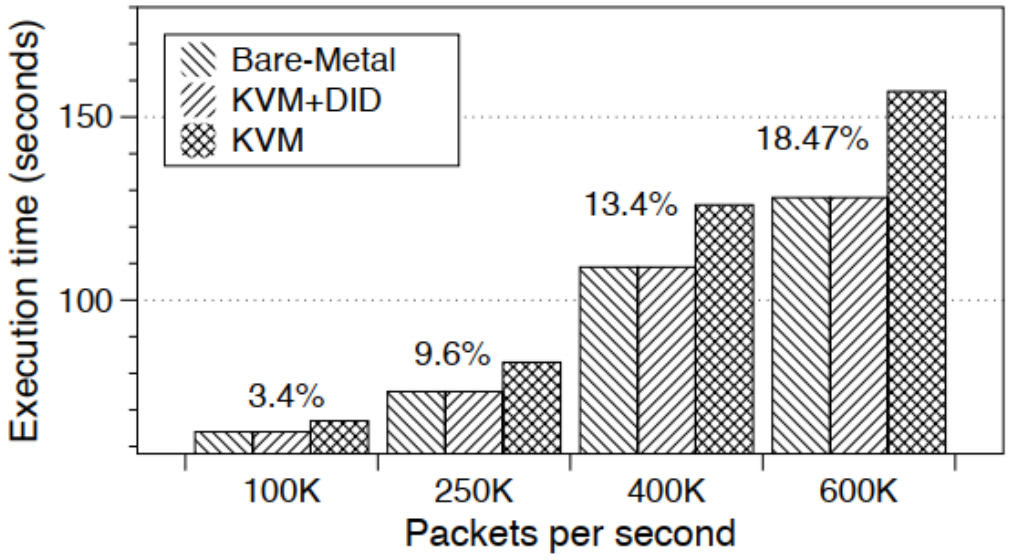

Figure 5. *The execution time of a while-loop program on a bare metal machine, a VM running on KVM, and on KVM with DID, when there is a background load of receiving UDP packets through an SRIOV at different rates.*

5.4中断调用延迟

除了减少VM退出开销外，DID的另一个主要性能优势是减少了中断调用延迟，这减少了生成硬件中断与相应的中断处理程序开始处理之间的时间。 减少中断调用延迟对于实时计算系统至关重要，因为它可以减少最坏情况下的延迟范围。 DID通过从中断传递路径中删除管理程序来减少中断调用等待时间。 我们使用循环测试程序来评估DID的中断调用延迟，在这种情况下，该延迟被专门定义为计时器生成中断与调用用户级循环测试程序进行处理之间的时间差。 在原始KVM配置中，中断是通过管理程序间接传递的，影响中断调用延迟的因素有：

1. 系统管理程序可以暂时禁用中断传递，从而延迟从硬件设备到系统管理程序的中断传递。
2. 在将收到的中断转换为虚拟中断并将其注入其目标VM之前，系统管理程序可能会引入其他延迟。
3. VM的guestOS可能会禁用中断，从而延迟从管理程序到guestOS的虚拟中断传递。
4. 在guestOS处理传入的虚拟中断之后，安排循环测试程序可能会有延迟。

在此测试中，我们将循环测试程序的调度优先级提高到尽可能高的水平，从而减少了上述第四因素的差异。 但是，前三个因素由管理程序和guest OS中的中断机制确定。

图6绘制了运行循环测试程序的100,000个计时器操作后，裸机，原始KVM和KVM + DID配置的中断调用延迟的概率密度函数。原始KVM的平均中断调用延迟为14µs。不出所料，此配置具有最高的中断延迟，因为循环测试程序中的每个计时器操作都至少需要退出三个VM才能设置LAPIC计时器（特别是TMICT寄存器），接收计时器中断并确认计时器的完成中断。 上述第二个因素相关的延迟和可变性的增加主要与VM退出有关。

KVM + DID的平均中断调用等待时间为2.9µs，这是因为DID消除了由于TMICT寄存器编程，定时器中断传送和EOI写入而导致的所有VM退出。尽管接近裸机，但KVM + DID的平均中断调用等待时间要高0.9µs。尽管大多数定时器中断都是在DID下直接传递给CPU内核的，但在中断时，目标CPU内核可能处于host mode而不是guest mode。发生这种情况时，管理程序将设置self-IPI位图，以在恢复guest执行时生成目标VM的计时器中断。因此，当系统管理程序接收到中断时，中断调用等待时间将增加，以使管理程序完成正在进行的操作所花费的时间。在我们的测试中，即使在空闲的虚拟机中，每秒仍然有大约500个虚拟机退出，其中大部分是由于虚拟机中的I / O指令和扩展页表（EPT）违反所致。这些VM退出的服务时间导致了裸机和KVM + DID之间的较小的中断调用延迟时间。

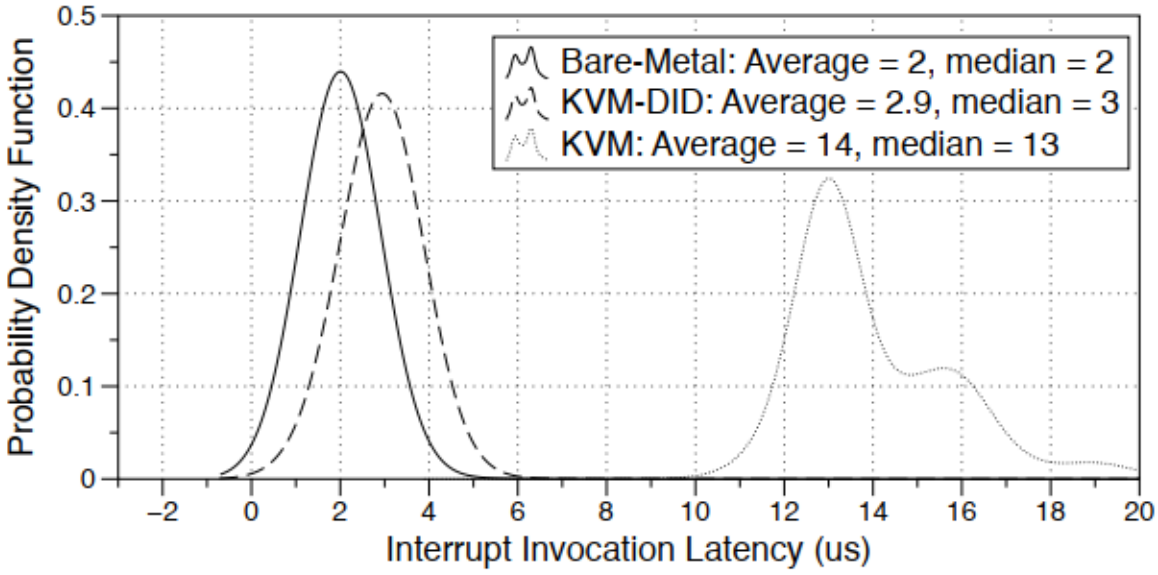

Figure 6. *The probability density function of the interrupt invocation latency of the bare metal, vanilla KVM and KVM+DID configuration during a 100-second cyclictest run*

在100秒的循环测试运行中，有100,000个计时器中断，其中991个VM退出是由于违反EPT而导致的，平均VM退出服务时间为9.9µs，6550个VM退出是由于I / O指令导致了，且平均VM退出服务时间为8.65µs。在此期间，目标CPU内核处于guest mode或VM退出期间，仅向目标CPU内核传递了3,830个计时器中断；这些VM退出的服务时间分配如图7所示。3830个定时器中断中的1782个由于违反EPT而在VM退出中占地，平均VM退出服务时间为11.07µs，而其余定时器中断在VM退出中占地是由于I / O指令，平均VM退出服务时间为24.11µs。结果，在运行100秒的过程中，这些VM退出服务时间对计时器中断的调用延迟的总贡献为84,300µs。由于裸机配置的平均中断调用等待时间为2µs，因此KVM + DID中的平均中断调用等待时间可以近似为（（100,000-3,830）* 2 + 84,289）/ 100,000 = 2.76µs，与实测值相近结果为2.9µs。

5.5网络性能优势

为了衡量DID对数据包延迟的影响，我们使用了NetPIPE基准测试工具[28]，该工具采用乒乓测试来测量两台服务器之间的半程往返时间。图8显示了NetPIPE报告的延迟测量，因为我们将消息大小从32字节更改为1024字节。每个邮件大小的最左侧的三个条形对应于在外部服务器和另一台以裸机配置（SRIOV-BM），KVM + SRIOV + DID配置（SRIOV-DID）运行的服务器之间测量的NetPIPE结果。 KVM + SRIOV配置（SRIOV）。使用SRIOV NIC时，对VF进行编程不会触发VM退出。结果，SRIOV-BM配置和SRIOV-DID配置的数据包延迟之间没有明显的区别，因为后者不会在中断传递时导致VM退出。在SRIOV配置中，仅观察到的VM退出是由于VF和EOI写入产生的中断，导致这些VM退出的平均服务时间分别为0.85µs和1.97µs。在SRIOV的情况下，执行NetPIPE基准测试时，我们观察到每个收到的数据包有两种类型的VM退出。第一个退出归因于外部中断的到来，指示数据包的到达，而第二个VM归因于中断的确认（EOI）。 EOI的平均退出处理时间为0.85us，而外部中断的eixt处理时间为1.97us。因此，SRIOV配置的平均数据包延迟比SRIOV-BM配置的平均数据包延迟高约2.44µs，相当于1.97 + 0.85 = 2.82µs。当数据包大小增加时，等待时间也会增加，因为每字节开销开始控制数据包等待时间，因此，数据包等待时间随数据包大小而增加。

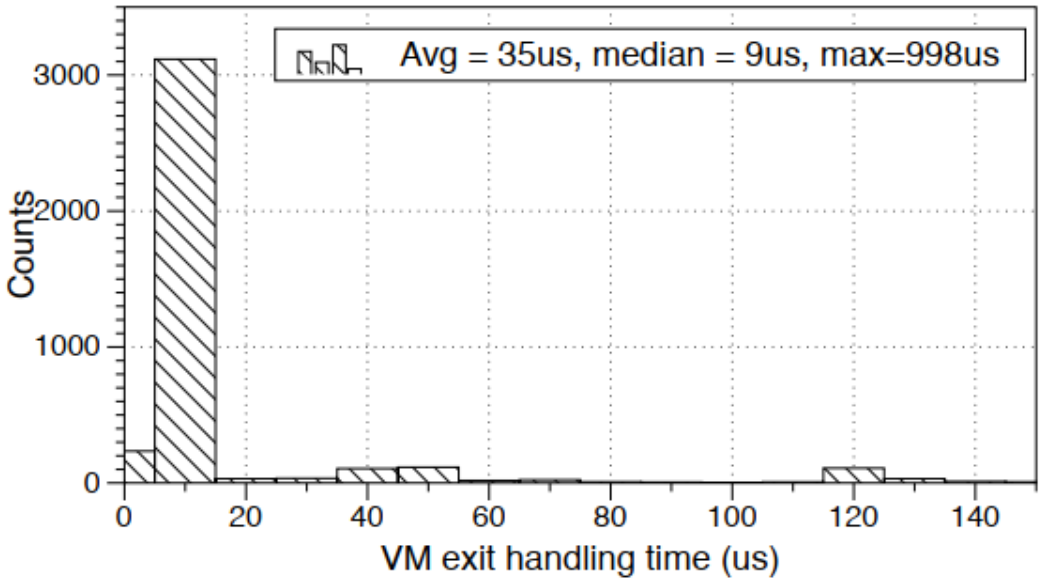

Figure 7. *Service time distribution of the VM exits during which a timer interrupt of a cyclictest program run is delivered*

图8中最右边两个带状图显示了每个消息的大小，对应于NetPIPI的测量结果，在Linux-KVM主机上运行了一个进程，在同一Linux-KVM上运行的VM内运行一个进程，依次在同一台Linux-KVM计算机上运行，且控制DID打开（PV-DID）或关闭（PV）。这些进程通过半虚拟化前端驱动程序，virtio-net后端驱动程序和Linux虚拟网桥相互通信。从理论上讲，每个数据包交换都需要三个VM退出，一个退出用于中断传递，另一个退出用于EOI写入，第三个退出用于更新后端设备的内部状态。实际上，与由EOI写入导致的VM退出数量相比，virtio-net实现对处理多个数据包所需的设备状态更新进行批处理，并显着减少了由于I / O指令而导致的VM退出数量。并中断传送。因此，PV配置的平均数据包等待时间比PV-DID配置的平均数据包等待时间最多高2.41μs，这与由EOI写入和中断传递导致的VM退出的平均服务时间之和相当。尽管SRIOV和SRIOV-DID的数据包延迟随消息大小而增加，但是PV和PV-DID的数据包延迟与消息大小无关，因为当在同一消息包内交换数据包时，后者不会复制消息的有效负载物理服务器[10，26]。

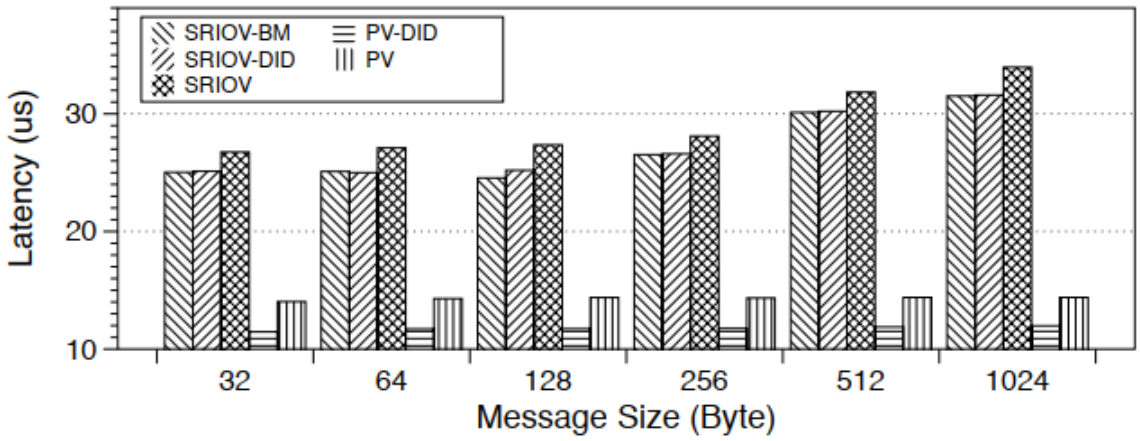

Figure 8. *One-way packet latency between an external server and another server running in the bare metal configuration (SRIOV-BM), the KVM+SRIOV+DID configuration (SRIOV-DID) and the KVM+SRIOV configuration (SRIOV), and between a process running directly on top of a Linux-KVM machine and another process running inside a VM that in turn runs on the same Linux-KVM machine with DID turned on (PV-DID) or off (PV)*

为了量化DID的网络吞吐量收益，我们使用了iperf工具[30]。我们的结果表明，在10Gbps链路上，即使SRIOV-DID的TIG有所改善，SRIOV-DID配置的iperf吞吐量也为9.4Gbps，比SRIOV配置（9.3Gbps）的吞吐量高1.1％。超过SRIOV是16.8％。由于物理网络链接的原始容量几乎可以满足要求，因此节省的CPU时间无法完全转化为网络吞吐量的提高。在机器内连接上，PV-DID配置的iperf吞吐量为24Gbps，比PV配置（19.8Gbps）好21％，尽管PV-DID的TIG改进仅优于PV 11.8％。 CPU时间的节省比网络吞吐量的增加要多得多，因为实际上没有任何有效负载被复制用于机器内通信，因此CPU时间的减少不会直接转化为吞吐量的增加。

另一方面，我们还发现DID在DPDK l2fwd基准方面没有显示出明显的改进。对于DPDK，我们设置了SRIOV NIC，并使用VM的VF设备执行了DPDK的第2层转发程序l2fwd。我们使用DPDK版本的Pktgen生成从请求生成服务器到虚拟机的转发流量，并测量l2fwd程序处理的最大接收和转发数据包数量。由于DPDK的轮询特性，所有网络数据包都通过VF设备传送到l2fwd程序，而不会触发任何中断。结果，无论有无DID，l2fwd都能够每秒转发790万个128字节的数据包。

5.6块I / O性能优势

为了分析DID在高性能直接连接的磁盘I / O系统（例如固态磁盘阵列）下的性能优势，我们在主机上配置了1GB的虚拟磁盘，并将其暴露给使用该虚拟机在主机上运行的测试VM virtio-blk，并在测试VM中运行Fio基准测试。我们测量了IOPS和I / O完成时间，这是Fio发出I / O请求与该请求完成并返回给Fio之间的时间差。图10显示，当DID关闭时，IOPS为14K，平均I / O完成时间为34µs。当DID打开时，IOPS增加到14.7K，平均I / O完成时间为32µs。这些性能差异再次是由于DID消除了由于中断传递（EXTINT）和MSRWR写入而导致的VM退出的事实。如预期的那样，DID的性能增益受到块I / O速率的限制，因此相关的中断速率通常要低得多。
与iperf不同，在iperf中，由于中断传递而导致的VM退出次数与MSRWR写操作的次数大致相同，Fio观察到，由于MSRWR写入而导致的VM退出次数是中断传递导致的VM退出次数的三倍。对Fio基准的分析表明，该程序在提交I / O请求之前设置了计时器，以保护自己免受磁盘无响应的影响，并在每个请求完成后清除计时器。因此，对于每个I / O请求，需要三个MSRWR写入，一个用于EOI写入，两个用于TWICT写入。 DID成功-完全消除了由于这些MSR写入而导致的所有VM退出。

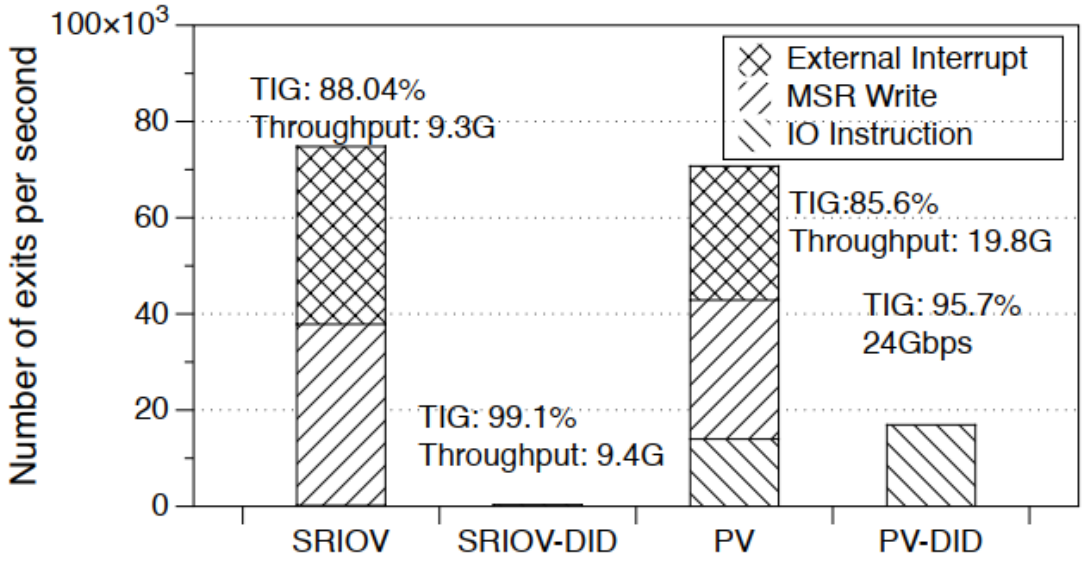

Figure 9. *The iperf throughput improvement of DID when a SRIOV NIC is used for inter-server communication and a paravirtualized NIC is used for intra-server communication*

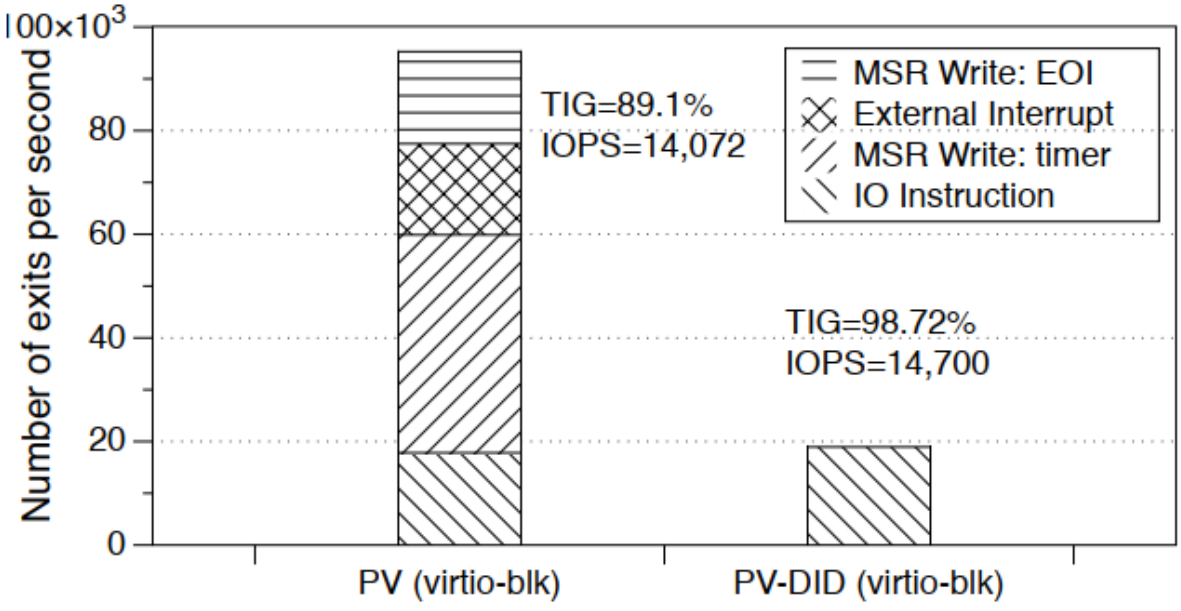

Figure 10. *The VM exit rate and the breakdown of VM exit reasons of the Fio benchmark*

5.7 Memcached工作负载

为了评估DID在常见服务器工作负载上的性能改进，我们在VM内设置了双线程Memcached 600MB服务器（测试VM配置了一个vCPU和1GB RAM）。我们生成了600MB的数据集，并通过预加载数据集来预热服务器。然后，我们运行Memcached客户端模拟器，该模拟器创建8个线程和200个TCP / IP连接，获取/设置比率为4：1。为了保证每个实验的服务质量，我们凭经验找到了峰值请求速率，该速率使服务器可以在10毫秒内完成所有请求的95%。我们在客户端和服务器端都关闭了Nagle的算法（TCP nodelay选项）。

图11显示了PV，PV-DID，SRIOV和SRIOV-DID配置的VM退出率，其RPS分别为47.3K，50.4K，45.8K和151.5K。SRIOV-DID具有SRIOV和DID的优点，并且消除了大多数VM退出，TIG为99.8%，因此它在很大程度上优于所有其他配置。我们比较了Memcached服务器在相同硬件的裸机设置下的性能，观察到152.3K RPS，仅比SRIOV-DID高0.6%。第二好的设置是PV-DID，TIG为99.7%，其次是PV配置，TIG为97.7%。值得注意的是，SRIOV以81.55%的TIG位居最后。尽管由于I / O指令，SRIOV不会导致任何VM退出开销，但SRIOV的性能仍然比PV差，因为与PV相比，由于中断传递和EOI写入，SRIOV会导致大量VM退出。在PV配置中，虚拟主机线程会定期轮询物理NIC，对传入的数据包进行批处理，然后中断目标VM中的前端驱动程序。结果，PV配置中每个中断传递给目标VM的数据包数量明显高于SRIOV配置中。

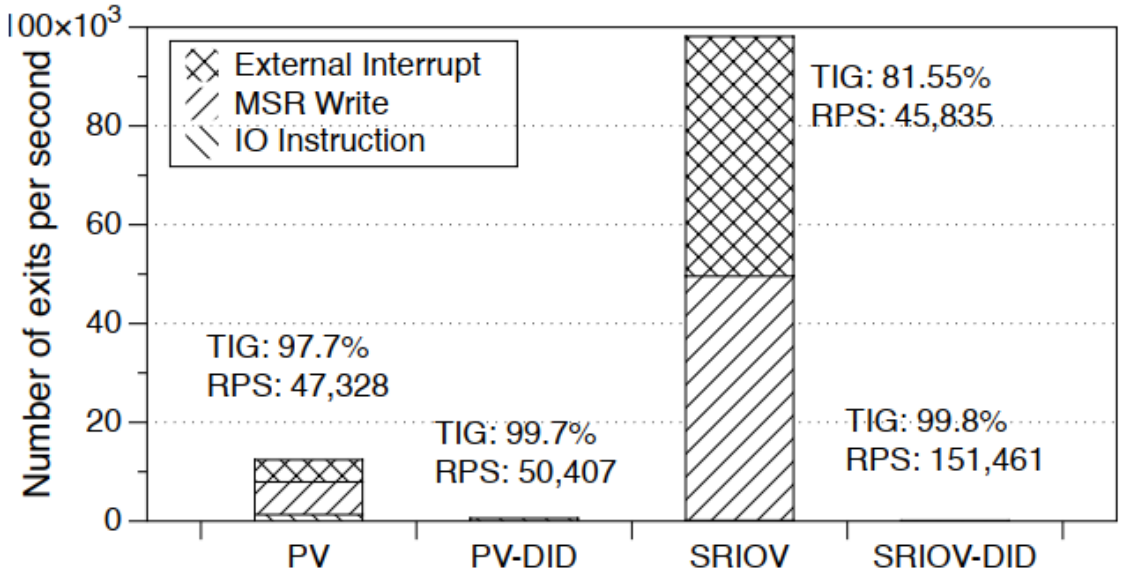

Figure 11. *The VM exit rate and the break-down of VM exit reasons of a Memcached server under the PV, PV-DID, SRIOV, and SRIOV-DID configurations*

实现与PV配置中的轮询虚拟主机线程相同的中断聚合优势的一种方法是利用Linux的NAPI工具，该工具旨在在传入的中断速率超过特定阈值时通过轮询来减轻中断开销。为了确保通过轮询降低中断速率是SRIOV性能不佳的原因，我们将Linux VM的NAPI阈值从其默认值64降低到32、16、8和4，从根本上增加了可能性。 -guest的SRIOV VF驱动程序以轮询模式运行。当NAPI阈值设置为4或8时，SRIOV配置的结果RPS上升到48.3K，比PV配置提高。但是，将NAPI阈值降低到4或8的代价是将CPU利用率分别提高3%和6%。这些结果证实，在某些情况下，仔细的调整可以减轻SRIOV的VM退出开销，使其与PV相当。

除了更高的CPU使用率外，由于请求批处理，PV和PVDID配置还增加了请求等待时间。由于增加了请求等待时间，因此无法以相同的请求速率实现服务质量目标。这解释了为什么即使PV-DID和SRIOV-DID之间的TIG差异仅为0.1%，SRIOV-DID的RPS仍比PV-DID的RPS约高三倍。

5.8 VoIP工作量

为了评估B2BUA系统中DID的性能优势，我们将SIPp [8] UAC（用户代理客户端）配置为请求生成主机上的呼叫始发端点，DID服务器上VM内的B2BUA服务器。 SIPp UAS（用户代理服务器）作为DID服务器的管理程序域上的呼叫应答端点。 UAS和UAC之间的所有SIP消息均由B2BUA的呼叫控制逻辑处理和转发。具体来说，通过向B2BUA的呼叫控制逻辑发送INVITE消息来执行UAC与UAS之间的呼叫，该消息将执行身份验证和授权。然后，B2BUA将INVITE消息转发给应答端点UAS。收到INVITE消息的UAS将开始振铃并发送回180 SIP临时响应。应答端点接听电话后，就会将200 OK SIP消息发送到始发端点并建立会话。由于我们每秒设置100个呼叫，每个呼叫持续10秒，因此B2BUA中维护的最大同时呼叫会话数为1000。

表2显示了五种配置下的呼叫会话建立延迟。对于每个实验，我们将UAC配置为进行10,000个呼叫，并测量呼叫会话的建立等待时间，这是从UAC发送INVITE消息到UAC接收200 OK消息的时间。我们观察到，尽管UAC每秒产生100个呼叫，但是我们可以实现的最佳平均呼叫速率是BareMetal配置为90.9，SRIOV-DID配置为90.8。影响呼叫速率结果的重要因素是重传的INVITE消息的数量。 PV显示出最低的85.5呼叫速率，因为它招致了更多的INVITE消息重发。对于会话建立延迟，除了Bare-Metal配置外，SRIOV-DID的9061呼叫建立在10ms内完成时表现最佳，而PV表现最差，有8159个呼叫建立在10ms内完成和1335个呼叫设置在200毫秒内完成。针对SRIOV，SRIOV-DID，PV和PV-DID测得的VM退出率分别为4608、1153、6815和1871。总体而言，DID对SRIOV和PV的改进来自于通过避免VM退出而将更多的CPU时间保持在guest mode下结果，允许B2BUA服务器处理更多的SIP消息并降低总体会话建立延迟。

	<10	10-100	100-200	>200	Call Rate	INVITE Retrans.
Bare-Metal	9485	112	147	256	90.9	79
SRIOV	8342	186	248	1224	86.8	5326
SRIOV-DID	9061	159	242	538	90.8	2440
PV	8159	243	263	1335	75.6	5961
PV-DID	8473	280	61	1186	85.5	4920

Table 2. *Call session set-up latency (ms) distribution of 10,000 calls processed by SIP B2BUA server .*

5.9虚拟机退出APIC虚拟化分析

为了分析APICv的性能优势，我们设置了一台配备Intel Xeon E5-2609v2 CPU，16GB内存的服务器，并安装了Linux内核3.14，并在KVM中提供了对APICv的最新支持。我们提供三种类型的工作负载下的VM退出率：代表LAPIC定时器中断的循环测试工作负载，用于虚拟中断的Iperf-PV TCP工作负载以及用于外部中断的Iperf-SRIOV TCP工作负载。图12显示了结果，从左到右的每个条形表示原始KVM设置，启用了APICv的KVM，启用了DID和禁用APICv的KVM。

循环测试结果显示，与APICv关联的MSR Write VM退出数量是普通KVM数量的一半。这是因为APICv通过EOI虚拟化避免了EOI退出，而其余的MSR Write退出是由对定时器寄存器（TMICT）进行编程引起的。相反，DID完全消除了这些类型的VM退出。对于Iperf-PV实验，APICv在减少VM退出次数方面与DID相同。这是因为APICv的发布中断机制可以在不触发VM退出的情况下从后端驱动程序向运行VM的虚拟机传递虚拟中断，而DID可以达到相同的效果，而无需修改客户操作系统或需要硬件支持。最后，在Iperf-SRIOV实验中，APICv表明，尽管EOI虚拟化有助于消除MSR写入退出，但到达运行内核的VM的外部中断仍会触发VM退出。作为比较，DID禁用VMCS中的EIE位，以便外部中断不会触发任何VM退出。

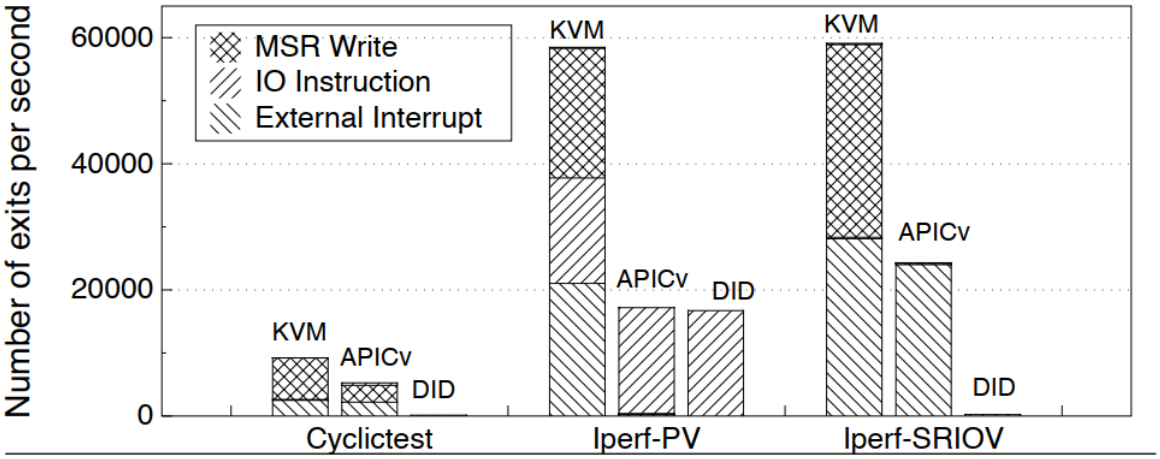

Figure 12. *The VM exit rate and the breakdown of exit reasons under KVM, KVM with APICv support, and DID.*

6.讨论

中断在两种情境中被触发和处理。中断要么是由给VM配置的直通设备触发的，要么是由给主机配置的设备触发的。当系统未完全加载（具有可用的备用物理内核）时，DID将主机的中断定向到备用物理内核，从而避免了对正在执行VM的内核的干扰。因此，来自主机设备的中断永远不会传递给运行VM的内核。但是，当系统超额预订时，发往主机的中断可能会到达执行VM的内核，因为主机和VM正在分时共享物理内核。在这种情况下，DID将主机设备配置为以NMI模式传递中断。当设备触发发往主机的中断，但此中断到达运行VM的内核时，NMI会强制VM退出并将控制权传递给主机。主机的中断处理程序（在Linux中为IRQ）检查中断的向量号，并根据主机的IDT将中断分派给主机的中断处理程序。请注意，为NMI模式配置中断不会丢失该中断的原始向量号。结果，当控制权传递给主机时，主机知道中断源。

DID不仅为硬件中断配置NMI，而且还为虚拟机管理程序触发的IPI配置NMI。由于DID使用IPI将虚拟中断直接发送到目标VM，因此，主机最初使用的IPI（用于重新安排中断和TLB关闭的操作）必须使用NMI模式中断来强制VM退出。 NMI模式IPI会触发VM退出，并使用中断的原始向量号来调用主机的中断处理程序。请注意，NMI可能到达已经在主机模式而不是guest mode下运行的内核。由于DID能够识别中断的源设备或内核，因此它可以正确地区分该中断是针对guest的，是否需要生成一个自身IPI，还是该中断是针对主机的，并且需要直接调用相应的中断?

7.结论

I / O虚拟化的性能开销源于由于I / O指令和中断传递而导致的VM退出，这些中断又包括中断分派和中断结束（EOI）确认。尽管SRIOV旨在消除由于I / O指令而导致的VM退出，但本文提出了DID，这是解决虚拟服务器上的中断传递问题的综合解决方案。 DID完全消除了由于SRIOV设备，半虚拟设备和计时器的中断调度和EOI通知而导致的大多数VM退出。结果，就我们所知，DID展示了文献中发布的最高效（即使不是最高效）的中断传递系统之一。 DID通过利用IOAPIC的中断重新映射硬件，避免了直接中断的错误传递而实现了这一壮举，并采用了self-IPI机制来注入虚拟中断，从而实现了直接EOI写入而不会引起中断之间的优先级反转。除了改善延迟和吞吐量之外，DID还大大减少了中断调用延迟，从而形成了网络功能虚拟化的关键技术构建块，网络功能虚拟化旨在在虚拟化的IT基础架构上运行电信功能和服务。