arXiv:2304.05148v1 [cs.AR] 11 Apr 2023

采用 I/O 队列直通的高性能、可扩展的软件型 NVMe 虚拟化机制

Yiquan Chen
浙江大学
阿里巴巴集团
中国

Zhen Jin
浙江大学
中国

Yijing Wang
阿里巴巴集团
中国

Yi Chen
浙江大学
中国

Hao Yu
阿里巴巴集团
中国

Jiexiong Xu
浙江大学
中国

Jinlong Chen
浙江大学
中国

Wenhai Lin
浙江大学
中国

Kanghua Fang
阿里巴巴集团
中国

Chengkun Wei
浙江大学
中国

Qiang Liu
阿里巴巴集团
中国

Yuan Xie
阿里巴巴达摩院
美国

Wenzhi Chen
浙江大学
中国

摘要

NVMe（非易失性内存快速通道）是固态硬盘（SSD）的一项行业标准，已在数据中心广泛采用。在云计算中，NVMe 虚拟化至关重要，因为它使虚拟机（VM）能够使用虚拟化的 NVMe 设备，从而提高存储资源的利用率。然而，传统的基于软件的解决方案虽然具有灵活性优势，但往往要以性能下降或较高的 CPU 开销为代价。另一方面，硬件辅助解决方案能够提供高性能和低 CPU 使用率，但其应用通常受到专用硬件支持需求或新硬件开发要求的限制。

在本文中，我们提出了 LightIOV，这是一种新颖的基于软件的NVMe 虚拟化机制，无需消耗宝贵的 CPU 资源，也无需专用硬件支持，即可实现高性能和可扩展性。LightIOV 可以在每台服务器上支持数千个虚拟机。LightIOV 背后的核心思想是 NVMe 硬件 I/O 队列直通，使虚拟机能够直接访问NVMe 设备的 I/O队列，从而消除虚拟化开销并提供接近原生的性能。我们的实验结果表明，LightIOV 能够提供与 VFIO相当的性能，其每秒输入输出操作数达到 VFIO 的 97.6%–100.2%。此外，在高密度虚拟机环境中，LightIOV 在运行 200个虚拟机时，延迟比 SPDK-Vhost低 31.4%，并且在真实应用程序中每秒操作数性能提升 27.1%。

CCS 概念：• **计算机系统组织** → **云计算**。

关键词：云计算、虚拟机、NVMe 虚拟化、I/O 队列直通

1 引言

存储虚拟化是数据中心的一个关键方面，因为它能够为虚拟机提供存储资源，将虚拟设备与物理设备解耦，并支持将多个虚拟设备灵活映射到单个物理设备 [42]。这样，存储虚拟化可以提高资源利用率，并为复杂功能提供简单且一致的接口 [24]，例如在线迁移 [27] 和整合 [45]。

随着技术不断发展，存储设备也在不断演进，以满足数据中心对性能的需求。与硬盘驱动器（HDD）相比，固态硬盘（SSD）在吞吐量和延迟方面表现更优[11, 12, 36]。此外，非易失性内存快速通道（NVMe） [18] 接口显著提升了固态硬盘相较于传统 SATA 接口 [49]的输入/输出性能。因此，NVMe 接口已成为业界标准

表1。 现有 NVMe 虚拟化与 LightIOV 的比较

	硬件辅助的					基于软件的			
	VFIO [46]	SR-IOV [44]	SIOV [22]	LeapIO [30]	FVM [28]	Virtio [43]	SPDK-Vhost [51]	Mdev-NVMe [40]	***LightIOV***
高性能	✓	✓	✓		✓		✓	✓	✔
低开销	✓	✓	✓	✓	✓				✔
灵活性 *	✓					✓	✓	✓	✔
高可扩展性		✓	✓	✓	✓	✓	✓	✓	✔

* 独立于特殊硬件支持。

，并广泛应用于固态硬盘；NVMe 固态硬盘也广泛部署在 AWS [7], Alibaba Cloud [6], Microsoft Azure[35], 和 Google Cloud [20]等数据中心。

在数据中心，云服务提供商使用 NVMe 虚拟化机制，将 NVMe 设备虚拟化并提供给多个虚拟机。这些机制应具备**高性能**、**低开销**和**灵活性**。NVMe 虚拟化机制的首要目标是尽量减少虚拟化造成的性能下降，确保性能尽可能接近裸机。其次，这些机制应以较低的开销运行，并尽可能少地占用服务器资源（例如 CPU 核心）。最后，灵活性也是一个需要考虑的重要因素。无需额外硬件即可实现 NVMe 设备虚拟化是最佳选择。使用专用 NVMe 固态硬盘（例如 SR-IOV）或额外的虚拟化硬件，会降低云服务提供商的灵活性。

可扩展性是一个日益重要的方面，因为部署在单台服务器上的虚拟机密度持续增长。目前，在单台服务器上部署数百台虚拟机已十分常见， [52], 而轻量级虚拟机也被广泛用于无服务器计算。市场上的双路服务器目前能够支持多达 384个CPU核心/超线程（HT）， [9], 凭借其强大的计算能力，单台服务器的虚拟机密度可以轻松达到数千台。此外，无服务器计算的发展趋势日益明显。轻量级虚拟机 [4, 31, 34],相比功能完整的虚拟机消耗更少的 CPU 资源和内存，出于安全性和隔离方面的考虑，通常被用于无服务器计算。在现有的云数据中心中，每台服务器上已经部署了数千台轻量级虚拟机 [4, 31]。因此，NVMe 虚拟化机制必须具备高可扩展性，以支持多达数千台虚拟机。

现有方法的局限性。 NVMe虚拟化机制应具备高性能、低开销、灵活性和高度可扩展性。然而，如表1所示，现有的基于软件和硬件辅助的解决方案在这些方面存在若干局限。

- 基于软件的解决方案 （例如，virtio [43],SPDK vhost-NVMe [51], Mdev-NVMe [40]）采用纯软件方式对 NVMe 设备进行虚拟化。因此，它们能够适用于通用 NVMe 设备，并展现出较高的灵活性。然而，现有的基于软件的解决方案会导致严重的性能下降或较高的 CPU 开销。Virtio [43] 是一种早期方法，在客户机操作系统与虚拟机监控器之间创建虚拟设备接口。但它会显著限制 NVMe 固态硬盘的高性能，其吞吐量仅达到原生性能的 50% [40]。随后，人们提出了基于轮询的解决方案（例如，SPDK vhost-NVMe [51], MDev-NVMe [40]）。它们使用专用 CPU 核心执行虚拟 I/O [19, 26, 33] ，以提高虚拟化 NVMe 固态硬盘的 I/O 性能。然而，所占用的主机 CPU 核心对于云服务提供商而言非常宝贵。例如，在我们的云实例中，我们采用 SPDK vhost-NVMe 解决方案。一台配备 12 个固态硬盘的典型服务器需要分配 8–10 个 CPU 核心来执行存储虚拟化。此外，随着虚拟机密度的增加，需要更多 CPU 资源来维持高性能。

- 硬件辅助解决方案 （例如，VFIO [48],SR-IOV[44], SIOV [22], FVM [28], 和 LeapIO [30]）在硬件层对 NVMe 设备进行虚拟化，具有高性能且无需占用 CPU 的优点。然而，这些解决方案缺乏灵活性，或需要开发额外的专用硬件。具体而言，虽然支持 SR-IOV 或 SIOV 的 NVMe 设备能够直接实现设备虚拟化，但这种方法会增加 SSD 控制器的复杂性，而且仅少数 NVMe 固态硬盘支持。另一方面，为 NVMe 虚拟化开发专用硬件也是一种选择，例如 FVM [28] 和 LeapIO [30], , 但额外的硬件会导致硬件成本和功耗增加，也使部署更加困难。

本文提出了 LightIOV，这是一种新颖的基于软件的 NVMe 虚拟化机制，无需消耗宝贵的 CPU 资源或专用硬件支持，即可实现高性能和高可扩展性。

LightIOV 的核心思想是 NVMe I/O 队列直通，使虚拟机能够直接访问 NVMe 设备的 I/O 队列。具体而言，LightIOV 由三个组件构成：LightIOV 前端驱动程序、LightIOV 后端驱动程序和 LightIOV 设备。LightIOV 前端驱动程序负责向客户机操作系统呈现标准的 NVMe 设备，无需修改客户机应用程序或其他操作系统组件。LightIOV 后端驱动程序创建 I/O 队列，并将 I/O 队列缓冲区映射到虚拟机，使虚拟机能够直接访问 I/O 队列。此外，后端驱动程序利用 IOMMU 支持，使客户机虚拟机能够在无需主机软件参与的情况下执行 DMA 事务和中断处理。最后，虚拟机监控器中的 LightIOV 设备模拟控制资源（PCIe 配置、BAR 空间和管理队列），并将其与 LightIOV 后端驱动程序分配的数据资源（NVMe I/O 队列、门铃寄存器、中断资源和逻辑块地址）相结合。

我们评估了 LightIOV 原型，并将其与现有 NVMe 虚拟化机制进行比较，包括硬件辅助的 VFIO [48],和基于软件的 SPDK-Vhost-NVMe [51], virtio [43]。结果表明，LightIOV 可提供接近原生的 NVMe 性能，其每秒输入输出操作数达到 VFIO 的 97.6%–100.2%。此外，在高密度虚拟机环境中，运行 200 台虚拟机时，LightIOV 的延迟比 SPDK-Vhost 低 31.4%，并且在实际应用中，OPS 性能提升了 27.1%。总之，我们的贡献如下：

- 我们提出了 LightIOV，这是一种新颖的基于软件的 NVMe 虚拟化机制，在不消耗宝贵 CPU 资源且不需要特殊硬件支持的情况下，实现了接近原生的性能和高可扩展性。
- 我们设计并实现了 NVMe I/O队列直通，使虚拟机能够直接访问 NVMe 设备的 I/O 队列。
- 我们提出了一种全新的、可高度扩展的 NVMe 虚拟化机制，通过软件为数千台虚拟机模拟完整的 NVMe 设备。
- 我们实现了 LightIOV，并从 I/O 性能、可扩展性和公平性等方面，对 LightIOV 与其他虚拟化方案进行了大量比较评估。合成基准测试和实际应用的结果均表明，LightIOV 优于其他方案。

2 背景

本节介绍 NVMe 和虚拟化方案的背景。

2.1 非易失性内存高速接口

非易失性内存（NVM）技术推动了打造更加强大的存储设备这一不可阻挡的趋势。

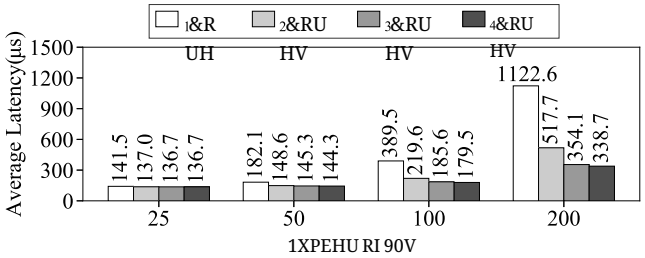


图 1. SPDK vhost-NVMe 在使用不同数量的 CPU 核心进行轮询时的平均延迟。随着虚拟机密度增加，SPDK vhost-NVMe 需要更多宝贵的 CPU 资源来维持高性能。

然而，由于串行 ATA (SATA) [15] 接口中高级主机控制器接口 (AHCI) [15] 架构设计的限制，高性能存储设备仍未得到充分利用。因此，为了解决输入/输出性能瓶颈，非易失性内存高速接口(NVMe) [18]应运而生，旨在挖掘快速非易失性内存存储的潜力，并利用大量输入/输出队列深度优化输入/输出路径。

NVMe 规范允许一个管理队列对关联多达 **65,535 个 I/O 队列对**，管理队列对负责执行影响整个控制器的功能，例如创建 I/O 队列对、命名空间管理和设置功能特性。另一方面，I/O 队列对专用于执行 I/O 操作（读/写）。每个队列对（QP）由一个提交队列（SQ）和一个完成队列（CQ）组成，前者负责提交命令，后者负责接收这些命令的完成通知。SQ 和 CQ 都是主机内存中的环形缓冲区，通过直接内存访问（DMA）与设备共享。每个队列（SQ 或 CQ）都会在其门铃寄存器（DB）中记录环形缓冲区的头指针或尾指针。**控制器内存缓冲区（CMB）**是 NVMe 设备控制器上的通用读写内存区域，可用于多种用途。例如，将提交队列存储在 CMB 中，可使 NVMe 驱动程序将整个 SQ 条目直接写入控制器的内部内存空间，避免从主机内存中获取该条目。

NVMe 命令的处理过程如下：NVMe 驱动程序将 NVMe 命令放入 SQ，并敲响目标 SQ 门铃寄存器，指示有新命令生成。随后，NVMe 设备获取并处理新添加的命令。之后，NVMe 设备通过 DMA 从主机读取数据或向主机写入数据。NVMe 命令完成后，NVMe 设备将完成消息写入关联的 CQ，并生成中断。最后，NVMe 驱动程序处理 CQ 条目，并更新目标 CQ 门铃寄存器，以清除中断并释放 CQ 条目。

2.2 NVMe 虚拟化

*1)*基于软件的虚拟化。基于软件的存储虚拟化机制可分为三类：全虚拟化、半虚拟化和基于轮询的虚拟化。

在全虚拟化中，虚拟机监控器通过软件按照硬件规范模拟硬件设备的功能。这种方式对虚拟机是透明的，且不需要专用硬件支持。然而，由于性能下降严重，全虚拟化已逐渐淡出视野。

半虚拟化是为解决全虚拟化的性能问题而提出的方案，而 virtio [43]是半虚拟化驱动程序规范的事实标准。virtio 驱动程序由前端和后端两部分组成，即客户机操作系统内核中的 virtio 前端驱动程序，以及虚拟机监控器中的 virtio 后端驱动程序。这两部分通过共享内存中的虚拟 I/O 队列相互协商。

由于 virtio 后端驱动程序效率低下，强大的 NVMe 设备无法得到充分利用 [51]。基于轮询的方法[40, 50, 51] 使用专用 CPU 核心来实现高性能。在所有 SPDK vhost-target 解决方案 [10] （包括 SPDK vhost-scsi、SPDK vhost-blk、SPDK vhost-NVMe）中，针对 NVMe 固态硬盘进行了优化的 SPDK vhost-NVMe 在 NVMe 设备上性能最佳 [51]。SPDK vhost-NVMe 解决方案的工作流程如下：（1）在专用 CPU 核心上运行一个线程，轮询共享内存区域中的虚拟 NVMe I/O 队列。（2）将新请求转换为 NVMe 命令。（3）随后通过 SPDK 用户级 NVMe 驱动程序将请求发送到物理设备。（4）NVMe 设备从客户机虚拟机读取数据或向其写入数据。（5）请求完成后，线程处理请求完成信息，并通过虚拟机监控器向虚拟机注入中断。

同样，Mdev-NVMe [40] 利用中介式直通机制，并在内核中为影子 SQ、完成队列和主机完成队列设计主动轮询机制，以获得高性能。

*2)*硬件辅助虚拟化。 直接设备分配（或 VFIO）使虚拟机能够直接使用 NVMe设备。它利用 I/O 内存管理单元（IOMMU）对 DMA和中断重映射的支持（例如 Intel-VTd[23],AMD-Vi [8]），从而消除了与软件虚拟化相关的开销，并实现接近原生的性能。然而，VFIO 要求将物理设备专门分配给单个虚拟机，因此牺牲了虚拟化的资源共享特性。

为解决 VFIO 缺乏可共享性的问题，提出了 PCIe 单根 I/O 虚拟化（PCIe SR-IOV）[44] ，使多个虚拟机能够在硬件层共享同一物理设备。支持 SR-IOV 的存储设备由一个物理功能（PF）和多个虚拟功能（VF）组成。物理功能由主机软件管理，负责设备的资源管理；每个虚拟功能都关联到一个物理功能，并可分配给一个虚拟机。支持 SR-IOV 的设备既能实现接近原生的磁盘性能，又能支持设备共享。FVM[28],是一种硬件辅助存储虚拟化机制，它在 FPGA 卡上实现 SR-IOV 层，在支持各种虚拟机功能特性的同时，实现高性能和可扩展性。此外，LeapIO [30] 将整个存储栈卸载到 ARM SoC，以减轻主机 CPU 的负担。

可扩展IOV（SIOV） [22] 是一种硬件辅助的输入/输出虚拟化规范，专注于高效且可扩展地共享输入/输出设备。具体而言，SIOV允许更频繁且对性能至关重要的操作直接在硬件上运行，而复杂的控制和配置操作则通过软件进行仿真。

3 动机

本节讨论数据中心现有 NVMe 虚拟化机制面临的挑战。我们发现，基于软件的解决方案要么性能下降，要么 CPU 开销过高。与此同时，基于硬件辅助的解决方案则存在灵活性不足的问题，且需要专用硬件支持。

3.1 现有基于软件的虚拟化的缺点

性能下降。 全虚拟化和半虚拟化都会遭遇显著的性能下降。例如，传统的半虚拟化解决方案 virtio 只能达到原生性能的 50% [40]。这种性能下降可归因于两个因素。首先，当虚拟机向提交队列分发请求时，会产生开销高昂的 VM_Exit 事件；而虚拟机监控器则会向虚拟机发送虚拟中断。这些事件会导致客户机挂起、主机恢复运行，并因上下文切换开销和缓存污染[3, 13, 14, 29]而造成性能下降。其次，对于 NVMe 设备与虚拟机之间的输入/输出操作，虚拟机监控器至少需要进行两次数据搬移，这会降低输入/输出性能[21]。

CPU 开销高。 尽管能够提供接近原生的性 near-nativeper- 能，基于高级轮询的NVMe 虚拟化（例如 SPDKvhost-target [51],Mdev-NVMe [40]）会产生较高的 CPU 开销。在对 SPDK vhost-NVMe 解决方案的测试中，我们发现，要达到2 个Intel P4510 固态硬盘的每秒输入输出操作数上限，需要 1个CPU 核心。此外，云服务提供商通常会提供具有热升级和在线迁移等功能特性的存储服务，这会增加软件复杂性并需要更多 CPU资源。因此，在生产环境中，在配备12 个固态硬盘的服务器上虚拟化 NVMe 存储通常会消耗 8-10 个 CPU 核心。由于服务器是

通常配置有96-128 个CPU 核心，存储虚拟化开销最高可占用 10.4% 的 CPU 资源。

此外，在虚拟机未产生I/O 工作负载的期间，基于轮询的解决方案会浪费宝贵的 CPU 资源。在基于轮询的解决方案中，专用 CPU 核心必须持续轮询相关内存区域，以处理虚拟 I/O 请求。因此，即使不存在新的 I/O请求，100% 的 CPU 周期仍会被消耗。这种对 CPU 资源的低效使用可能导致系统性能降低和能耗增加。

此外，基于轮询的解决方案还存在另一个固有缺陷：随着虚拟机密度增加，这些解决方案需要更多 CPU 资源来维持高性能。图1 展示了在两块固态硬盘上使用 4k-随机读-1-1 工作负载运行 25、50、100 和 200 台虚拟机时，为 SPDK-Vhost 轮询分配不同数量 CPU 核心所产生的影响。结果表明，运行 25台虚拟机时，使用不同数量的 CPU核心进行轮询，性能没有显著差异。然而，运行 200 台虚拟机时，使用两个核心进行轮询的输入/输出延迟比仅使用一个核心低 53.9%。因此，随着虚拟机密度增加，基于轮询的解决方案需要更多宝贵的 CPU 资源，才能维持其高性能优势。否则，它们可能导致显著的性能下降。

3.2 现有硬件辅助虚拟化的局限性

缺乏灵活性。 硬件辅助机制需要专门的硬件特性，这限制了其灵活性。支持 SR-IOV 或 SIOV 的 NVMe 固态硬盘可通过 SSD 控制器直接虚拟化这些设备。然而，只有数量有限的固态硬盘支持 SR-IOV 功能特性，目前还没有 NVMe 固态硬盘支持 SIOV。此外，SIOV 不仅要求固态硬盘具备 SIOV 能力，还需要配备新兴功能的最新服务器 CPU（例如，通过 DMA 重映射硬件实现 PASID 粒度的地址转换）。虽然可以将这些虚拟化能力标准化，以便所有 NVMe 固态硬盘都能支持，但这会增加 SSD 控制器的复杂性，并导致不同制造商生产的 NVMe 设备之间出现兼容性问题。

专用硬件的总拥有成本挑战。 使用专用硬件是实现 NVMe 虚拟化的另一种方法，例如 FVM [28], 和 LeapIO [30]。然而，专用硬件在总成本方面也面临自身的挑战

总拥有成本 (TCO)。FVM 必须基于 FPGA 开发新硬件，这会增加硬件成本和功耗。LeapIO 将整个存储栈卸载到 ARM SoC，并同时支持本地和远程存储。然而，由于 ARM CPU 的计算能力有限，其性能下降严重，吞吐量仅达到单块原生磁盘的 68% [28] 。

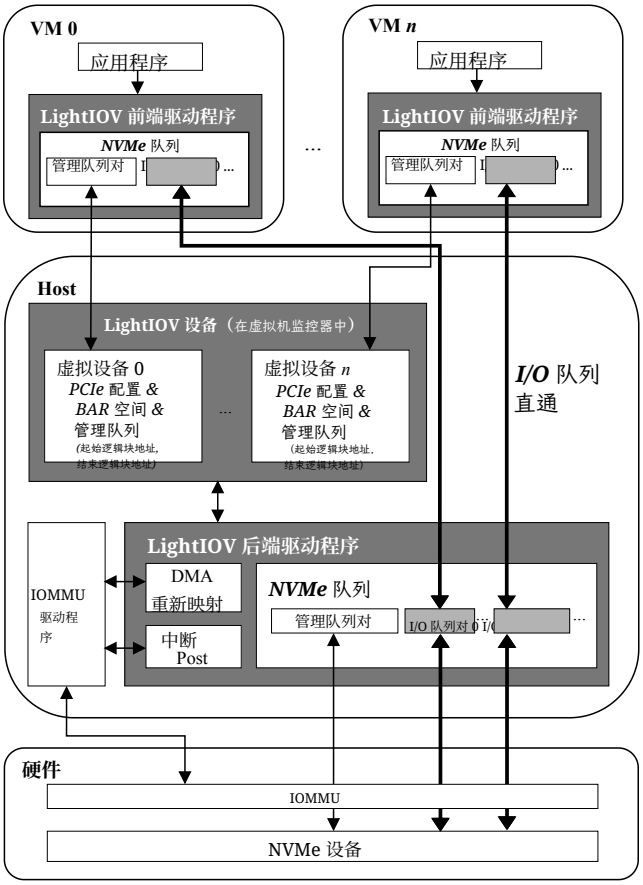


图 2。LightIOV 的整体架构

3.3 总结

如表 1所总结，现有基于软件的 NVMe 虚拟化方案存在显著的性能下降或较高的 CPU 开销，而硬件辅助解决方案缺乏灵活性，并面临总拥有成本方面的挑战。这些挑战促使我们提出 LightIOV。该方案消除了虚拟化开销，实现了接近原生的性能，并且不需要专用硬件支持。

4 设计与实现

本节介绍 LightIOV 的设计及其实现。

4.1 Design Goals

我们审查了 NVMe 虚拟化领域最先进的解决方案，发现现有的基于软件的解决方案和硬件辅助解决方案各有优缺点。基于软件的解决方案不依赖硬件，因此具有很高的灵活性；而硬件辅助解决方案在硬件层对 NVMe 设备进行虚拟化，使虚拟机能够高效地与物理设备交互，同时不消耗主机的 CPU 资源。为此，我们为 LightIOV 制定了以下设计目标:

高性能。该机制应提供与原生 NVMe 固态硬盘相当的吞吐量和延迟性能。

低开销。它不应占用宝贵的 CPU 资源，因为云计算环境中的 CPU 核心数量有限且十分宝贵。通过节省 CPU 资源，可以运行更多应用，这对于希望降低总拥有成本的云服务提供商而言颇具吸引力。

灵活性。无论 NVMe 设备是否具备 SR-IOV 等虚拟化能力，该机制都应适用于所有 NVMe 设备，并且不应要求额外的专用硬件来实现虚拟化。

高可扩展性。随着单台服务器上虚拟机密度不断增加，NVMe 虚拟化机制应能够良好地扩展，因为虚拟机密度正在持续提升，且 Firecracker 等轻量级虚拟机 [5] 和 RunD [32] 正在云数据中心中得到广泛部署。该机制应能够支持数千台虚拟机，并保持较高的性能。

4.2 总体架构

为实现上述设计目标，我们提出了 Light-IOV，这是一种新颖的基于软件的 NVMe 虚拟化机制，兼具基于软件的解决方案（灵活性）和硬件辅助解决方案（高性能和低 CPU 开销）的优势。

LightIOV 将 NVMe 设备资源划分为**控制资源**（PCIe 配置、BAR 空间和管理队列）和**数据资源**（输入/输出队列、门铃寄存器、中断资源和逻辑块地址）。由于 NVMe 设备只有一组控制资源，而每个模拟设备都必须拥有独立的控制资源，因此 LightIOV 必须在虚拟机监控器中虚拟化控制资源，以供共享。控制资源用于设备初始化、查询和管理，不参与输入/输出请求。因此，使用软件模拟控制资源不会影响输入/输出性能。根据 NVMe 规范，NVMe 设备最多可支持 65,535 个输入/输出队列 [18], ，因此 LightIOV 无需虚拟化这些资源，输入/输出队列可以直接分配给不同的虚拟机。

图2展示了 LightIOV 的架构及其组件。LightIOV 由三个组件构成：主机内核中的 LightIOV 后端驱动程序、客户机虚拟机中的 LightIOV 前端驱动程序，以及虚拟机监控器中的 LightIOV 设备。

LightIOV 后端驱动程序 负责使虚拟机能够直接访问输入/输出队列，并进行 DMA 重映射——
门铃和中断投递。它创建 I/O 队列，并通过虚拟设备的控制器内存缓冲区 (CMB) 将 I/O 队列缓冲区映射到虚拟机，使虚拟机可以直接访问 I/O 队列。DMA 重映射和中断投递组件无需虚拟机监控器介入，即可为客户机虚拟机处理 DMA 事务和中断。

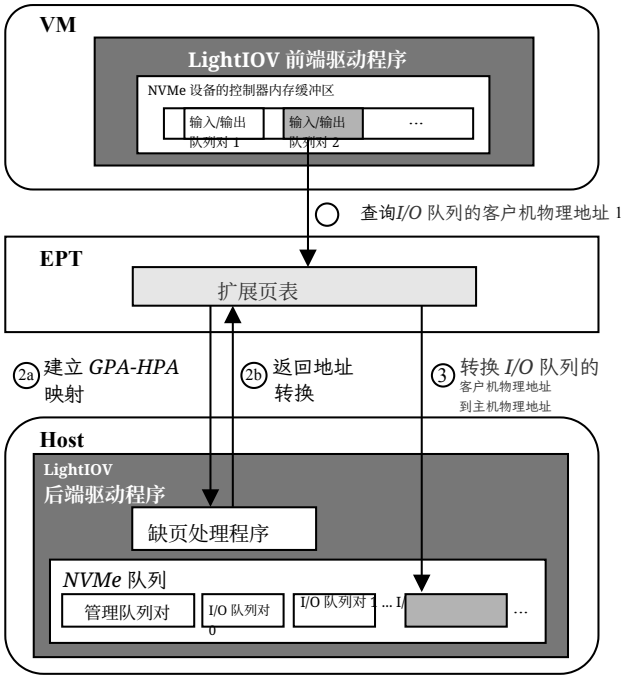


图 3。I/O 队列直通机制

LightIOV 设备虚拟化 NVMe 控制资源，并为虚拟机模拟完整的 NVMe 设备。每个虚拟 NVMe 设备都包含控制资源（PCIe 配置、BAR 空间和管理队列）以及数据资源（I/O 队列、门铃寄存器、中断资源和逻辑块地址）。数据资源由 LightIOV 后端驱动程序分配。LightIOV 设备管理虚拟 NVMe 的逻辑块寻址 (LBA) 资源。

LightIOV前端驱动程序 为客户机虚拟机提供标准 NVMe 设备，因此用户应用程序或其他操作系统组件无需进行任何修改。LightIOV前端驱动程序直接利用 LightIOV 设备提供的控制器内存缓冲区中的输入/输出队列来创建 NVMeI/O队列。由于 NVMe I/O 队列直通，LightIOV 前端驱动程序还负责将虚拟LBA转换为 NVMe 设备的物理 LBA，此过程只需添加偏移量即可。

4.3 I/O 队列直通的工作原理

NVMe 的控制器内存缓冲区（CMB）是 NVMe 设备内部的一种通用内存，可用于多种用途。I/O 队列直通的原理是，LightIOV 后端驱动程序利用 CMB 在主机与客户机虚拟机之间映射 I/O 队列内存；这通过已注册的缺页处理程序在扩展页表（EPT）中建立客户机 I/O 队列的 GPA-HPA 映射来实现。这样，虚拟机便可通过 EPT 地址转换直接向主机 NVMe I/O 队列写入 I/O 请求。图中展示了 I/O 队列直通的工作方式 3。

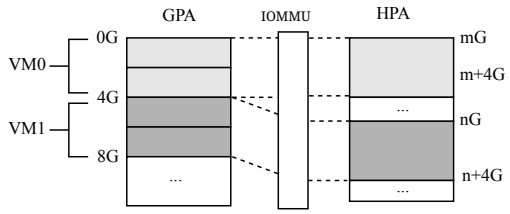


图 4. IOMMU 中的 DMA 地址隔离

◯1 虚拟机中的 LightIOV 前端驱动程序将 I/O 请求提交到 CMB 中的 I/O SQ，内存管理单元（MMU）随后将 I/O 队列的客户机虚拟地址（GVA）转换为客户机物理地址（GPA）。接着，扩展页表（EPT）接管 MMU 提供的 GPA，并在 EPT 中查找该 I/O 队列 GPA 对应的主机物理地址（HPA）。如果该 GPA 未映射到任何主机内存，就会触发缺页中断，随后转到 2；否则，转到 3 。

◯2 LightIOV 实现了自己的缺页处理程序。缺页处理程序获取相应 I/O 队列对的主机物理地址（HPA），并在扩展页表（EPT）中建立客户机物理地址—主机物理地址（GPA-HPA）映射（2a）。地址映射完成后，它会返回到扩展页表（EPT）中的 HPA-GPA 地址转换（2b）。门铃寄存器的 GPA-HPA 映射建立过程与此类似。每个 I/O 队列只会调用一次缺页处理程序。I/O 队列页表建立后，该队列上的 I/O 请求将不会触发缺页，因此不会影响 I/O 性能。

◯3 扩展页表（EPT）根据已建立的页表项，将客户机 I/O 队列的客户机物理地址（GPA）转换为主机物理地址（HPA）。因此，虚拟机可以访问主机内存中的 I/O SQ 以提交 I/O 命令，该命令随后由 NVMe 设备获取并执行。

因此，一旦在扩展页表（EPT）中建立 I/O 队列和门铃寄存器的页表项，虚拟机就可以像主机一样直接访问 NVMe I/O 队列。鉴于 NVMe 固态硬盘能够持续支持每秒数十万次 I/O 操作，由这些 I/O 请求触发的缺页事件所产生的开销可以忽略不计。此外，现有服务器通常都支持扩展页表（EPT），LightIOV 巧妙地利用这一现有地址转换机制实现了 I/O 队列直通。

4.4 DMA 重映射与中断投递

DMA 数据传输和中断处理对于高性能 NVMe 虚拟化机制至关重要。借助 IOMMU 对 DMA 和中断重映射的支持，Light- IOV 使虚拟机能够通过 DMA 与 NVMe 设备传输数据，并为虚拟机处理中断，而无需经历代价高昂的 VM_Exit。

DMA 重映射。客户机 I/O 请求中的 DMA 地址是客户机物理地址（客户机物理地址）。客户机物理地址必须转换为主机物理地址，供 NVMe 使用。

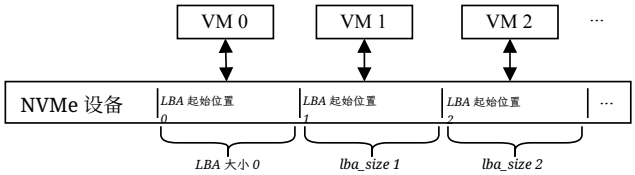


图 5. 通过将不同的逻辑块地址范围分配给虚拟机来实现 NVMe 设备存储的隔离

发起 DMA 事务的设备，并且 IOMMU 会自动执行 GPA-HPA 地址转换。然而，由于IOMMU使用PCIe 设备的 BDF（总线、设备、功能）来区分页表项，访问同一PCIe 设备的不同虚拟机会共享 IOMMU页表项。来自不同虚拟机的 GPA可能相同，因此可能被转换为相同的 HPA，从而由于 GPA 冲突导致错误的DMA 事务。为了保证不同虚拟机的 DMA 地址隔离，虚拟机监控器会在启动时为虚拟机设置不重叠的客户物理地址。例如，如图 4所示，虚拟机监控器为 VM0分配从0G 到 4G 的客户物理地址，为 VM1分配从 4G 到 8G 的客户物理地址。因此，VM0 和VM1的输入/输出命令中的 DMA 地址经过 IOMMU转换后会映射到不同的 HPA。

中断投递。LightIOV利用 IOMMU中断重映射单元，并将其与中断后置相结合。中断后置是一种硬件机制，允许虚拟机直接接收中断。LightIOV后端驱动程序使用 *IRQ* 旁路管理器 [47] 配置中断后置 I/O 队列。

这样，当 I/O请求的中断到达IOMMU 时，IOMMU硬件会查询相关的中断重映射表项，并将物理中断转换为虚拟机的虚拟中断。随后，虚拟中断被注入虚拟机，并由虚拟机注册的中断处理程序进行处理。整个中断处理过程无需主机软件参与，也无需虚拟机退出，从而能够有效提高虚拟机处理中断的效率，进而提升 I/O 性能。

4.5 虚拟设备仿真

LightIOV 在虚拟机监控器中为虚拟机呈现完整的 NV Me 设备，通过软件仿真控制资源，并将其与由 LightIOV 后端驱动程序分配的数据资源结合起来。控制资源包括 PCIe 配置、BAR 空间和管理队列，它们对于 NVMe 设备至关重要。LightIOV 使用传统的陷入并仿真方法在虚拟机监控器中仿真控制资源。当虚拟机访问虚拟设备的控制资源时，会触发 VM_Exit 事件并陷入主机，随后由虚拟机监控器接管虚拟机请求。然后，虚拟机监控器读取/更新相关虚拟寄存器。最后，虚拟机监控器向虚拟机生成中断，以通知请求已完成。通常，只有管理

表2。 测试用例

用例名称	(bs,rw,numjobs,iodepth)
随机读-4k-4-128	(4k,随机读, 4,128)
randwrite-4k-4-128	(4k,随机写, 4,128)
顺序读-128k-4-128	(128k,读,4,128)
顺序写-128k-4-128	(128k,写,4,128)
randread-4k-1-1	(4k,随机读,1,1)
randwrite-4k-1-1	(4k,randwrite,1,1)
seqread-4k-1-1	(4k,read,1,1)
seqwrite-4k-1-1	(4k,write,1,1)

用于设备初始化、管理和状态查询的命令会访问控制资源。相比之下，用于读取和写入数据的 I/O请求（这对于确定I/O 性能至关重要）不涉及控制资源。因此，对控制资源使用陷入并仿真方法不会影响 I/O 性能。

LightIOV 设备将控制资源（PCIe配置、BAR 空间和管理队列）与数据资源（I/O 队列、门铃寄存器、中断资源以及逻辑块地址）相结合，为虚拟机呈现完整的 NVMe 设备。LightIOV 能够提供的虚拟化 NVMe 设备数量受固态硬盘中实现的 NVMe I/O 队列数量限制。

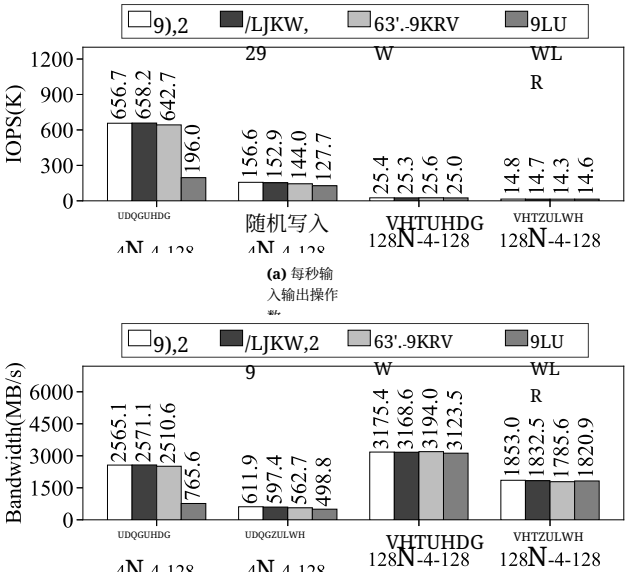
根据 NVMe 规范 [18], 一台 NVMe 设备的 I/O队列最大数量为 65,535，因此 LightIOV可以轻松扩展至**数万台虚拟机**。

4.6 隔离

由于队列映射完成后，虚拟机可以直接向物理设备发送 I/O 请求，因此检查 I/O 请求中目标地址的有效性对于防止虚拟机之间发生未授权访问至关重要。

LBA隔离。NVMe设备利用逻辑块地址（LBA）对其内部空间进行寻址。为确保每个虚拟机只能访问其指定的磁盘区域，LightIOV通过 LightIOV 前端驱动程序和后端驱动程序的协作，实现了 LBA地址隔离。具体而言，如图5所示，LightIOV 后端驱动程序为每个虚拟机分配不重叠的逻辑块地址范围，并将相应的 `lba_start` 和`lba_size` 信息存储在虚拟设备的控制器内存缓冲区（CMB）中。当虚拟机通过`nvme_submit_cmd`提交 I/O 命令时，LightIOV 前端驱动程序会将`lba_start`加到`slba` I/O命令的起始逻辑块地址（SLBA）中，并检查所得的逻辑块地址是否在虚拟机可访问的范围内。如果超出该范围，则会发生 I/O 错误，且无法发送请求。

客户物理地址隔离。 第4.4 节说明了虚拟机监控器如何为每个虚拟机分配互不重叠的客户物理地址，以确保每个虚拟机在IOMMU转换后访问自己的主机物理地址。为了进一步防止恶意虚拟机读取



(b) 带宽

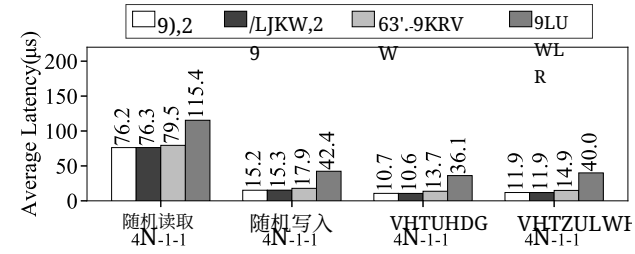
图 6. 使用一个 SS 时的单虚拟机吞吐量 D

并写入其他虚拟机的内存（例如，如果虚拟机0的 GPA范围为0-4G，但恶意虚拟机 0 发出的地址空间的 GPA 为 6G ），LightIOV前端驱动程序会对 I/O 命令的目标内存地址（GPA）执行合法性检查。具体而言，LightIOV 前端驱动程序会验证客户机发送的 NVMe 请求中的 PRP 地址是否位于该虚拟机的 GPA 地址范围内。如果不在，LightIOV 将生成 I/O 错误，并且无法将该请求发送到 NVMe 设备。

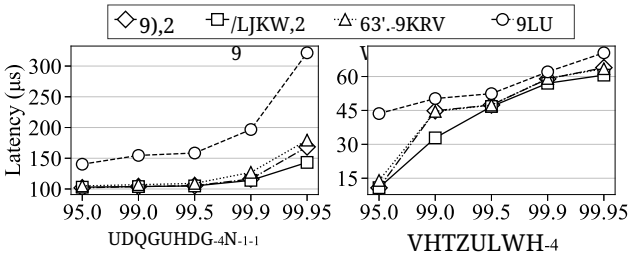
4.7 实现

我们在 Linux 内核（版本 4.19）和 Firecracker（版本 0.14.0）上实现了 LightIOV。Firecracker [5] 是一种新型开源虚拟机监视器（VMM），也称为虚拟机监控器。LightIOV 的实现总计包含 6,600 行代码，其中客户机内核包含 300 行，主机内核包含 4,200 行，Firecracker 包含 2,100 行。

对于客户机内核，我们将原始 NVMe 驱动程序修改为 LightIOV 前端驱动程序，该驱动程序直接利用控制器内存缓冲区中的 I/O 队列，并为客户机 NVMe 命令的起始逻辑块地址添加偏移量，从而实现虚拟机之间的存储隔离。在主机内核中，我们基于原始 NVMe 驱动程序和 VFIO-mdev 框架实现了 LightIOV 后端驱动程序 [37]。LightIOV 后端驱动程序在主机上创建 NVMe I/O 队列。然后，它通过 VFIO-mdev 框架将这些 I/O 队列以及门铃寄存器和 LBA 范围暴露给虚拟机监控器。在 Firecracker 中，我们添加了对 VFIO 设备的支持，并采用 VFIO-mdev 框架 [37]，为 LightIOV 设备提供 NVMe 数据资源。Firecracker 还实现了对 PCIe 配置空间、BAR 空间和 NVMe 管理队列的仿真。

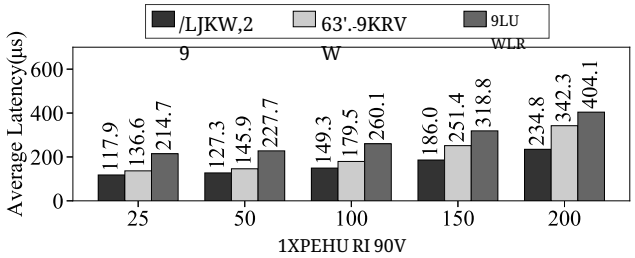


(a) 平均延迟

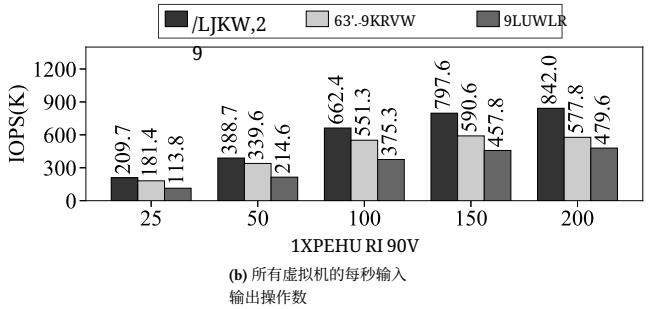


(b)尾延迟

图 7. 使用一块固态硬盘时单个虚拟机的延迟



(a)所有虚拟机的平均延迟



(b) 所有虚拟机的每秒输入输出操作数

图 8. 两个固态硬盘上运行 4k-randread-1-1 的多个虚拟机的 I/O 性能

5 评估

我们评估了 LightIOV，并将其 I/O 性能和可扩展性与其他 NVMe 虚拟化机制解决方案（包括 VFIO、SPDK-Vhost 和 virtio）进行了比较。我们还评估了 LightIOV 在多个虚拟机上的公平性，并在真实应用程序上运行了各种合成基准测试。

5.1 实验设置

系统设置。我们在一台配备两颗 48 核 Intel Xeon Platinum 8163 CPU 的主机上部署了 LightIOV。为了避免跨非一致性内存访问（NUMA）访问设备导致测试结果出现不可预测的波动， [2], 我们仅使用一个 NUMA 节点。我们使用 Intel SSD P4510 2T [17] 作为 NVMe 设备。在实现方面，我们在主机中安装了 LightIOV 后端驱动程序和 LightIOV 设备，并在客户机中安装了 LightIOV 前端驱动程序。主机和客户机操作系统均运行于 64 位 CentOS，使用 Linux 内核（版本 4.19）。此外，我们使用经过修改的 Firecracker（版本 0.14.0）作为虚拟机监控器来管理虚拟机。

比较虚拟化解决方案。 我们将 LightIOV 与具有代表性的硬件辅助和软件解决方案进行了比较。VFIO 的性能接近原生磁盘，其性能在硬件辅助解决方案中表现最佳，因此我们将其作为基准。对于软件解决方案，我们将 LightIOV 与 SPDK-Vhost 和 virtio 进行了比较。SPDK-Vhost 指的是 SPDK vhost-NVMe，它在所有 SPDK vhost-target 解决方案中性能最佳。

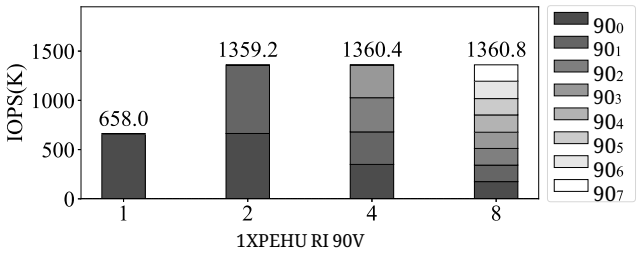
基准测试和真实应用。 我们使用 fio [1] 作为合成评估基准工具，通过指定特定的输入/输出模式（包括随机读写、顺序读取）对磁盘进行压力测试。等），以及不同的输入/输出压力（不同的线程数和队列深度）。具体而言，我们将 libaio 作为 fio引擎，并运行了表 2中所示的各种测试用例。通过将 direct参数设置为 1，我们绕过了客户机操作系统内核的页面缓存。此外，RocksDB [41], 这一种用于闪存和 RAM 存储的持久化键值存储，被用于测量各种虚拟化方案在真实应用程序中的性能。

5.2 I/O 性能

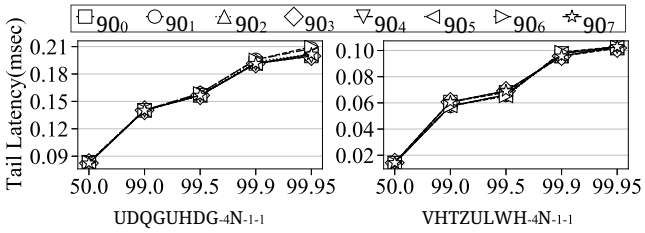
我们评估了 VFIO、LightIOV、SPDK-Vhost 和 virtio 使用一个固态硬盘时的 I/O 性能。我们在单个虚拟机上运行了表2 中所示的各种测试用例。该虚拟机分配有 4 个 CPU 核心和 4GB 系统内存。对于 SPDK-Vhost，我们还为 SPDK vhost-target 分配了一个额外的 CPU 核心。

吞吐量。 图 6 展示了单个虚拟机的吞吐量表现。就每秒输入输出操作数和带宽而言，LightIOV 可提供接近原生性能的表现。

在 randread-4k-4-128 和 randwrite-4k-4-128 测试中，LightIOV 的每秒输入输出操作数分别达到 VFIO 的 100.2% 和 97.6%。VFIO 将整个物理设备直通给虚拟机，虽能实现接近原生性能，但会牺牲设备共享能力。相比之下，LightIOV 不仅能确保接近原生性能，还能实现多个虚拟机之间的设备共享。与 SPDK 相比，LightIOV 在 randwrite-4k-4-128 测试中的每秒输入输出操作数高出 6.2%。SPDK-Vhost 通过占用宝贵的 CPU 核心进行轮询，取得了接近 LightIOV 的性能。virtio 则遭受严重的性能下降，虚拟化 I/O 性能最差。在 randread-4k-4-128 测试中，LightIOV 的每秒输入输出操作数达到 virtio 的 335.8%。



(a) 运行在 2 块固态硬盘上的多个虚拟机的总体每秒输入输出操作数



(b)八个虚拟机的尾延迟分布

图9. LightIOV的公平性

在 128K顺序读写测试用例中，所有方案均达到存储设备的带宽上限。LightIOV 的平均性能达到 VFIO 的 99.4%、SPDK 的 100.9% 和 virtio 的 101%。处理大块请求时，它们之间的性能差异并不显著。

延迟。 图 7a描绘了单线程和单队列深度I/O工作负载的平均延迟，且随机读-4k-1-1和seqwrite-4k-1-1工作负载的尾部延迟如图 7b所示。

如图7a所示，LightIOV 与 VFIO 的平均延迟差异不足 1%，可以忽略不计。这表明 LightIOV 能够实现与 VFIO 相同的低延迟。在 seqread-4k-1-1 中，SPDK 的延迟比 LightIOV 高出最多 29.4%，这是因为 SPDK-Vhost 的 I/O路径比 LightIOV 更长。在 LightIOV 中，虚拟机直接向设备发送 I/O 请求，无需额外的软件处理。因此，LightIOV 的 I/O 延迟低于 SPDK。由于虚拟化开销严重，virtio 在延迟方面表现不佳，其延迟比 LightIOV 高出 51.2%–236.1%。此外，从图7b 可以看出，LightIOV 的尾延迟表现最佳，其次是 VFIO 和 SPDK-Vhost，最后是 virtio。

总而言之，LightIOV 在吞吐量和延迟方面均能提供高性能。其性能与 VFIO 相当，略优于 SPDK-Vhost，且远优于 virtio。

5.3 可扩展性

在可扩展性测试中，我们在 2 块固态硬盘上运行了 25 至 200 个虚拟机，并评估了 LightIOV、SPDK-Vhost 和 virtio。VFIO 会将设备直接分配给一个虚拟机，无法实现多个虚拟机之间的设备共享，因此未将其纳入

本小节。我们为 SPDK- Vhost 轮询分配了 4 个 CPU 核心，因为如图所示，4 个核心足以支持两块固态硬盘上的 200 个虚拟机 1。我们每 5 个虚拟机分配一个核心。服务器配置了两块 Intel P4510 固态硬盘，每个虚拟机拥有 10G 容量。我们在这些虚拟机上运行了 4k-randread-1-1，并测量了所有虚拟机的平均延迟和总体吞吐量。

图 8a展示了不同虚拟机密度下 LightIOV、SPDK-Vhost 和 virtio 的延迟性能。值得注意的是，在所有虚拟机密度下，LightIOV 均表现出最低的延迟。随着虚拟机密度增加，LightIOV 相对于 SPDK-Vhost 的性能优势变得更加显著，尽管 SPDK-Vhost 比 LightIOV 多消耗四个 CPU 核心。随着虚拟机数量从 25 个增加到 200 个，LightIOV 的延迟比 SPDK-Vhost 低 12.8% 至 31.4%，这可归因于 LightIOV 的 I/O 队列直通功能。除了优于 SPDK-Vhost 之外，LightIOV 相较于 virtio 也展现出显著的延迟优势。具体而言，在运行 25 台虚拟机时，LightIOV 的延迟比 virtio 低 45.1%。

图8b所示结果表明，在高密度虚拟机场景下，LightIOV 的整体每秒输入输出操作数性能优于 SPDK-Vhost 和 virtio。具体而言，在运行 200 台虚拟机时，LightIOV 的每秒输入输出操作数分别比 SPDK-Vhost 和 virtio 高 45.7% 和 75.6%。

总而言之，随着虚拟机数量增加，LightIOV 相比基于软件的 NVMe 虚拟化解决方案具有更出色的 I/O 性能，展现出出色的可扩展性。

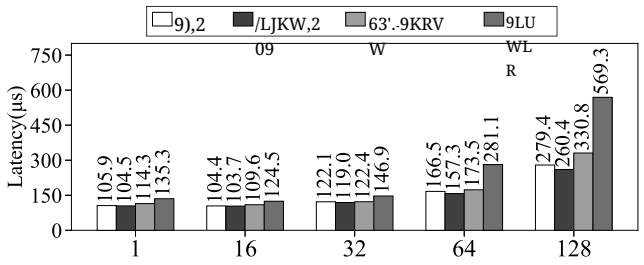
5.4 公平性

本节评估了使用 LightIOV 时多个虚拟机之间的公平性。我们为每个虚拟机分配了四个 CPU 核心，并运行了 4k-随机读-4-128 用例。

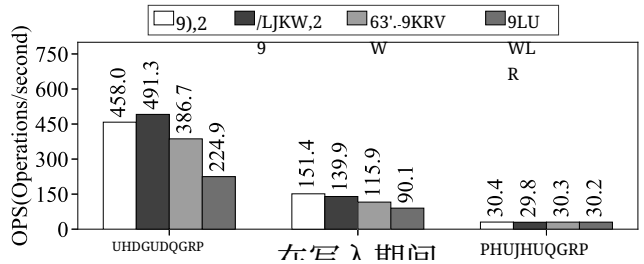
图9a 展示了多个虚拟机中配备 2 块固态硬盘时 LightIOV 的整体每秒输入输出操作数。在此实验中，我们使用了八个虚拟机，其中虚拟机 0/2/4/6 使用一块磁盘，而虚拟机 1/3/5/7 使用另一块磁盘。运行一个或两个虚拟机时，整体每秒输入输出操作数分别达到一块固态硬盘（658.0K 每秒输入输出操作数）和两块固态硬盘（1359.2K 每秒输入输出操作数）的上限。然而，当我们将虚拟机数量增加到 4 和 8 时，所有虚拟机的整体每秒输入输出操作数仍保持在 2 块固态硬盘的上限（约 1360K 每秒输入输出操作数）。此外，每个虚拟机的性能分配均衡，表明资源得到了高效利用，并在所有虚拟机之间公平分配。

此外，我们还考察了八台虚拟机的尾延迟分布，如图所示 9b。我们观察到，每台虚拟机的尾延迟分布都较为集中，这表明存储资源并未偏向某些虚拟机。这进一步说明，所有虚拟机获得的存储资源份额相同。

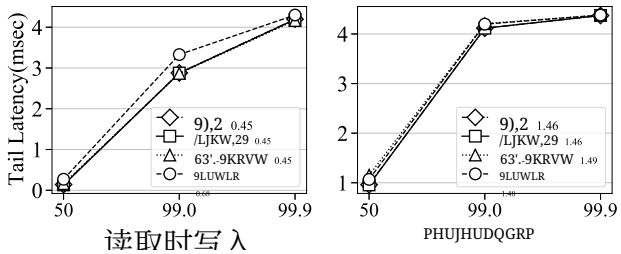
总之，随着虚拟机数量增加，LightIOV 仍能维持整体吞吐量，并保证每个虚拟机的公平性 。



(a) 并发线程数增加时单个虚拟机的延迟



(b) 使用 128 个线程运行不同测试用例时单个虚拟机的每秒操作数



(c)单个虚拟机使用 1 个线程时的尾延迟

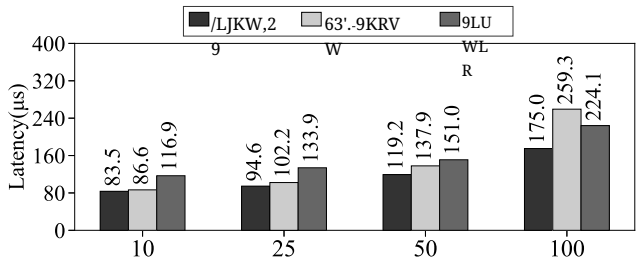
图 10.运行 RocksDB 工作负载的单个虚拟机

5.5 实际应用

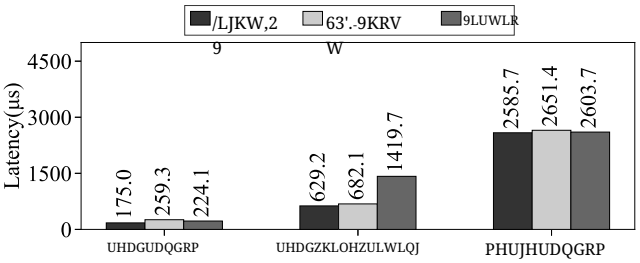
对于实际应用，我们评估了 RocksDB（一种用于闪存和 RAM 存储的持久化键值存储）在单虚拟机和多虚拟机场景下的性能，并使用 db_bench 发起 I/O 请求。在单虚拟机场景中，我们为每个虚拟机分配了 4 个 CPU 核心；在多虚拟机场景中，我们每 5 个虚拟机分配 1 个核心。具体而言，我们运行了“bulkload”用例，按顺序将 100 万条数据（键大小为 = 16 字节，值大小为 = 100 字节，总大小为 = 1106.3 MB）插入数据库。

单个虚拟机。 我们在逐步将 readrandom 测试用例中的线程数从 1 增加到 128 的过程中，对各种虚拟化方案进行了测试。图10a 表明，在平均延迟方面，LightIOV 优于 VFIO 和 SPDK-Vhost，并且显著优于 virtio。具体而言，在 128 个并发线程下，LightIOV 的平均延迟比 VFIO 低 6.8%，比 SPDK-Vhost 低 21.3%，其每秒操作数比 virtio 高 54.3%。

我们还在单个虚拟机上使用 128 个线程运行了多种工作负载，包括 readrandom、readwhilewriting 和 mergerandom。每秒操作数（OPS）如图所示



(a) 1 个线程运行 readrandom 时，随着虚拟机数量增加的平均延迟



(b) 运行各种工作负载的 100 台虚拟机的平均延迟，使用 1 线程

图11。运行 RocksDB 工作负载的多个虚拟机

图10b中给出了结果，数值越高表示性能越好。LightIOV 在所有测试用例中的性能都接近 VFIO。具体而言，在 readrandom 测试中，LightIOV 的每秒操作数达到 VFIO 的 107.3%。此外，在 readrandom 和 readwhilewriting 测试中，LightIOV 的表现优于 SPDK-Vhost 和 virtio。具体来说，LightIOV 在 readrandom 和 readwhilewriting 测试中的每秒操作数分别比 SPDK-Vhost 高 27.1% 和 20.7%，比 virtio 高 118.5% 和 55.2%。

图 10c 展示了在并发度为 1 的单个虚拟机上执行读写混合和 mergerandom 工作负载时的尾延迟结果。这些结果表明，LightIOV 的延迟性能与 VFIO 和 SPDK-Vhost 相近，并且优于 virtio。LightIOV 借助 Queue 直通，能够实现与 VFIO 相近的延迟性能。由于 I/O 路径较长以及上下文切换开销，virtio 在测试的虚拟化方案中性能最差。

总体而言，结果表明 LightIOV是一种很有前景的 NVMe 虚拟化解决方案，可提供高吞吐量和低延迟。

多虚拟机。首先，我们在多个虚拟机上运行并发度为 1 的 readrandom 工作负载，并逐步将虚拟机数量增加至 100。如图 11a 所示，随着虚拟机数量增加，LightIOV 始终保持性能优势。当虚拟机数量从 10 增至 100 时，LightIOV 的总延迟始终比 SPDK 低 3.6% 至 32.5%。值得注意的是，当虚拟机数量达到 25 时，LightIOV 的性能比 virtio 高 29.3%，展现出卓越的可扩展性。这些

结果表明，LightIOV 是一种具有高度可扩展性的解决方案，能够在真实世界工作负载中保持其性能优势。

随后，我们在 100 个虚拟机上运行了各种工作负载，每个虚拟机仅运行一个线程。每个虚拟机的平均延迟如图11b所示。具体而言，在 readrandom 场景中，LightIOV 的延迟比 SPDK-Vhost 低 32.5%。与 virtio 相比，LightIOV 的性能也更优，在 readrandom 和 readwhilewriting 场景中的延迟分别低 21.9% 和 55.7%。

单个和多个虚拟机上的测试结果表明，在实际应用中，就每秒操作数和延迟而言，LightIOV 的表现均优于 SPDK-Vhost 和 virtio。

6 相关工作

基于软件的 NVMe 虚拟化。Virtio [43]是一系列维护良好的 Linux 通用 I/O 设备虚拟化驱动程序，但存在明显的性能下降问题。为解决这一问题，研究人员提出了先进的基于轮询的方法，以协助虚拟机监控器处理虚拟设备的 I/O。SPDK vhost-NVMe [51] 是一种 I/O 服务目标，它依赖用户空间 NVMe 驱动程序来消除不必要的虚拟机_退出开销，并缩短宿主操作系统中的 I/O 执行栈。MDev-NVMe [40] 是一种采用轮询模式的中介式直通机制；进一步的研究 [39] 提出了适用于 MDev-NVMe 的自适应轮询模式，以提升性能或节省 CPU 资源，从而提高可扩展性。FinNVMe [38] 是一种面向 NVMe 存储设备的高吞吐量存储虚拟化管理方案，能够在多租户虚拟机之间根据工作负载进行管理。最后，Direct-Virtio [25]利用用户级存储直接访问，借助轮询方法避免重复的 I/O 栈开销。然而，这些基于轮询的解决方案需要付出较高的 CPU 开销。

硬件辅助NVMe虚拟化。 VFIO [48]为单个虚拟机提供独占的设备分配，这违背了虚拟化的可共享性理念。SR- IOV [44] 支持多个虚拟机共享物理功能（PF）和虚拟功能（VF），并实现高性能。FVM [28] 是一种基于 SR-IOV 构建的解决方案，通过将虚拟化层卸载到 FPGA卡上，使其能够直接管理物理设备。这种方法可以减少与基于软件的虚拟化相关的开销，并为高负载工作负载实现高性能。LeapIO [30] 将整个存储栈卸载到 ARMSoC，并同时支持本地和远程存储。ScalableIOV [22] 是一种新颖的硬件辅助I/O 虚拟化方法，旨在为I/O 设备提供高度可扩展且高性能的虚拟化，该方法依赖先进的硬件特性和优化来支持大规模高效虚拟化 I/O 设备。BM- Store [16]依靠基于 FPGA 的 BMS-Engine 和基于 ARM 的 BMS-Controller，为裸金属云提供透明且高性能的虚拟本地存储。

7 未来工作

LightIOV 允许虚拟机直接访问 NVMe 设备，通过 NVMe I/O 队列直通实现高性能、低开销、灵活性和可扩展性。然而，由于 SSD 控制器内部的请求处理机制，即使每台虚拟机都拥有独占的 I/O 队列和唯一的逻辑块地址，虚拟机之间的性能干扰仍不可避免。从主机内存中取出 I/O 请求后，SSD 控制器还需通过主机接口逻辑 (HIL) 、闪存转换层 (FTL) 和闪存通道控制器 (FCC) 进行进一步处理，然后才会将 I/O 请求发送至后端地址空间。因此，来自不同虚拟机的请求可能会在 SSD 控制器内部相互干扰，导致性能下降。例如，当两台虚拟机共用一块固态硬盘时，一台运行轻负载（块大小较小且队列深度较低），另一台运行重负载（块大小较大且队列深度较高），前者的性能将受到严重影响。未来，我们计划在固件层面增强 NVMe 控制器，通过将 I/O 队列与后端队列、专用通道或闪存裸片绑定，并优化缓存管理，以减少不同虚拟机之间的性能干扰。

此外，NVMe 设备通常需要具备服务质量（QoS）能力，以便根据上层业务需求区分 I/O 读写服务，例如满足 99.99% 的尾延迟要求。为应对这一挑战，我们旨在将 NVMe 设备控制器内可共享的硬件资源（如 NVMe 命名空间或 LBA 范围、I/O Queue 和 PRP/SGL）关联起来，为虚拟机构建具有优先级的 NVMe 资源域。通过这种方式，不同虚拟机可以获得不同级别的 I/O 优先级服务，确保每台虚拟机都能获得满足其独特业务需求所需的资源。

8 结论

在本文中，我们提出了 LightIOV，这是一种新颖的高性能且可扩展的基于软件的 NVMe 虚拟化机制，能够轻松扩展到数万台虚拟机。LightIOV 的核心思想是 NVMeI/O 队列直通，使虚拟机能够直接访问 NVMe I/O 队列。LightIOV 通过将虚拟化的控制资源（包括 PCIe 配置、BAR空间和管理队列）与数据资源（包括 NVMe I/O队列、门铃寄存器、中断资源和逻辑块地址）相结合，提供完全虚拟化的 NVMe设备。此外，在IOMMU 支持下，LightIOV 实现了绕过主机的 DMA 事务和中断

处理以增强NVMe 虚拟化性能。通过这种方式，LightIOV在不消耗宝贵的CPU 资源且不需要专用硬件支持的情况下，实现了高性能和高可扩展性。

参考文献

[1] 2023。灵活的输入/输出测试器。https://github.com/axboe/fio [2] 2023。非一致性内存访问（NUMA）。https://en.wikipedia.org/wiki/Non-uniform_memory_access [3] Keith Adams和Ole Agesen。2006 年。面向 x86 的软件和硬件虚拟化技术比较。ACM Sigplan Notices（41，11，2006，2–13）。[4] Alexandru Agache、Marc Brooker、Alexandra Iordache、Anthony Liguori、Rolf Neugebauer、Phil Piwonka 和 Diana-Maria Popa。2020 年。Firecracker：面向无服务器应用的轻量级虚拟化。在 NSDI，第 20 卷。419–434。[5] Alexandru Agache、Marc Brooker、Alexandra Iordache、Anthony Liguori、Rolf Neugebauer、Phil Piwonka 和 Diana-Maria Popa。2020 年。Firecracker：面向无服务器应用的轻量级虚拟化。在 NSDI，第20卷。419–434。[6] 阿里巴巴。2023 年。阿里云。检索日期：2023 年 3 月 28 日，来源：https://www.alibabacloud.com/en[7] 亚马逊。2023 年。亚马逊网络服务。检索日期：2023 年 3 月 28 日，来源：https://www.aboutamazon.com/what-we-do/amazon-web-services[8] AMD。2022 年。AMD 输入/输出虚拟化技术（IOMMU）规范。https://www.amd.com/en/support/tech-docs/amd-io-virtualization-technology-iommu-specification[9] AMD。2023年。AMD EPYCT M 7773X。检索日期：2023 年 3 月 28 日，来源：https://www.amd.com/en/products/cpu/amd-epyc-7 554 [10] 2023 AMD。。使用 Vhost-user 的 SPDK 输入/输出虚拟化。https://spdk. io/doc/vhost_processing.html [11] Ganesh Ananthanarayanan、Ali Ghodsi、Scott Shenker和IonStoica。2011 年。数据中心计算中的磁盘局部性被认为无关紧要。在 HotOS，第 13 卷。12–12。[12] Amro Awad、BrettKettering 和YanSolihin。2015 年。高性能输入/输出系统中的非易失性内存主机控制器接口性能分析。在 2015 年 IEEE 系统与软件性能分析国际研讨会（ISPASS）上。IEEE，145–154。[13] Muli Ben-Yehuda、Michael D.Day、Zvi Dubitzky、MichaelFactor、Nadav Har'El、Abel Gordon、Anthony Liguori、OritWasserman 和 Ben-AmiYassour。2010 年。Turtles 项目：嵌套虚拟化的设计与实现。在第9届 USENIX 操作系统设计与实现研讨会（OSDI 10）上。USENIX协会，加拿大不列颠哥伦比亚省温哥华。https://www.usenix.org/conference/osdi10/turtles-project-design-and-implementation-nested-virtualization [14] Muli Ben-Yehuda、MichaelFactor、Eran Rom、Avishay Traeger、Eran Borovik和 Ben-AmiYassour。2012 年。通过低开销虚拟化添加高级存储控制器功能。在 FAST，第 12 卷。15–15。[15]JamesBoyd。2023 年。串行 ATA AHCI：规范，修订版 1.3.1。https://www.intel.com/content/www/us/en/io/serial-ata/serial-ata-ahci-spec-rev1-3-1.html [16] Yiquan Chen、Jiexiong Xu、Chengkun Wei、Yijing Wang、Xin Yuan、Yangming Zhang、Xulin Yu、Yi Chen、Zeke Wang、ShuibingHe等。2023年。BM-Store：一种透明且高性能的本地存储架构，为裸金属云实现大规模部署。在 2023 年 IEEE高性能计算机体系结构国际研讨会（HPCA）上。IEEE，1031–1044。[17]英特尔公司。2017 年。Intel DC P4510 固态硬盘规范。https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/data-center-ssds/dc-p4510-series.html [18] NVMExpress。2023 年。NVM Express。检索日期：2023 年 3 月 28 日，来源：https://nvmexpress.org/specifications [19] Ada Gavrilovska、Sanjay Kumar、Himanshu Raj、Karsten Schwan、Vishakha Gupta、Ripal Nathuji、Radhika Niranjan、AditRanadive和Purav Saraiya。2007 年。高性能虚拟机监控器架构：HPC 系统中的虚拟化。载于 HPC 系统级虚拟化研讨会（HPCVirt）。[20] Google。2023 年。Google Cloud。检索于 2023 年 3 月 28 日，网址：https: //cloud.google.com [21] AbelGordon、Nadav Amit、Nadav Har'El、Muli Ben-Yehuda、Alex Landau、Assaf Schuster 和 DanTsafrir。2012 年。ELI：面向输入/输出虚拟化的裸机性能。ACM SIGPLAN Notices 47，4（2012），411–422。[22] Intel。2018 年。Intel® 可扩展输入/输出虚拟化技术规范。https://www.intel.com/content/www/us/en/developer/articles/technical/introducing-intel-scalable-io-virtualization.html [23] Intel®。2023 年。面向定向输入/输出的 Intel® 虚拟化技术。https://edc.intel.com/content/www/us/en/publications/specification-nuc12dc m-nuc12edb/intel-virtualization-technology- for-directed-i-o/[24] Sangeeth Keeriyadath。2016 年。面向云中机器的NVME 虚拟化构想。载于存储开发者大会。24–27。[25] SewoogKim、Heekwon Park 和 Jongmoo Choi。2021 年。Direct-Virtio：一种面向 NVMe固态硬盘的新型直接虚拟化输入/输出框架。Electronics 10，17（2021），2058。[26] Sanjay Kumar、HimanshuRaj、Karsten Schwan和Ivan Ganev。2007年。为多核系统重新设计虚拟机监控器：Sidecore 方法。载于操作系统与计算机体系结构交互研讨会（WIOSCA）。Citeseer。[27] Yossi Kuperman、Eyal Moscovici、Joel Nider、Razya Ladelsky、AbelGordon 和 Dan Tsafrir。2016 年。半虚拟化远程输入/输出。ACM SIGARCHComputerArchitecture News 44，2（2016），49–65。[28] Dongup Kwon 、 Junehyuk Boo 、 Dongryeong Kim 和 JangwooKim。2020年。FVM：用于快速、可扩展且灵活的存储虚拟化的FPGA辅助虚拟设备仿真。载于第 14届USENIX 操作系统设计与实现会议论文集。955–971。[29] Alex Landau、MuliBen-Yehuda 和 AbelGordon。2011年。SplitX：多核上的客户机/虚拟机监控器分离执行。载于第 3 届输入/输出虚拟化研讨会（WIOV 11）。USENIX协会，美国俄勒冈州波特兰。https://www.usenix.org/conference/wiov11/splitx-split-guesthypervisor-execution-multi-core[30] HuaichengLi、Mingzhe Hao、Stanko Novakovic、Vaibhav Gogte、Sriram Govindan、Dan RK Ports、Irene Zhang、Ricardo Bianchini、HaryadiS Gunawi 和 AnirudhBadam。2020 年。LeapIO：ARMSoC 上高效且可移植的虚拟 NVMe 存储。载于第二十五届编程语言与操作系统架构支持国际会议论文集。591–605。[31] Zijun Li、Jiagan Cheng、Quan Chen、Eryu Guan、Zizheng Bian、Yi Tao、Bin Zha、Qiang Wang、WeidongHan 和 Minyi Guo。2022年。{RunD}：面向无服务器计算中高密度部署与高并发启动的轻量级安全容器运行时。载于 2022 年 USENIX年度技术会议（USENIX ATC 22）。53–68。[32] Zijun Li、Jiagan Cheng、Quan Chen、Eryu Guan、Zizheng Bian、Yi Tao、Bin Zha、Qiang Wang、Weidong Han 和MinyiGuo。2022 年。RunD：面向无服务器计算中高密度部署与高并发启动的轻量级安全容器运行时。载于 2022 年 USENIX年度技术会议（USENIX ATC 22）。USENIX协会，美国加利福尼亚州卡尔斯巴德，53–68。https://www.usenix.org/conference/atc22/presentation/li-zijun-rund[33] Jiuxing Liu 和BulentAbali。2009年。使用专用CPU 核心加速输入/输出虚拟化的虚拟化轮询引擎（VPE）。载于第 23 届国际超级计算会议论文集。225–234。[34] Filipe Manco、Costin Lupu、Florian Schmidt、JoseMendes、Simon Kuenzer、Sumit Sati、Kenichi Yasukata、CostinRaiciu和Felipe Huici。2017 年。我的虚拟机比你的容器更轻（也更安全）。载于

第*26*届操作系统原理研讨会.218–233. [35] 微软。2023。Microsoft Azure。检索于2023年3月28日，来源：https://azure.microsoft.com/en-us [36] Dushyanth Narayanan、Eno Thereska、Austin Donnelly、SamehEl- nikety和Antony Rowstron。2009。将服务器存储迁移至固态硬盘：权衡分析。载于 第*4*届*ACM*欧洲计算机系统会议论文集. 145–158. [37] Kirti Wankhede Neo Jia。2016。VFIO中介设备。 https://docs.kernel.org/driver-api/vfio-mediated-device.html [38] 彭博、杨明、姚建国和管海兵。2021。一种面向吞吐量、具备工作负载感知管理的NVMe存储虚拟化方案。*IEEE*计算机学报。 70, 12 (2021), 2112–2124. https://doi.org/10 1109 2020 3037817 ./TC..彭博、姚建国、董耀祖和管海兵。2022。MDev-NVMe：采用自适应轮询的中介直通NVMe虚拟化解决方案。*IEEE*计算机学报。 71, 2 (2022), 251–265. https: //doi.org/10 1109 2020 3045785 ./TC..彭博、张浩中、姚建国、董耀祖、徐宇和管海兵。2018。MDev-NVMe：采用中介直通的NVMe存储虚拟化解决方案。载于 *2018*年*USENIX*年度技术会议（*USENIX ATC 18*）。USENIX协会，马萨诸塞州波士顿，665–676。https://www.usenix.org/conference/atc18/presentation/peng [41] RocksDB。2023。快速存储环境下的持久化键值存储。 https://rocksdb.org [42] MendelRosenblum和CarlWaldspurger。2011。输入/输出虚拟化：将逻辑设备与其物理实现解耦可带来诸多显著优势。 *Queue* 9, 11 (2011), 30–39. [43] Rusty Russell。2008。virtio：迈向虚拟I/O设备的事实标准。*ACM SIGOPS*操作系统评论 42, 5 (2008), 95–103. [44] PCI SIG。2010。单根I/O虚拟化与共享规范。[45] Akshat Verma、Ricardo Koller、Luis Useche 和 RajuRangaswami。2010。SRCMap：使用动态整合实现能耗按比例扩展的存储。在 *FAST*，第 10 卷。267–280。[46] Alex Williamson。2012。VFIO：用户视角。在*KVM*论坛。[47] Alex Williamson。2015。IRQ 旁路管理器。https://lwn.net/Articles/653706/ [48] Alex Williamson。2023。VFIO：用户视角。检索于 2023 年 3 月 28 日，来源：https://docs.huihoo.com/kvm/kvm-forum/2012/2012- forum-VFIO.pdf [49] Qiumin Xu、Huzefa Siyamwala、MrinmoyGhosh、Tameesh Suri、Manu Awasthi、Zvika Guz、Anahita Shayesteh 和Vijay Balakrishnan。2015。NVMe 固态硬盘的性能分析及其对实际数据库的影响。在第 *8* 届 *ACM*国际系统与存储会议论文集。1–11。[50] Ziye Yang、James R Harris、Benjamin Walker、Daniel Verkamp、刘昌鹏、Cunyin Chang、曹刚、Jonathan Stern、Vishal Verma和 Luse EPaul。2017。SPDK：用于构建高性能存储应用的开发工具包。在*2017 IEEE*云计算技术与科学国际会议（*CloudCom*）。IEEE，154–161。[51] Ziye Yang、刘昌鹏、Yanbo Zhou、Xiaodong Liu 和 曹刚。2018。SPDK vhost-NVMe：通过用户空间vhost目标端加速虚拟机在 NVMe 固态硬盘上的输入/输出。在*2018 IEEE* 第 *8* 届云计算与服务计算国际研讨会（*SC2*）。IEEE，67–76。[52] 张先涛、郑晓、王智、李琦、傅俊康、张扬 和 沈一彬。2019。在大型云基础设施中实现快速且可扩展的 VMM在线升级。在第二十四届编程语言与操作系统架构支持国际会议论文集。93–105。