華中科技大學

研究生（数据中心技术）课程论文（报告）

题目：现代计算存储设备的虚拟化机制研究综述

学　　号　　M202173734

姓　　名　　张纯

专　　业　　电子信息

课程指导教师　　施展、童薇

院（系、所）　计算机科学与技术

2021 年 12 月 28 日

现代计算存储设备的虚拟化机制研究综述

张纯

摘要 存储虚拟化技术是计算机虚拟化技术的重要结构，它的思想是将资源的逻辑映像与物理存储分开，为系统和管理员提供一幅简化、无缝的资源虚拟视图。存储虚拟化技术具有提高动态适应能力。它将存储资源统一集中到一个大容量的资源池，无需中断应用即可改变存储系统和实现数据移动，对存储系统能够实现单点统一管理。在存储单元内部或附近包含计算单元的计算存储设备是一种非常有前途的技术，可以最大化存储服务器的性能。然而，要应用这种计算存储设备并在虚拟化环境中充分发挥其潜力，服务器架构师必须解决一个基本挑战：具有成本效益高的虚拟化。这个关键的挑战可以通过以下问题直接解决：（1）如何虚拟化两个不同的硬件单元（即计算和存储），（2）如何集成它们来构建虚拟计算存储设备，以及（3）如何将它们提供给用户。然而，由于缺乏硬件辅助的虚拟化支持，现有的计算存储虚拟化方法严重存在低性能和高成本的问题。本文综述了近年来存储设备虚拟化上的一些研究成果，分别介绍了 3 篇文献中针对虚拟化性能与成本优化的方案和实现效果，最后根据这 3 篇文献的研究成果，提出了针对计算存储设备虚拟化的性能和成本优化的一些可能的解决办法。

关键词 存储虚拟化 计算存储设备 硬件辅助

1 引言

虚拟化技术到底是什么?广义上来说，就是通过映射或抽象的方式屏蔽物理设备复杂性，增加一个管理层面，激活一种资源并使之更易于透明控制。它可以有效简化基础设施的管理，增加 IT 资源的利用率和能力，比如服务器、网络或存储。存储虚拟化是一种贯穿于整个 IT 环境、用于简化本来可能会相对复杂的底层基础架构的技术。存储虚拟化的思想是将资源的逻辑映像与物理存储分开，从而为系统和管理员提供一幅简化、无缝的资源虚拟视图。对于用户来说，虚拟化的存储资源就像是一个巨大的“存储池”，用户不会看到具体的磁盘、磁带，也不必关心自己的数据经过哪一条路径通往哪一个具体的存储设备。

将存储资源虚拟成一个“存储池”，这样做的好处是把许多零散的存储资源整合起来，从而提高整体利用率，同时降低系统管理成本。与存储虚拟化配套的资源分配功能具有资源分割和分配能力，可以依据“服务水平协议（service level agreement）”的要求对整合起来的存储池进行划分，以最高的效率、最低的成本来满足各类不同应用在性能和容量等方面的需求。特别是虚拟磁带库，对于提升备份、恢复和归档等应用服务水平起到了非常显著的作用，极大地节省了企业的时间和金钱。

除了时间和成本方面的好处，存储虚拟化还可以提升存储环境的整体性能和可用性水平，这主要是得益于“在单一的控制界面动态地管理和分配存储资源”。

在当今的企业运行环境中，数据的增长速度非常之快，而企业管理数据能力的提高速度总是远远落在后面。通过虚拟化，许多既消耗时间又多次重复的工作，例如备份/恢复、数据归档和存储资源分配等，可以通过自动化的方式来进行，大大减少了人工作业。因此，通过将数据管理工作纳入单一的自动化管理体系，存储虚拟化可以显著地缩短数据增长速度与企业数据管理能力之间的差距。

存储虚拟化可在三个层次上实现，分别是：基于主机的虚拟化、基于存储设备的虚拟化、基于网络的虚拟化。它有两种实现方式，分别是带内虚拟化、带外虚拟化。实现的结果有：块虚拟化，磁盘虚拟化，磁带、磁带驱动器、磁带库虚拟化，文件系统虚拟化，文件/记录虚拟化。近来，兼容异构存储，同时具备完整数据保护和管理功能的成熟存储虚拟化产品也被广泛应用于两地三中心容灾以及双活数据中心的建设当中，作为一种积极的，可靠的技术

手段有效提升用户原有生产系统对各类型灾难的防御能力。

本文将综述在存储虚拟化性能与成本优化上近 3 年的一些研究。本文后续的章节结构如下：

2 理论

2.1. SSD-FPGA 硬件平台

SSD-FPGA 硬件平台。现代计算存储设备将其计算和存储单元集成在一起，并通过板载互连将它们耦合。图 1（下）显示了现代 SSD-FPGA 计算存储设备的典型硬件平台。它利用一个 FPGA 来进行计算，并允许它直接访问一个附加的 NVMeSSD。对于近存储数据处理，它还支持通过其内部交换机(例如，板载 PCIe 交换机)在计算单元和存储单元之间的对等(P2P)数据通信。例如，通过将 FPGA 的 DRAM 空间暴露给 SSD，并在内部交叉条交换机中实现路由策略，计算存储设备可以使计算和存储单元能够直接交换数据，而不需要使用软件仲裁。

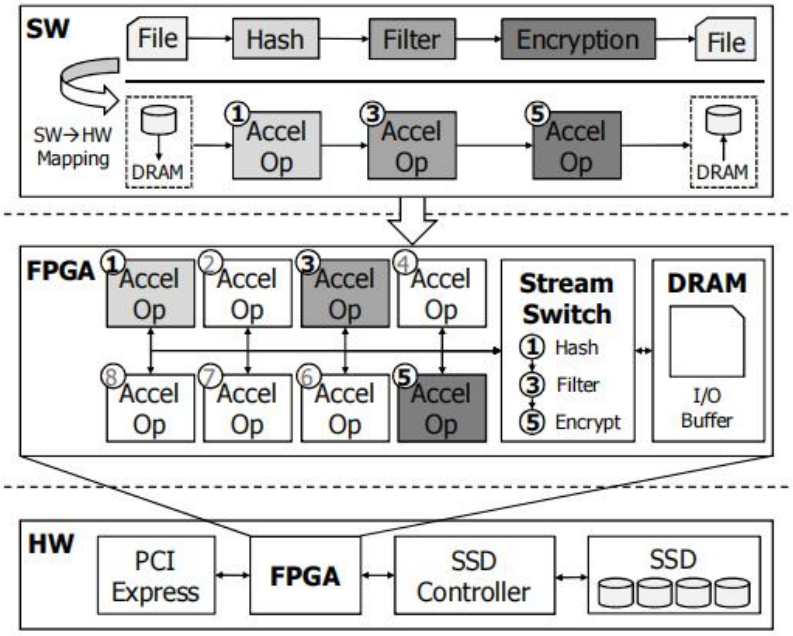

图 1

FPGA 覆盖架构。在计算存储设备上实现的 FPGA 覆盖架构提供了可编程的近存储处理。它们包括可重构的操作符、横条流开关和板载 DRAM。操作符是可重构的流处理单元，每个处理单元都可以被选择性地激活来配置全流操作。车载横条开关协调操作员和 FPGA 的 DRAM 之间的数据移动。图 1（中间）显示了一个用于可编程近存储处理的覆盖架构示例。这个覆盖体系结构包含 8 个操作符，其中三个被激活以服务于用户指定的流工作负载（散列、过滤器、加密）。然后，在流交换机中安装一个路由策略，以正确地将传入的数据流路由到目标操作符（哈希→过滤器→加密）或 DRAM 缓冲区。

FPGA 覆盖体系结构还实现了存储接口，以允许其操作员直接访问存储单元。现代计算存储设备采用 NVMe 标准存储协议，提供快速和并行的存储访问。NVMe 具有以下关键优势。首先，NVMe 支持多个 I/O 队列来充分利用高带宽的存储单元。通过将单独的 NVMe 提交队列(SQ)/完成队列(CQ)对分配给不同的处理核，NVMe 可以同时运行多个存储操作。其次，NVMe 通过最小化与内存映射的 I/O(MMIO)操作的数量来实现快速存储操作。例如，NVMe 设备公开了一组 SQ/CQ 门铃寄存器，并且只需要一个 MMIO 写入操作（即，门铃寄存器写入）。

2.2. 硬件辅助虚拟化

为了克服基于软件的虚拟化机制的性能开销和主机效率低下，硬件辅助虚拟化机制允许客户操作系统直接访问目标 PCIe 设备，而不需要任何软件仲裁。为了实现 PCIe 设备的直接分配（即直通虚拟化），介绍了 I/O 存储器管理单元(IOMMU)及其直接内存访问(DMA)和中断重映射机制。DMA 重映射机制允许使用客户物理地址来完成来自虚拟设备的 DMA 操作。类似地，中断重映射机制将由虚拟化设备引起的中断向量转换为虚拟机上下文。但是，这种方法要求将一个物理设备只分配给单个虚拟机，并且不支持跨多个虚拟机的设备共享。

为了解决这些缺点，PCIeSR-IOV 允许在硬件级别上由许多虚拟机共享一个物理设备。具有 SR-IOV 功能的设备在设备接口上显示多种物理功能(PFs)和虚拟功能(VFs)（即，虚拟设备实例）。由于 VFs 具有单独的 PCIe 配置寄存器，包括基本地址寄存器(BARs)，因此 SR-IOV 可以在为多个 VM 提供服务的同时强制执行资源隔离。此外，一个具有 SR-IOV 能力的设备实现了如何在其内部桥接模块上复用自身，因此不依赖任何主机软件来复用其虚拟设备实例。

3 论文介绍

3.1. LeapIO: Efficient and Portable Virtual NVMe Storage on ARM SoCs[1]

3.1.1. 研究背景

由于云对存储资源的管理和隔离有较高需求，复杂虚拟化存储栈不仅拖慢了虚拟磁盘的性能，还带来了宿主机 x86 核的开销。ARM 核相对 x86 核价格更低，功耗也更低，因此整体使用成本可以降低很多倍。因此，文章中的主要工作是解决将虚拟化存储栈负载转移到 ARM SoC 的各种具体问题。

3.1.2. 设计方法

在抽象级别上，作者使用 NVMe。从客户操作系统、LeapIO 运行时到新的存储服务/功能，所有涉及的软件层都可以看到相同的设备抽象：虚拟 NVMe 驱动器。它们都通过成熟的 NVMe 队列对机制进行通信，该机制可以通过 x86 和 ARM、QPI 或 PCIe 之间普遍存在的基本内存指令进行访问。

在软件方面，作者构建了一个运行时，它对存储服务隐藏了 NVMe 映射的复杂性。该运行时跨 x86 和 ARM 内核提供了统一的地址空间。

详细设计如下：

(1) 硬件设计如图 2：

① 通过 SoC 访问主机 DRAM。SoC 必须有一个连接到主机 DRAM 的 DMA 引擎(就像 rNIC 一样)。但是，它必须允许用户空间 LeapIO 运行时(在 ARMSoC 中运行)访问 DMA 引擎，以到达在 on-x86 用户 VM、rNIC、SSD 和内部 SoC 服务之间映射的所有 NVMe 队列对的位置。

② 通过 SoC 访问 IOMMU。受信任的-SoCLeapIO 运行时必须访问与主机一致的 IOMMU，以便对提交 IO 的 VM 地址空间的页表行走。当 on-x86 用户 VM 访问一段数据时，该数据驻留在主机 DRAM 中，但是 VM 只提交数据的来宾地址。因此，SoC 必须方便 LeapIO 运行时通过 IOMMU 将对象转换为主机物理地址。

③ SoC 的 DRAM 已映射到主机。对于零拷贝 DMA，on-SoCDRAM 必须由 rNIC 和 SSD 可见。为此，SoC 必须将其 DRAM 空间作为一个 PCIeBAR（基本地址寄存器）公开给主机 x86。然后，BAR 将被主机操作系统映射为主机物理地址空间的一部分。

④ 网卡共享。NIC 必须在主机 x86 和 ARMSoC 之间“共享”，因为在 x86VM、其他主机代理和 SoC 服务都使用 NIC。主机可以使用 NIC 来服务 VM 流量，也可以使用 SoC 来卸载远程存储功能。一种可能性是将 ARM 核心和 NIC 共同定位在同一 PCIe 卡上(图 2 中的“NIC**”)，因此不依赖于外部 NIC 能力。

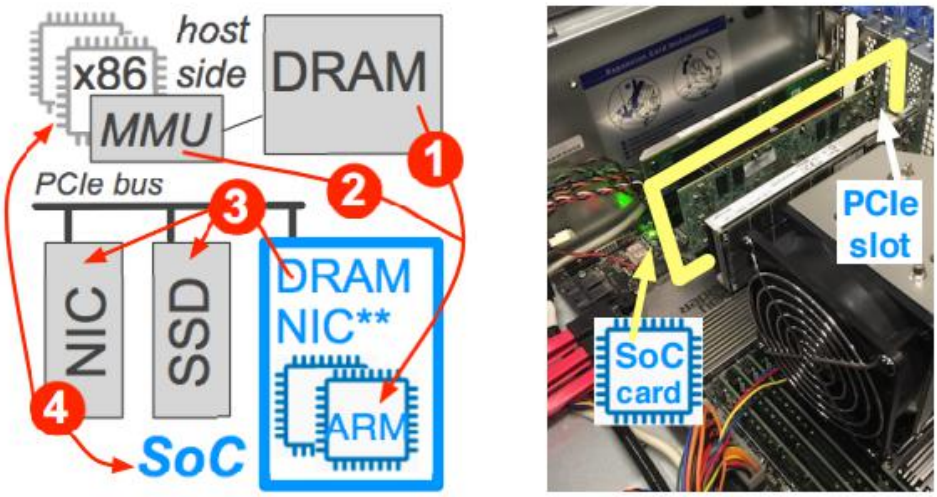

图 2

(2) 软件设计如图 3：

a) 用户的虚拟机。在客户端，用户在不需要修改的 VM 上运行她自己的、她选择的应用程序和客户操作系统。对于存储，客户 VM 运行在典型的 NVMe 设备接口(例如，/dev/nvme0n1)上，由 LeapIO 公开为包含提交和完成队列(SQ 和 CQ)[1]的队列对（由●表示）。

b) 主机的操作系统。在主机操作系统中添加了一个用于构建队列对映射的功能，这样 LeapIO 运行时 c 就可以看到暴露给 VM 的相同的 NVMe 命令队列。

c) 短暂的存储。如果用户 VM 使用本地 SSD（例如，用于吞吐量），那么这些请求将被放入映射在 LeapIO 运行时和 SSD 设备之间（向下的●-●）的 NVMe 队列中。因为 SSD 不在 SoC 中（不在粗体边内），所以它们需要共享存储在主机 DRAM 中的 NVMe 队列。

d) 客户端 LeapIO 运行时和服务。客户端运行时（阴影区域）表示在 ARMSoC 上运行的 LeapIO 运行时（蓝色粗边）。此运行时“胶连接”网络连接（◆-◆）上的所有 NVMe 队列对（●-●）和端到端存储路径。

e) 远程访问(NIC)。如果用户将数据存储在远程 SSD 或服务中，则客户端运行时只需通过 NIC（◆-◆）通过 TCP 或 RDMA 将 IO 请求转发到服务器运行时。注意，NIC 与 ARMSoC 相同的 PCIe 槽（虚线粗边）中，以便填充属性 HW4(可共享的 rNIC)。

f）服务器端 LeapIO 运行时和服务。服务器端 LeapIO 运行时通过轮询连接到客户端（◆）的队列来准备传入的命令和数据。然后它调用服务器端存储函数，该函数也在 SoC 的用户级别运行。服务器端服务可以将 IO 转发/转换到一个或多个 NVMe 驱动器（●）或远程服务。该图显示了对其本地 SSD（最右边的●-●）的访问路径。

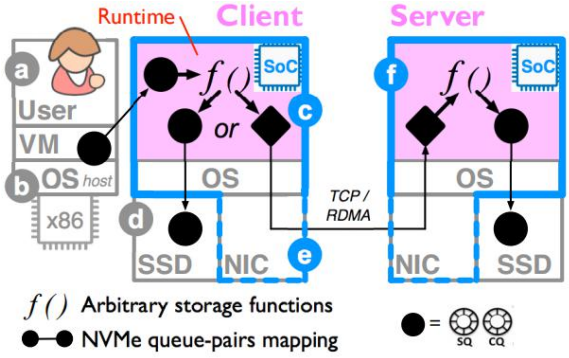

图 3

3.1.3 实验

Leap 与其它虚拟化技术的运行时速度对比如图 4：

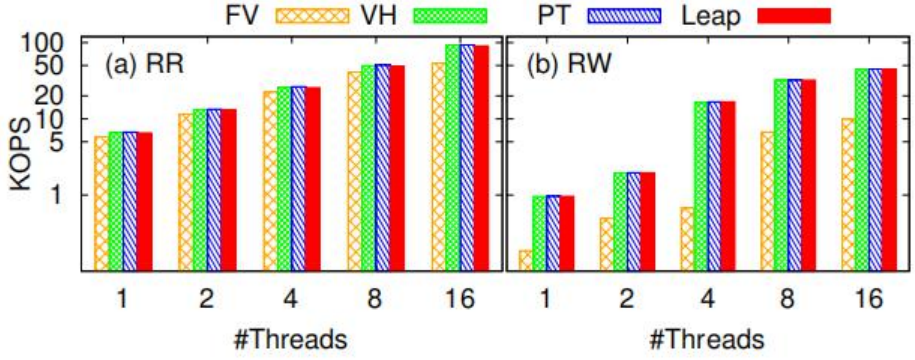

图 4

LeapIO 运行时速度快，因为 NVMe 直接映射。对于每个 64 字节的提交条目，LeapIO 运行时只需要通过简单的计算和内存获取来翻译 2-3 个字段（用于翻译）。

3.2. FVM: FPGA-assisted Virtual Device Emulation for Fast, Scalable, and Flexible Storage Virtualization[2]

3.2.1. 研究背景

硬件辅助虚拟化可以为快速存储设备实现合理的性能，但它牺牲了虚拟化环境中有限的功能（例如，迁移、复制、缓存）。为了以最小的性能退化恢复 VM 特性，最近的进展建议通过将计算核心用于虚拟设备仿真来实现一个新的基于软件的虚拟化层。然而，由于具有昂贵的通用内核和主机驱动的存储设备管理的性质，随着每个服务器的存储设备数量和性能的增加，所提出的方案提高了关键的性能和可伸缩性问题。

FVM 关键思想是在与主机资源解耦的 FPGA 卡(FVM 引擎)上实现存储虚拟化层，实现设备控制方法，使卡直接管理物理存储设备。通过这种方式，配备了 FVM 引擎的服务器可以从虚拟和物理设备管理中节省宝贵的主机端资源(即 CPU、内存带宽)，并利用解耦的 FPGA 资源进行虚拟设备仿真。

3.2.2. 设计方法

（1）FPGA 辅助的存储虚拟化

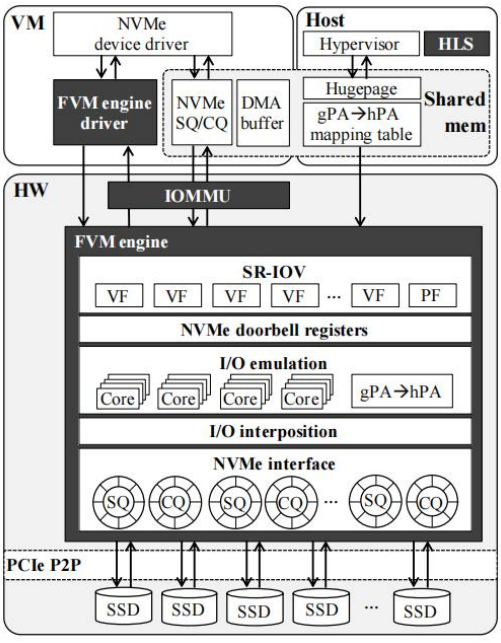

图 5

图 5 显示了 FVM 的体系结构及其组件。首先，FVM 完全绕过主机软件堆栈，并最小化对主机资源的使用。通过在 FVM 引擎上的 SR-IOV 实现，虚拟机可以进入一个虚拟化层，而不需要来自主机软件的任何仲裁支持。此外，其硬件级的 NVMe 接口使该卡通过 PCIe 直接管理物理 NVMe 设备。其次，通过使用并行的设备仿真架构，FVM 能够与许多 NVM 设备进行扩展。FVM 引擎不依赖于设备上的 SoC 核心，而是集成了许多专门的硬件单元来轮询和模拟客户 I/O 操作。第三，其基于 hls 的设计流程为 FVMmengent 和其他存储管理服务实现了灵活和可编程的实现。

(2) 前端：VM 到 FVM 引擎，如图 6

① 直接 FVM 引擎分配。FVM 通过其 SR-IOV 接口将每个 FVM 引擎的虚拟实例分配给 VM。当前的 FVM 引擎实现集成了一个 PCIeIP 块来启用它自己的 SR-IOV 接口，并支持最多 4 个物理功能(PFs)和 252 个虚拟功能(VFs)。PFs 由主机软件管理，用于资源管理，每个 VF 被分配给一个 VM，专门用于直接访问 FVM 引擎。FVM 引擎还通过将非重叠地址转换应用于其内部地址空间，成功地从不同的虚拟机中分离出 MMIO。同时，通过 IOMMU 支持，FVM-引擎可以对客户内存空间执行 DMA 事务，并在没有主机软件仲裁的情况下注入中断。

② 门铃寄存器重新映射。FVM 引擎为 NVMe 门铃寄存器保留了一个内存空间，并通过 PCIeBARs 公开它。当来宾 NVMe 设备驱动程序调用 nvme_write_sq_db()提交 I/O 请求时，来宾 FVM 引擎驱动程序会拦截它们并获取它们的（虚拟）设备 iD、SQiD 和 SQ 尾部信息。然后，来宾 FVM 引擎驱动程序计算目标门铃寄存器的地址，并将接收到的 SQ 尾指针写入 FVM 引擎（图 6-①）。

③ 虚拟 I/O 队列仿真。为了在硬件层处理客户 I/O 请求，FVM 引擎使用多个 FVM 核心轮询门铃寄存器（图 6-②）。算法 1 通过其轮询例程来模拟虚拟 NVMe 设备。首先，FVM 核心获取新更新的 SQ 尾（第 5 行），并将其与存储先前尾值的 SQ 头（第 6 行）进行比较。由于 FVM-ement 使用芯片内存保留门铃内存区域，它可以快速轮询这些区域。

④ PRP 和 LBA 翻译。FVM-enent 处理从 V 机接收到的 NVMe 命令，然后将其提交到物理 NVMe 设备。具体来说，FVM 引擎操作物理区域页面(PRP)条目(指向客户 DMA 缓冲区)从 gPAs 到 hPAs。为了在硬件级别上启用这种 gpa 到 hpa 的转换，FVM 利用页面分配固定内存。由于当前的操作系统不改变它们的物理位置，FVM 引擎可以通过合并 gPA--hPA 映射表来静态翻译 PRP 条目。在此设计中，转换不会产生任何性能开销，因为 FVM 引擎使用其芯片上的内存来管理映射表。此外，由于主页(2MB)，所需的表大小足够小，可以将它们保存在片上内存中（即，4KB 表以覆盖 1GB 的来宾内存空间）。

⑤ 虚拟管理队列仿真。FVM 通过 QEMU 和 SPDK 虚拟目标实现管理虚拟 NVMe 管理 SQ/CQ 对。由于 QEMU 和 KVM 可以跟踪管理门铃寄存器上 MMIO 引起的虚拟机出口，它们仍然能够通过 UNIX 域插字与 SPDK 主机目标交互。QEMU 按照传统的虚拟目标协议，向 SPDK 目标实现提供关键的管理命令(例如，I/O 队列创建、删除、关闭)。

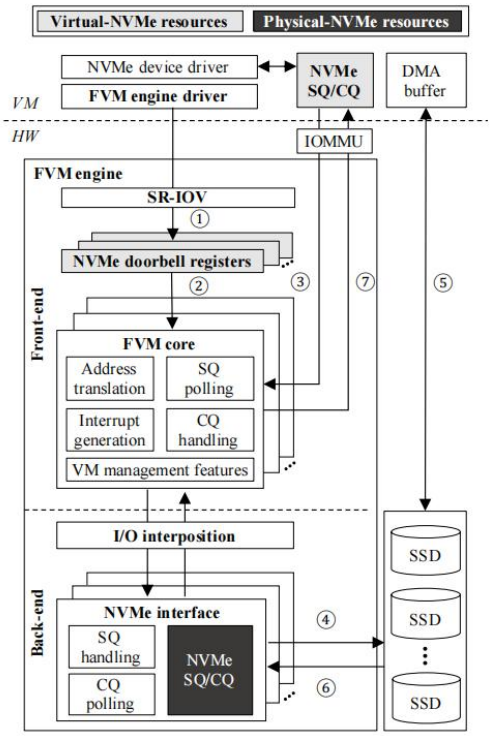

图 6

(3) 后端：FVM-Engine 到 SSD

① 物理 SQ/CQ 重新映射。为了允许 FVM 引擎直接与物理 NVMe 设备交互，主机软件将其 NVMeI/O 队列重新映射到 FVM 引擎的 PCIeBAR 区域。在安装时，主机 FVMmegent 驱动程序向物理 NVMe 设备提供内存映射区域的地址。NVM 设备并不知道 FVM 引擎，但是一个 PCIe 交换机可以将 DMA 事务无缝地传递到 FVM 引擎中。

② 直接的 NVMe 设备控制机制。FVM 引擎集成了标准的 NVMe 接口来实现一个直接的设备控制机制。（1）FVM-engen 将 NVMe 命令移动到 FVMmengent 芯片内存中的提交队列。（2）FVM-engen 然后按响 NVMe 设备中的门铃寄存器，通知新提交的命令数量。（3）NVMe 控制器通过 PCIeP2P 通信来获取 NVMe 命令（图 6-④）。（4）NVMe 设备处理命令后（图 6-⑤）后，将命令完成数写入 FVM 引擎地址空间（图 6-⑥）。（5）FVMmegent 处理它们，（6）敲响位于 NVMe 设备中的门铃寄存器。

③ Polling CQs。为了立即处理来自物理设备的完成操作，NVMe 接口将轮询其 CQ 内存空间。算法 2 显示了它的轮询功能。首先，NVMe 接口处理其头指针指向的 CQ 条目，并将其相位与当前轮进行比较。这使 NVMe 界面可以确定一个新条目是否作为前一轮完成通知或当前一轮完成通知的一部分发布。之后，它处理完成项，并将它们转发到前端。然后，FVM 核心会将完成消息写入客户 CQ 内存空间。由于 FVM-engne 使用芯片内存管理所有 SQ/CQ 对，因此它的轮询例程不会产生任何性能开销。

(4) FVM 核心设计，如图 7 所示：

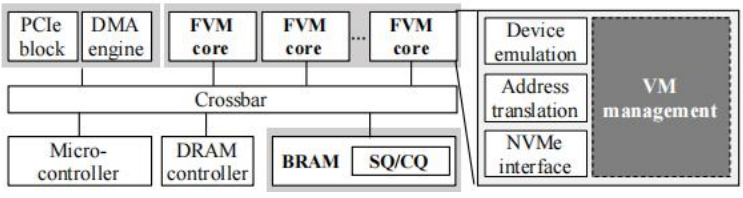

图 7

3.2.3 实验

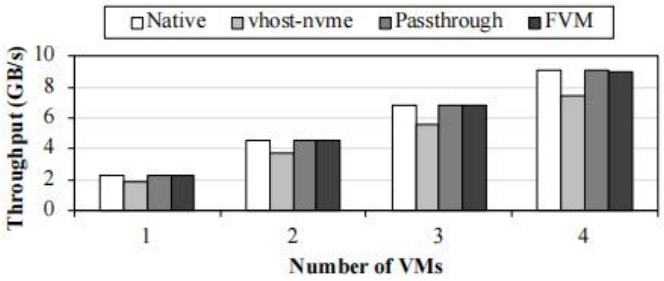

图 8

3.3 A Fast and Flexible Hardware-based Virtualization Mechanism for Computational Storage Devices[3]

3.3.1. 研究背景

将计算单元集成在其存储单元内部或其存储单元附近的现代计算存储设备正在成为高性能存储服务器的一种非常有前途的解决方案，因为它可以通过近存储处理将数据移动开销降到最小化。配备了计算存储设备的服务器可以将数据密集型例程卸载到计算单元，并使其能够在没有软件干预的情况下访问存储在存储单元中的数据。例如，当前现成的计算存储设备包含高端现场可编程门阵列(FPGAs)进行计算，并采用 NVMExpress(NVMe)存储协议，允许 FPGAs 直接访问固态驱动器(ssd)。

由于缺乏硬件辅助的虚拟化支持，现有的计算存储虚拟化方法性能低，成本高。

首先，基于软件的虚拟化机制（例如，副虚拟化）不能充分利用近存储处理，因为它们的重管理程序和主机操作系统堆栈通过客户和主机操作系统来模拟虚拟计算存储设备，及其间接资源编排机制。

其次，现有的虚拟化机制由于不能为每个虚拟机静态分配计算和存储单元，将导致较高的硬件成本。

3.3.2. 设计方法

首先，FlexCSV 实现了由硬件辅助的虚拟化和资源编排。通过硬件级别的标准单根 I/O 虚拟化(SR-IOV)层，FlexCSV 提供了一个快速的、绕过主机的虚拟化堆栈，并允许多个虚拟机利用接近存储的处理能力。此外，FlexCSV 在硬件级别管理用户请求的流操作，以减轻协调近存储处理的软件负担。

其次，FlexCSV 通过动态构建和调度计算单元和存储单元，实现了较高的成本效益。为了提高其可伸缩性，FlexCSV 采用了 SSD-FPGA 解耦架构，并允许 FPGA 加速器卡构建许多具有多个 PCI 快速(PCIe)附加 ssd 的虚拟计算存储设备。此外，它从共享硬件操作符池进行的动态资源配置和部分重新配置支持可以捕获 VM 工作负载的动态行为，并以最低的硬件成本显著减少 QoS 违规。

(1) FlexCSV 架构

图 9 显示了 FCSV-Engine 体系结构及其主要硬件组件。FCSV-Engine 实现了①一个基于 PCIeSR-IOV 的硬件辅助虚拟化层，②一个硬件级别的直接设备编排机制，③一个 SSD-FPGA 解耦架构，以及通过操作员重命名和部分重新配置支持的，④动态资源分配。

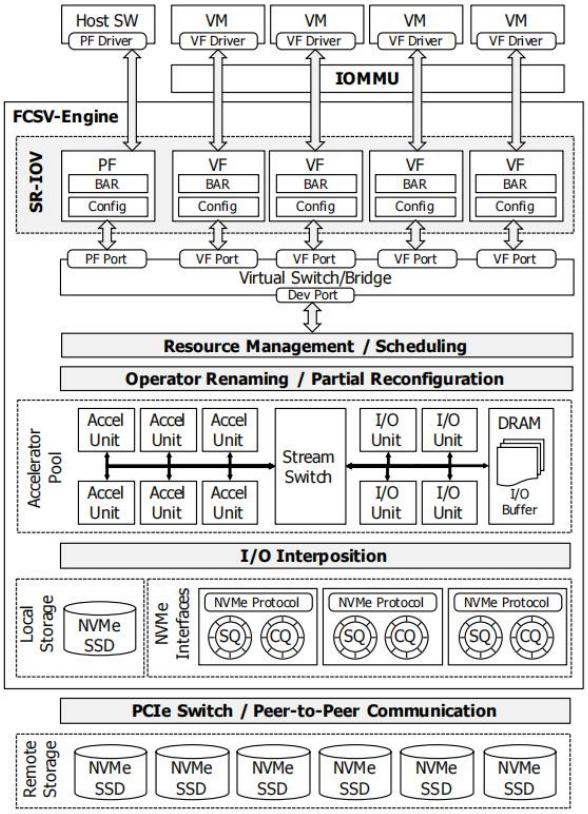

图 9

(2) 硬件辅助虚拟化

a) SR-IOV 和 VM 隔离

为了减轻计算存储虚拟化中的软件开销，FCSV-Engine 在一个标准的 PCIeSR-IOV 层下提供了硬件辅助的虚拟化。通过合并 SR-IOV，FCSV-Engine 可以在硬件级别上虚拟化自己，每个 VF 都可以被专门分配给一个 VM 以进行直接访问。另一个优点是，SR-IOV 不需要额外的服务器 CPU 内核来轮询客户 I/O 活动和间接中断注入，这使得比传统的以软件为中心的虚拟化机制（例如，陷阱和模拟、侧孔）更具有可伸缩性和成本效益的服务器配置。

b) 设备接口

FCSV-Engine 实现了一个多队列设备接口和一个门铃机制来与客户操作系统交互。对于这个多队列设备接口，我们将 FCSV-Engine 的 PCIeBAR 区域安排为虚拟 FCSVEngine 实例的(即每个 VF)的门铃寄存器。从软件方面，安装在客户操作系统上的 FCSV-Engine 的设备驱动程序分配多个 SQ/CQ 对，并初始化映射到 FCSV-Engine 的 BARs 上的门铃寄存器。FCSV-Engine 的设备驱动程序将分配队列对的来宾物理地址传递给 FCSV-Engine。队列对的数量由 FCSV-Engine 的 BAR 配置和 FPGA 片上资源预算决定。

(3) 硬件级的资源编排

a) 近存储处理命令

图 10 显示了一个接近存储器的处理命令结构。首先，op_chain 指定应该激活哪些操作符以及被激活的操作符的目标流顺序。此外，如果操作员需要参数来处理，用户可以直接或 file_param 字段或间接携带参数。用户还可以通过操作 request_id 字段来管理近存储处理命令之间的依赖关系。如果请求包含与以前请求相同的清除，则在之前的请求完成其接近存储处理之前不能发出当前请求。类型决定当前请求是否涉及存储访问。

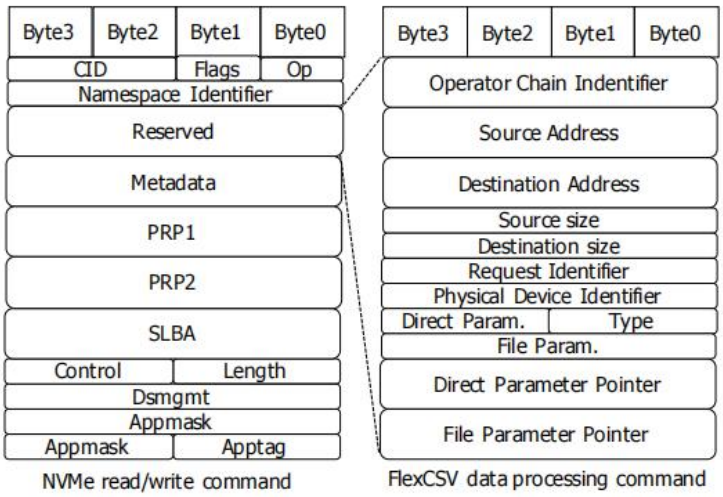

图 10

b) 资源业务流程

图 11 显示了硬件级别的调度机制。首先，FCSV-Engine 通过轮询提交门铃寄存器来识别一个新发布的命令。如果该命令重新查询近存储处理，则 FCSV-Engine 读取数据处理命令并保存其上下文(例如，源/目标 FPGADRAM 地址)。之后，FCSV-Engine 操作接收到的 NVMe 命令的 DMA 缓冲区地址，以便 SSD 可以将用户请求的数据传输到/从 FPGA 的 DRAM。FCSV-Engine 还根据用户请求的数据移动（即读或写）和目标操作符的方向，强制执行正确的计算和存储操作顺序。

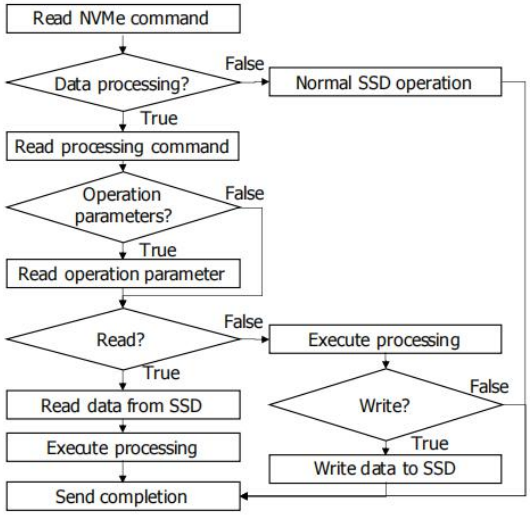

图 11

c) 软件 FCSV-引擎

图 12 说明了软件和硬件的 FCSV-Engine 的实现。在软件实现中，客户操作系统可以利用 nvme 扩展的近存储处理协议，从而减少客户-主机层转换的数量。它还允许设备通过操作 DMA 缓冲区地址到目标设备的内存地址，将数据直接传输到另一个设备的内部内存。

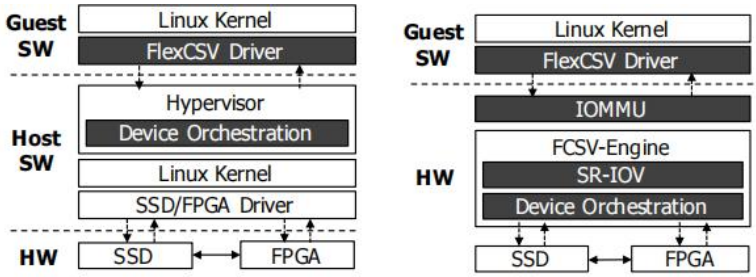

图 12

3.3.3 实验

如图 12 所示，FlexCSV 在各种虚拟化环境中进行的加速比较。

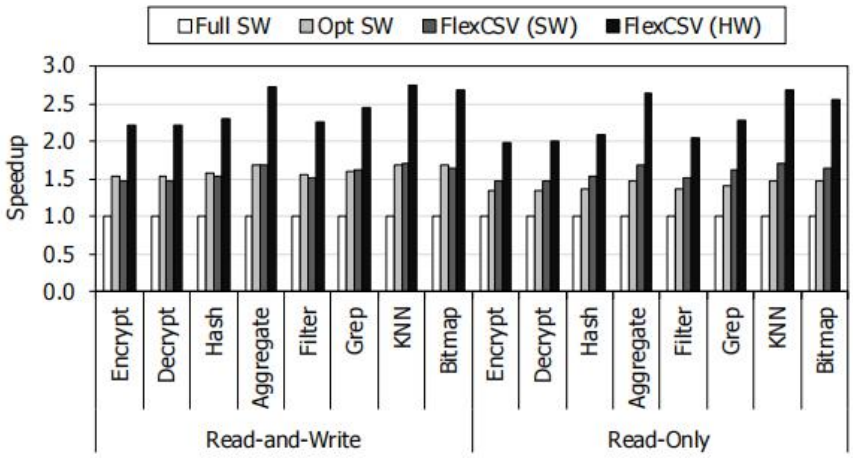

图 13

4 总结与展望

因为 FPGA 的高可编程性、低延迟、高能效等特点，越来越多的云服务提供商，如 Amazon、阿里云、微软 Azure，都开始在云端提供了赛灵思 FPGA 实例，迈出了云计算发展的重要一步。然而截至目前，FPGA 云服务仍以物理卡的形式、面向单一的静态任务，没有很好的针对云端 FPGA 虚拟化的解决方案。

FPGA 在未来将是一个与 CPU、GPU 同样必不可少的芯片品类，特别在数据中心场景里将扮演重要角色。未来的数据中心将是异构计算为核心的硬件数据中心，单独依靠每一个硬件都无法实现整个数据中心性能和能效的最大化，我们必须要综合考虑不同的硬件，以虚拟化思想尝试把所有硬件资源最大化利用起来。业界与学术界此前针对 CPU、GPU 的研究已经较为充分，但对于如何把 FPGA 性能与能效最大化以及如何与其他硬件相结合，以实现一个支持智能应用的异构系统，还有很大的空间可以发展，当然同时也是一个颇具挑战性的课题领域。

FPGA 硬件辅助的虚拟化机制可以极大降低存储虚拟化带来的性能损失，而软件层的虚拟化可以提供更好的兼容性，减少对存储设备等硬件的适配管理开销，真正做到数据共享。对于大规模的云服务上的存储虚拟化部署，需要更好地利用 FPGA 虚拟化及软件虚拟化的优势，在降低管理开销的同时提供更高的性能。

参考文献

[1] Huaicheng Li, Mingzhe Hao, Stanko Novakovic, Vaibhav Gogte, Sriram Govindan, Dan R. K. Ports, Irene Zhang, Ricardo Bianchini, Haryadi S. Gunawi, Anirudh Badam:LeapIO: Efficient and Portable Virtual NVMe Storage on ARM SoCs. ASPLOS 2020: 591-605

[2] Dongup Kwon, Junehyuk Boo, Dongryeong Kim, Jangwoo Kim:FVM: FPGA-assisted Virtual Device Emulation for Fast, Scalable, and Flexible Storage Virtualization. OSDI 2020: 955-971

[3] Dongup Kwon, Dongryeong Kim, Junehyuk Boo, Wonsik Lee, Jangwoo Kim:
A Fast and Flexible Hardware-based Virtualization Mechanism for Computational Storage Devices. USENIX Annual Technical Conference 2021: 729-743