计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(1):286−294 [doi: 10.15888/j.cnki.csa.008240]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 I/O 前后端模型的密码卡软件虚拟化①

唐乐爽, 窦同锐, 桑洪波, 张玉国

(三未信安科技股份有限公司, 济南 250098)
通信作者: 唐乐爽, E-mail: tangleshuang@foxmail.com

摘　要: 密码技术是云计算安全的基础, 支持 SR-IOV 虚拟化的高性能密码卡适用于云密码机, 可以为云计算环境提供虚拟化数据加密保护服务, 满足安全需求. 针对该类密码卡在云密码机使用过程中存在的兼容性不好、扩充性受限、迁移性差以及性价比低等问题, 本文提出了基于 I/O 前后端模型的密码卡软件虚拟化方法, 利用共享内存或者 VIRTIO 作为通信方式, 通过设计密码卡前后端驱动或者服务程序, 完成多虚拟机与宿主机的高效通信, 实现常规密码卡被多虚拟机共享. 该方法可以有效地降低云密码机的硬件门槛, 具有兼容性好、性能高、易扩展等特点, 在信创领域具有广阔的应用前景.

关键词: 信息安全; virtio; 软件虚拟化; 密码卡; 共享内存

引用格式: 唐乐爽,窦同锐,桑洪波,张玉国.基于 I/O 前后端模型的密码卡软件虚拟化.计算机系统应用,2022,31(1):286–294. http://www.c-s-a.org.cn/1003-3254/8240.html

Software Virtualization of Cryptographic Card Based on I/O Front-end and Back-end Model

TANG Le-Shuang, DOU Tong-Rui, SANG Hong-Bo, ZHANG Yu-Guo

(Sansec Co. Ltd., Jinan 250098, China)

Abstract: Cryptographic technology is the foundation of cloud computing security. The high-performance cryptographic cards supporting SR-IOV virtualization technology are suitable for cloud cipher machines, which can realize the encryption protection of virtualization data for cloud computing environments and meet the security requirements. However, these cryptographic cards have unsatisfactory compatibility, limited expansibility, poor migration, and low cost performance when applied in cloud cipher machines. Thus, this study proposes a software virtualization method of cryptographic cards based on an I/O front-end and back-end model. With shared memory or virtio as the communication mode, it completes the efficient communication between multiple virtual machines and the host by designing the front-end and back-end driver or service program of cryptographic cards and realizes that common cryptographic cards can be shared by multiple virtual machines. This method can effectively lower the hardware threshold of cloud cipher machines and makes cryptographic cards possess good compatibility and expansibility and high performance, thus showing broad application prospects in information technology applications and innovation.

Key words: information security; virtio; software virtualization; cryptographic card; shared memory

随着云计算的高速发展, 业务上云已经成为趋势, 这对云服务商保障云上数据的完整性、机密性和可用性提出了更高的要求, 而密码技术及密码模块正是保护重要信息和数据安全的核心技术[1]. 针对云计算环境安全应用多样化、功能复杂化, 密码设备提供商们设计开发了硬件密码产品-云密码机, 该产品结合密码技

① 收稿时间: 2021-03-15; 修改时间: 2021-04-09; 采用时间: 2021-04-20; csa 在线出版时间: 2021-12-17

术与虚拟化技术, 可以实现在一台硬件密码设备上运行多个虚拟化的密码机 (VSM)[2], 满足云计算虚拟环境中存在的高速数据加解密、数字签名等需求. 云密码机的核心密码模块是支持 SR-IOV[3] 虚拟化的高性能密码卡, 具备虚拟出 8–16 个逻辑密码卡的能力, 云密码机相对应的提供 8–16 个虚拟密码机服务.

密码卡虚拟化可以在一台宿主机上实现多虚拟机共享一张密码卡, 提高密码卡的利用率, 国内外在此方面做了相关工作. Berger 等[4] 提出了可信平台模块 (TPM) 的虚拟化, 对密码卡设备的虚拟化有一定的指导意义, 但缺少 PCI-E 总线设备的实现; 杨永娇等[5] 提出了一个基于硬件辅助虚拟化的 VT-d 技术在 Xen 云平台上的安全隔离框架, 缺少具体的实现; Sun 等[6] 设计了一种密码卡虚拟化框架用于实现密码卡在虚拟化环境中使用, 由于该框架性能阈值的调定与动态迁移的受限, 距离实际应用还需要进一步研究; 马龙宇[7] 设计与实现了基于 SR-IOV 虚拟化技术高速密码卡, 但是仅支持 160 位的哈希密码运算; 张嘉夫[8] 提出一种基于密码卡虚拟化的隐私保护方法, 使用动态多安全级虚拟密码卡隐私保护模型解决虚拟桌面下的隐私保护问题, 为了最大化发挥虚拟密码卡的性能, 把多组请求组包发送, 应用范围受限; 任继奎等[9] 基于 SR-IOV 理论设计实现了一种基于 FPGA 的虚拟化设备的数据交互模型, 解决云计算平台与虚拟设备间高速数据交互的需求, 本质上基于硬件虚拟化; 苏振宇[10] 提出了 3 种密码卡虚拟化设计方案, 并实现了基于 SR-IOV 虚拟化技术的硬件虚拟化密码卡和基于现场可编程门阵列 (FPGA) 的软件虚拟化密码卡, 但是硬件虚拟化密码卡方案仍然是基于硬件虚拟化, 而软件虚拟化方案需要重新设计密码卡, 并不适用于现有的密码卡.

综上所述, 目前国内外在密码卡虚拟化方面的研究仍然比较少, 技术发展相对滞后, 缺乏实际应用[10], 而且大部分集中在基于 SR-IOV 的硬件虚拟化方面, 软件虚拟化方面的研究更少. 基于 SR-IOV 的密码卡较常规密码卡价格昂贵, 对云密码机的软硬件有苛刻的兼容条件, 同时受制于有限的设备电路资源, 提供的虚拟逻辑密码卡数量有限, 可扩展性较差. 更为严重的问题是, SR-IOV 很难支持虚拟机的迁移和复制等操作[11].

因此针对该类密码卡在云密码机使用中存在的兼容性差、扩展性受限以及性价比低等问题, 本研究从软件虚拟化的角度出发, 提出了基于 I/O 前后端模型的密码卡软件虚拟化方法, 分别设计并实现了基于共享内存的软件虚拟化和基于 virtio 的软件虚拟化方案, 第 1 种方案适用于 X86 平台云密码机, 第 2 种方案适用于 ARM 平台的云密码机. 本文提出的技术不仅可以使得多虚拟机有效地共享普通密码, 很大程序降低云密码机使用密码卡的硬件门槛, 而且能够弹性扩充虚密码机的数量、实时迁移虚拟机并降低采购密码卡的成本.

1 I/O 虚拟化

I/O 虚拟化可以提高系统运行性能和简化硬件要求, 使得数量不多的物理设备被多个虚拟机共享, 从而增加 I/O 密集型虚拟机的数量. 在系统虚拟化基本结构中, 虚拟机管理器 (VMM) 处于硬件资源与操作系统之间, 负责完成 CPU、内存与 I/O 等硬件资源的虚拟化, 虚拟出来的 I/O 设备可以与物理设备接口保持一致, 也可以完全不同. 一般而言, I/O 虚拟化可以分为全虚拟化、半虚拟化与硬件辅助虚拟化 3 种.

(1) 全虚拟化技术

该 I/O 虚拟化技术通过软件方式模拟真实的硬件设备, 虚拟机无法区分模拟硬件设备与物理设备, 不需要修改虚拟机操作系统与驱动程序, 可以直接使用物理设备的驱动程序驱动该虚拟设备, 图 1 展现了 I/O 全虚拟化模型.

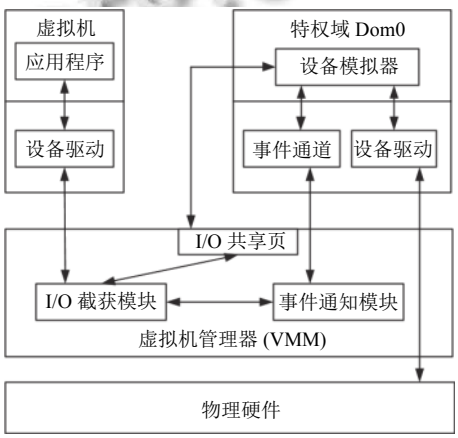


图 1 I/O 全虚拟化模型

当客户机访问模拟 I/O 设备时, 所有 I/O 访问请求都会被虚拟机管理器截获, 并把 I/O 指令内容写入 I/O 共享页, 接着事件通知模块告知设备模拟器收集 I/O 指令, 设备模拟器负责完成本次 I/O 请求的模拟,

最后将执行结果传递给虚拟机. 全虚拟化的移植性与兼容性非常好, 但是性能较差, 软件模拟 I/O 请求的处理需要虚拟机监视器与虚拟机多次交互, 频繁切换上下文, 会造成很大的时间开销.

(2) 半虚拟化技术

该 I/O 虚拟化模型又称为前后端模型, 引入了 I/O 环机制和事件机制, 向虚拟机提供一组新的设备接口. 虚拟机与虚拟机管理器调用这些设备接口, 通过 I/O 环机制进行数据传输, 并采用事件机制实现通信, 减少了传统的中断机制引入的上下文切换频率, 提升了虚拟设备的性能, 图 2 展现了 I/O 全虚拟化模型.

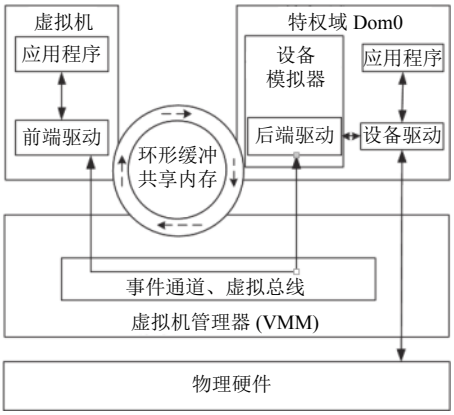


图 2 I/O 半虚拟化模型

半虚拟化比全虚拟化在性能上有很大的提高, 但是半虚拟化需要修改虚拟机操作系统和驱动程序适配新的设备接口, 因此在移植性与兼容性方面受限.

(3) 硬件辅助虚拟化技术

该虚拟化技术可以在硬件层次提供物理设备的共享, 常见的硬件虚拟化包括设备直传与单根 I/O 虚拟化 (SR-IOV). 设备直传需要额外的硬件支持, 如 Intel VT-d[12]、AMD IOMMU[13] 等, 可以直接把物理设备分配给虚拟机, 让虚拟机独占该设备, 但是设备直传技术只能把物理设备分配给一个虚拟机, 不能让多个虚拟机共享此物理设备, 因而在云计算环境并不常用.

SR-IOV 虚拟化是在设备直传技术的基础上, 引入了物理功能 (PF) 与虚拟功能 (VF) 的概念, 在硬件层面对物理资源进行映射, 通过创建不同虚拟功能 (VF) 设备的方式, 让一个 PCIe 物理设备可以对外显示为多个不同的物理设备, 虚拟机管理器可以把这些 VF 设备直接分配给虚拟机, 虚拟机安装特定的设备驱动对 VF 进行访问, 进而实现虚拟机对物理设备的原生共享, 访问过程不需要虚拟机管理器的参与, 图 3 展现了 SR-IOV 模型.

在 SR-IOV 设备中, PF 称为完整的 PCIe 功能, 可以配置管理 SR-IOV 设备, 控制 I/O 数据读取; VF 称为轻量级的 PCIe 功能, 可以进行 I/O 数据读取, 但在配置管理 SR-IOV 设备方面受限.

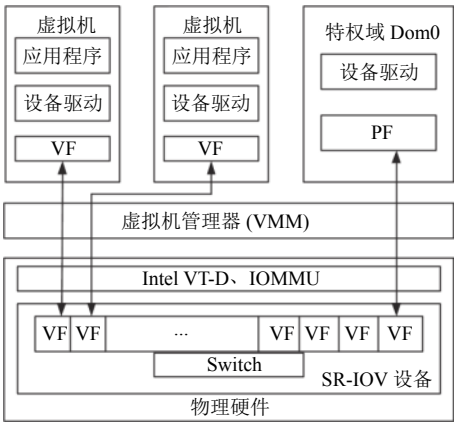


图 3 SR-IOV 模型

2 密码卡软件虚拟化

密码卡一方面作为 I/O 设备, 虚拟化后的性能会成为制约虚拟机密码运算性能的瓶颈, 另一方面作为安全设备, 虚拟机与设备交互的安全性是需要重点兼顾的内容. 因此实现密码卡软件虚拟化, 需要充分考虑性能、共享、安全、易用等问题[10], 借鉴 I/O 半虚拟化的思想, 本文提出并设计了两种基于 I/O 前后端模型的软件虚拟化方案.

(1) 性能: 实现虚拟机与密码卡的高效通信, 尽量减少性能损失, 力求接近物理卡的性能. 当采用虚拟技术后, 在虚拟机中存在的是模拟的逻辑密码卡, 逻辑设备对接物理设备要经过一次 I/O 访问过程, 导致虚拟机中密码卡的性能与直接使用物理密码卡的性能相比存在 50%~90% 的损失[14];

(2) 安全: 必须保证数据传输的安全稳定可靠, 不能够使用易受攻击的通信方式, 尤其是密码卡这种进行密码运算的设备, 如 TCP/IP 通信, 大部分攻击都是通过网络进行的[15];

(3) 易用: 不能修改操作系统与软件模拟器, 使用通用版的操作系统与软件模拟器才能保证密码卡软件虚拟化的易用性;

(4) 共享: 提高密码卡的共享能力, 可以保证密码卡为多虚拟机提供密码运算服务.

3 基于共享内存的密码卡软件虚拟化

3.1 总体设计

基于共享内存的密码卡软件虚拟化总体设计如图 4 所示, 虚拟机中的应用程序通过接口库调用 ivshmem 设备驱动, 设备驱动搬运数据到共享内存, 然后向伪虚拟机发送请求, 伪虚拟机接收到请求后, 服务程序读取共享内存数据并调用密码卡完成数据的密码运算处理, 处理完成之后回写共享内存, 最后通过唯一 ID 通知虚拟机, 对应的虚拟机收到中断, 操作共享内存, 完成一次通信.

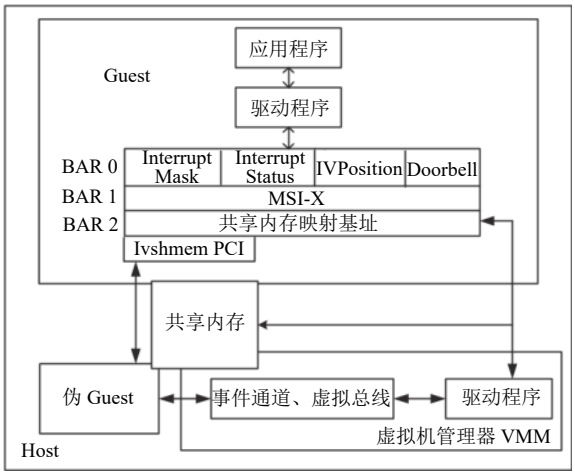


图 4 基于共享内存的密码卡软件虚拟化总体设计图

Ivshmem 可以让多虚拟机与宿主机通过共享内存实现通信, 启动虚拟机时需要预先设置 ivshmem 设备的中断数量、共享内存名称、共享内存大小等, 为了优化软件虚拟化的效率, 消除共享内存引入信号量带来的性能影响, 本方案对共享内存采用统一申请、统一分配的方式, 即使用服务程序申请一块内存, 使用虚拟机的唯一 ID 确定各虚拟机在该块共享内存的地址, 在多个虚拟机同时申请调用密码卡完成密码运算时, 可以先把数据拷贝到根据 ID 确定的共享内存地址, 通知伪虚拟机依次处理, 以此实现多虚拟机调用密码卡的异步请求, 减少时间开销, 充分发挥密码卡密码卡的性能.

其中, 除了服务程序要设计之外, ivshmem 设备的前端驱动也需要重新设计. 密码卡驱动的特殊之处在于可以根据虚拟机的 ID 定位在共享内存中分配的位置, 其余的本质上都是通用的 PCIe 驱动, 这里就不在详细描述, 调用接口库与密码卡驱动是不需要做任何修改的.

3.2 共享内存的实现

共享内存, 顾名思义就是指多个不相关的进程可以访问同一段物理内存, 这是一种非常有效地共享和传递数据的方式, 节省了一次数据拷贝的时间. 当然, 这要求使用共享内存通信的多个不相关的进程处于同一个计算机系统内, 此外共享内存没有提供同步机制, 为避免访问中出现的同步和互斥问题, 需要由操作共享内存读写的进程来控制同步访问[16]. 因此, 方案一采用共享内存方式实现通信, 同时选择使用 eventfd 中断实现同步.

目前 QEMU/KVM 是最流行的虚拟化组合, 也是绝大部分企业虚拟化方案的首先, QEMU 负责模拟各类输入输出设备, KVM 负责 CPU 虚拟化与内存虚拟化. Ivshmem 是 QEMU 模拟的一个 PCI 设备, 支持 eventfd 中断同步, 通过共享内存让多虚拟机与宿主机实现通信. 在虚拟机内, ivshmem 对应的 PCI ID 为 1af4:1110, 支持 3 个 PCI 基地址寄存器, BAR 0 为 1 KB MMIO (memory-mapped I/O) 区域的设备寄存器, 包括中断 MASK 寄存器、中断 Status 寄存器、IVPosition 寄存器与 Doorbell 寄存器, 其中 IVPosition 寄存器为只读寄存器, 其值为虚拟机启动时被分配的 ID, BAR 1 用于 MSI-X 中断, BAR 2 为共享内存映射基址. 通过 ivshmem 方式, 虚拟机与虚拟机之间可以实现中断模式下的通信, 而宿主机与虚拟机之间支持非中断模式的通信, 通信流程如图 5 所示.

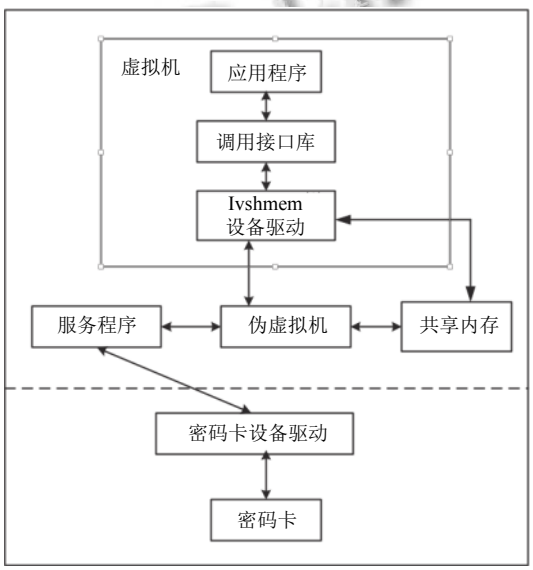


图 5 基于 ivshemm 的宿主机与虚拟机通信示意图

在重新设计基于 ivshmem 内存共享设备的前端驱动时, 为避免不必要的系统调用与数据拷贝, 本文使用

UIO 机制的用户态驱动替代常规的内核态驱动, 可以在用户空间直接操作 ivshmem 共享内存, 优化了从虚拟机到宿主机之间的通信效率.

3.3 服务程序的设计

为了使用 ivshmem 设备基于 eventfd 的中断机制, 需要设计一个服务程序, 运行在宿主机, 称之为 ivshmem-server. 该服务程序可以调用密码卡, 启动一个 ID 固定为 0 的虚拟机 (称之为伪虚拟机), 创建共享内存, 等待 qemu 的连接请求, 每启动一个虚拟机, ivshmem-server 会自动分配给虚拟机 eventfd 文件描述符与 ID 号, 并且把这两个信息通知给其它虚拟机, 每退出一个虚拟机, ivshmem-server 也会通知退出虚拟机的 ID. 每个虚拟机的 ID 都是独一无二的, 在收发数据时用来标识虚拟机, 虚拟机之间通过 eventfd 机制通知中断. 每个虚拟机都在与本身 ID 所绑定的 eventfd 文件描述符上侦听, 并且使用其它虚拟机的 eventfd 向其它 guest 发送中断, 服务程序的处理流程如图 6 所示.

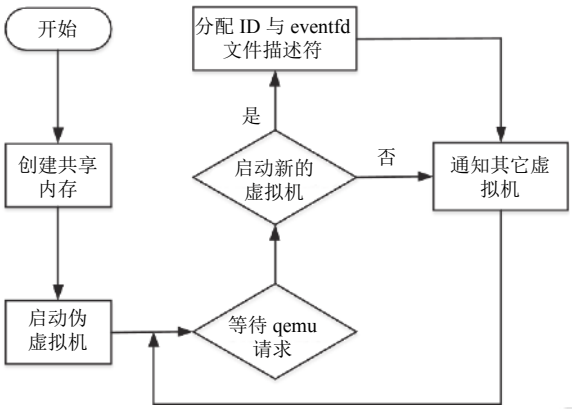


图 6 ivshmem-server 服务程序的处理流程图

在已检索的文献中, 文献 [15] 提出了一种比较通用的 ivshmem-server 服务程序设计方法, 一个伪虚拟机对应一个虚拟机, 同时启动一个守护线程响应调用请求. 与该方法相比, 本文改进的服务程序可以实现一个伪虚拟机对应多个虚拟机, 后台只需启动一个守护线程就可以响应不同虚拟机的调用请求, 不仅可以降低对系统资源的消耗, 还可以在守护线程中流水式的处理调用请求, 减少多线程响应请求带来的时间开销.

4 基于 vhost-vsock 的密码卡软件虚拟化

4.1 总体设计

基于 vhost-vsock 的密码卡软件虚拟化总体设计如图 7 所示, 虚拟机中的应用程序通过接口库调用 virtio 相关接口, 数据经由 vhost-vsock 传输直接送达宿主机的内核, 密码卡设备驱动从 vhost-vsock 模块获取数据后交给密码卡进行密码运算处理, 数据处理完成之后原路返回给虚拟机, 完成一次通信.

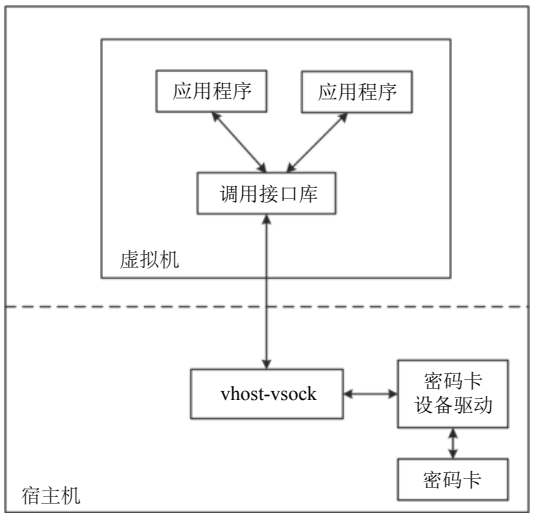


图 7 基于 vhost-vsock 的密码卡软件虚拟化总设计图

其中, 调用接口库需要稍微做一下修改, 密码卡设备驱动需要重新设计, 能够从 vhost-vsock 获取数据, 并可以处理多虚拟机的并发调用.

4.2 vhost-vsock

virtio[17] 是一种 I/O 半虚拟化优化解决方案, 提供了一整套虚拟化设备与上层应用的通用虚拟化框架, 不仅提高了 I/O 虚拟化的性能, 还提高了开发效率与跨平台的兼容性, KVM 中半虚拟化驱动实现的方式就是采用了针对 Linux 的 I/O 虚拟化框架 virtio.

在 qemu 中 virtio 设备具体表现为一个模拟的 PCIe 设备, 通过安装 virtio 的前端驱动, 可以实现 virtio 设备与后端 (KVM/QEMU) 的交互, 采用 virtio 的网络半虚拟化称为 virtio-net[18], 优化版的网络半虚拟化称为 vhost-net[18], 使用 vhost 技术, 在内核中加入了 vhost-net.ko 模块, 使得网络数据可以在内核态得到处理, 减少了用户态/内核态切换时间和包的拷贝次数, 进一步提升了性能, 如图 8 virtio-net 与 vhost-net 框图所示.

基于 vhost-net 的 vhost-vsock[19], 为虚拟机的网络协议栈提供了新的后端, 使用一对 32 位的整数 CID: PORT 来标识进程, 类似 TCP/IP 通信中的 IP: PORT, 可以直接与宿主机内核中的 vhost_vsock 模块通信, 宿

主机固定使用 2 作为 CID, 每个虚拟机在启动时会被自动分配一个 CID. vhost-vsock 定义了一套新的套接字地址族, 支持 socket API, 没有使用 TCP/IP 协议, 甚至不需要网络接口, 可以零配置实现虚拟机与宿主机间的高性能通信, 如图 9 vhost-vsock 通信示意图所示.

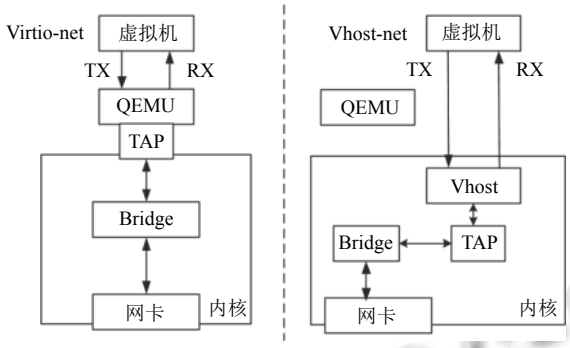


图 8 virtio-net 与 vhost-net

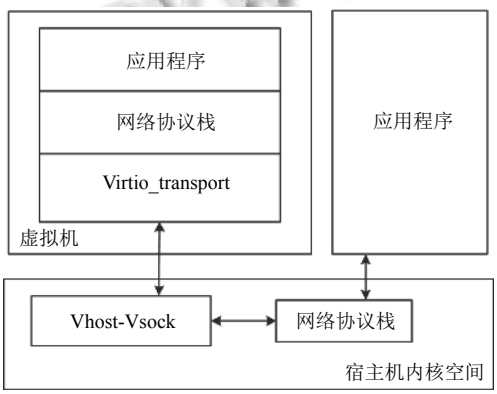


图 9 vhost-vsock 通信示意图

4.3 密码卡驱动的设计

原有的密码卡设备驱动位于操作系统底层, 负责密码卡的初始化, 透明传输密码卡与系统之间的数据, 不解析、不截获数据. 基于 vhost-vsock 的密码卡驱动, 不仅需要具备原有驱动的功能, 还得具备 vhost-vsock 服务器端的功能, 能够响应多虚拟机的并发请求, 从 vhost-vsock 获取数据.

驱动程序的处理流程如图 10 所示, 重新设计后的驱动在初始化设备后, 启动了两个处理线程, vhost-vsock 服务线程用来响应虚拟机的调用请求, 工作任务处理线程用来处理工作链表中的任务, 当虚拟机调用请求到达, vhost-vsock 服务线程会为该请求启动一个对应的服务线程, 把到达的任务加入任务链表, 等待工作任务处理线程的处理.

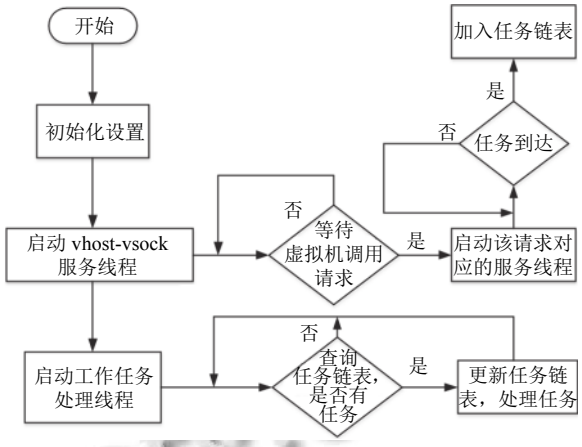


图 10 密码卡设备驱动处理流程图

重新设计后的驱动可以预先接收待处理的数据到宿主机的内核空间, 把虚拟机的请求任务加入工作链表异步处理, 减少了多虚拟机并发调用密码卡的等待时间, 提高了本方案软件虚拟化的性能.

5 实验与分析

本次实验采用 X86 与 ARM 双平台, 方案一 (基于共享内存的密码卡软件虚拟化) 适用于 x86 平台, 方案二 (基于 vhost-vsock 的密码卡软件虚拟化) 适用于 ARM 平台, 密码卡采用三未信安股份有限公司生产的 SJK1828(#2) 系列密码卡, 可以处理 0~16 KB 大小的数据包, 测试性能时选择 SM1、SM2 与 SM4 三种常用的国密算法, 分别覆盖对称算法与非对称算法, 平台硬件与软件具体配置如表 1 所示.

5.1 实验一. 性能测试

本实验测试两种方案在 X86 平台的性能表现, 测试数据在不经过密码卡处理的情况下从虚拟机应用层到宿主机内核层再到虚拟机内核层 (vm->vhost->vm) 的系统吞吐量, 数据包大小从 1–16 KB, 实验结果如图 11 所示.

表 1 实验配置

平台	CPU	内存 (GB)	硬盘 (TB)	操作系统	内核版本	Qemu 版本
X86	E5-2680 v3	32	1	Centos 7	4.9.250	4.1.1
ARM	Phytium, FT-2000+/64	128	1.8	Kylin v4	4.19.90-19.ky10.aarch64	3.1.0

由图 11 可知: 使用共享内存的方式进行数据传输, 通信速度与数据包长度基本成线性正比关系, 斜率变

化不大, 也即 vm->vhost->vm 每秒通信次数变化不大, 数据包长度越大意味着通信速度越高; 而采用 vhost-vsock 的方式进行数据传输, 斜率逐渐变小, 也即随着数据包长度变大, vm->vhost->vm 每秒通信次数变化逐渐变小; 同时, 相同数据包长度, 使用共享内存的方式的通信速度始终大于采用 vhost-vsock 的方式, 在包长为 4 KB 时, 共享内存通信速率可以达到 2000 Mb/s 左右, vhost-vsock 可以达到 1 600 Mb/s 左右, 在包长为 8 KB 时, 共享内存通信速率可以达到 4 000 Mb/s 左右, vhost-vsock 可以达到 3 000 Mb/s 左右.

本次实验证明, 在传输数据方面共享内存的方式优势非常明显. 由于在 ARM 平台使用 ivshmem 模拟设备, 会因共享内存页面属性不一致进而产生硬件不维护 cache 一致性问题, 导致虚拟机崩溃, 因此只能退而求其次在 ARM 使用 vhost-vsock 通信方式.

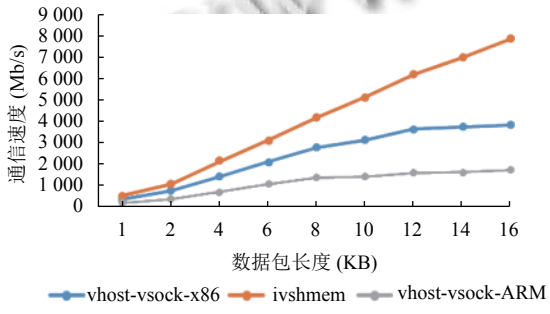
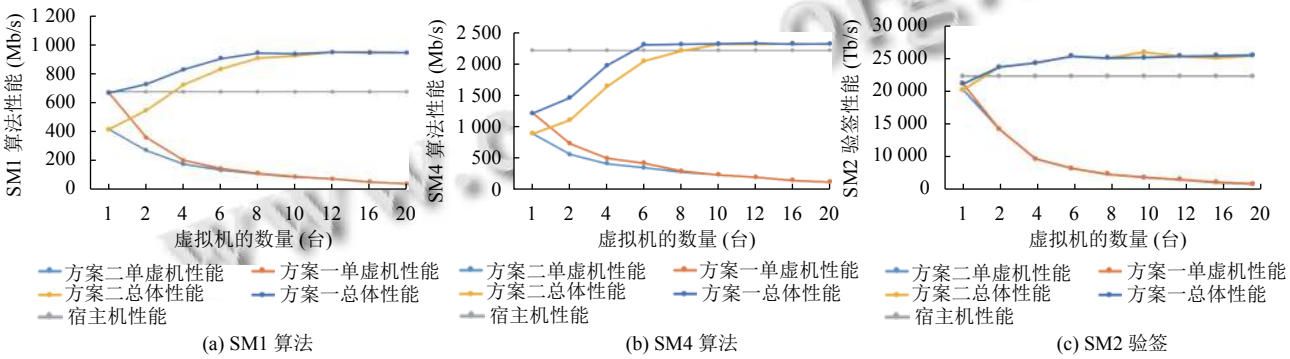


图 11 密码卡设备驱动处理流程图

5.2 实验二. x86 平台下算法性能

本实验测试方案一在 X86 平台调用密码卡的性能表现, 测试的算法分别包括 SM1、SM4 与 SM2, 测试的数据包长度为 4 KB, 测试过程中逐渐增加虚拟机的数量, 并在开启不同的虚拟机数量时, 测试方案 2 的算法性能做对比, 图中标注了密码卡在宿主机下的性能作为参考, 具体测试结果如图 12 所示.

由图 12 可知, 对于方案一, 只有一台虚拟机调用密码卡时, 方案二的 SM1 与 SM2 验签算法性能可以达到密码卡在宿主机性能下的 95% 以上, 随着虚拟机并发调用密码卡的数量增加, 单台虚拟机的算法性能逐渐下降, 多台虚拟机的整体算法性能超过宿主机算法性能, 并缓慢增加最后逐渐保持稳定; 对于方案二, 只有一台虚拟机调用密码卡时, 方案二的 SM1 算法性能可以达到密码卡在宿主机性能下的 60% 以上, SM4 算法性能可以达到密码卡在宿主机性能下的 40% 以上, SM2 验签算法性能可以达到密码卡在宿主机性能下的 90% 以上, 随着虚拟机并发调用密码卡的数量增加, 单台虚拟机的算法性能逐渐下降, 多台虚拟机的整体算法性能超过宿主机算法性能, 并缓慢增加最后逐渐保持稳定; 对比两种方案, 无论是在单台虚拟机还是多台虚拟机的情况下, 方案一的算法性能表现要优于方案二, 因此在 x86 平台采用方案一是最优选择.



(a) SM1 算法 (b) SM4 算法 (c) SM2 验签

图 12 x86 平台下 SM1、SM4 与 SM2 算法性能测试图

5.3 实验三. ARM 平台下算法性能

本实验测试方案二在 ARM 平台调用密码卡的性能表现, 测试的算法分别包括 SM1、SM4 与 SM2, 测试的数据包长度为 4 KB, 测试过程中逐渐增加虚拟机的数量, 并同密码卡在宿主机下的性能表现做对比, 图中标注了密码卡在宿主机下的性能作为参考, 具体测试结果如图 13 所示.

由图 13 可知, 只有一台虚拟机调用密码卡做密码运算时, 方案二的算法性能可以达到密码卡在宿主机性能下的 92% 以上, 随着虚拟机并发调用密码卡的数

量增加, 单台虚拟机的算法性能逐渐下降, 多台虚拟机的整体算法性能超过宿主机算法性能, 并缓慢增加最后逐渐保持稳定.

6 结语

本文设计实现了基于 I/O 前后端模型的密码卡软件虚拟化方式, 利用共享内存或者 virtio 作为通信方式, 通过重新设计密码卡前后端驱动或者服务程序, 完成多虚拟机与宿主机的高效通信, 实现常规密码卡被多虚拟机共享, 提出的方法降低了云密码机的硬件门槛, 以软件的方式有效地解决了使用 SR-IOV 密码卡过程中存在的兼容性不好、扩充性受限、迁移性差以及性价比低等问题, 为密码卡的软件虚拟化研究提供了很好地借鉴思路.

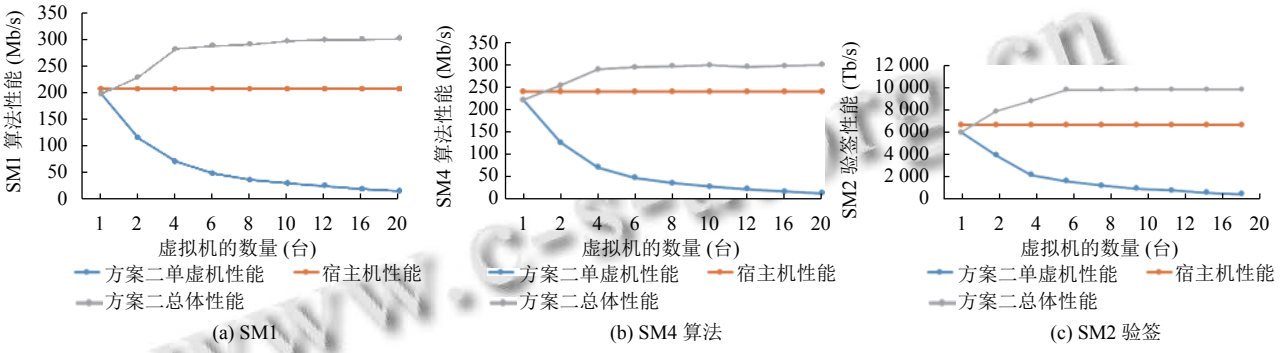


图 13 ARM 平台下 SM1、SM4 与 SM2 算法性能测试图

实验结果证明, 两种方案在各自平台的性能表现方面很好, 甚至在多虚拟机并发调用密码卡时可以获得超过宿主机下调用密码卡的性能, 完全能够满足实际应用的需求. 本文提出的技术不局限于云密码机产品的开发, 对于使用 SR-IOV 密码卡的服务器或者云主机同样适用, 具有很好的使用前景, 目前方案二已应用于信创项目的云密码机产品.

当然, 本文提出的方法还有进一步改进的空间, 根据两种不同方案在不同数据包长下宿主机与虚拟机通信性能的表现, 通过负载均衡组包来提升本文方案的算法性能是值得研究的方向. 同时, 与密码卡的 SR-IOV 硬件虚拟化相比, 本文提出的技术存在一些不足之处, 软件虚拟化不能从硬件层次屏蔽各虚拟机对卡内密钥的访问, 只能在软件层次设置密钥的访问权限.

参考文献

1 高志权. 高性能高安全的密码机研究. 信息安全与通信保密, 2019, (11): 28–35.
2 赵云, 庞振江, 夏信, 等. 适用于云计算的密码机技术研究. 信息安全与通信保密, 2016, (3): 103–105, 109. [doi: 10.3969/j.issn.1009-8054.2016.03.034]
3 Dong YZ, Yang XW, Li JH, *et al*. High performance network virtualization with SR-IOV. Journal of Parallel and Distributed Computing, 2012, 72(11): 1471–1480. [doi: 10.1016/j.jpdc.2012.01.020]
4 Berger S, Cáceres R, Goldman KA, *et al*. vTPM: Virtualizing the trusted platform module. Proceedings of the 15th Conference on USENIX Security Symposium. Vancouver: USENIX Association, 2006. 21.
5 杨永娇, 严飞, 于钊, 等. 一种基于 VT-d 技术的虚拟机安全隔离框架研究. 信息网络安全, 2006, (11): 21–14. [doi: 10.3969/j.issn.1671-1122.2006.11.009]
6 Sun L, Wang ZW, Sun RC. Research and design of crypto card virtualization framework. Proceedings of 2016 International Conference on Wireless Communication and Network Engineering (WCNE2016). Lancaster, Pennsylvania: DEStech Publications, 2016. 311–319.
7 马龙宇. 基于 SR-IOV 虚拟化技术高速密码卡的设计与实现 [硕士学位论文]. 上海: 上海交通大学, 2016.
8 张嘉夫. 基于密码卡虚拟化的虚拟桌面隐私保护研究 [硕士学位论文]. 武汉: 华中科技大学, 2017.
9 任继奎, 林山, 叶云峰. 基于 FPGA 的 SR-IOV 技术研究与实现. 通信技术, 2019, 52(9): 2321–2324. [doi: 10.3969/j.issn.1002-0802.2019.09.044]
10 苏振宇. 密码卡虚拟化技术研究与实现. 集成技术, 2019, 8(3): 31–41. [doi: 10.12146/j.issn.2095-3135.20181113001]
11 徐宇. 受控直通技术在设备内存虚拟化中的研究与优化 [硕士学位论文]. 上海: 上海交通大学, 2018.
12 Abramson D, Jackson J, Muthrasanallur S, *et al*. Intel

virtualization technology for directed I/O. Intel Technology Journal, 2006, 10(3): 179–192.

13 Morgan B, Alata É, Nicomette V, *et al*. IOMMU protection against I/O attacks: A vulnerability and a proof of concept. Journal of the Brazilian Computer Society, 2018, 24(1): 2. [doi: 10.1186/s13173-017-0066-7]

14 金海. 计算系统虚拟化——原理与应用. 北京: 清华大学出版社, 2008.

15 李玉玲. 云计算环境下密码算法模型的研究与实现 [硕士学位论文]. 济南: 山东大学, 2016.

16 李强, 涂二看见, 张明. 基于共享内存的测试指挥显示系统进程数据同步技术研究. 计算机应用与软件, 2020, 37(12): 13–16, 75. [doi: 10.3969/j.issn.1000-386x.2020.12.003]

17 Russell R. Virtio: Towards a de-facto standard for virtual I/O devices. ACM SIGOPS Operating Systems Review, 2008, 42(5): 95–103. [doi: 10.1145/1400097.1400108]

18 刘禹燕, 牛保宁. 半虚拟化框架 Virtio 的网络请求性能优化. 小型微型计算机系统, 2018, 39(1): 105–110. [doi: 10.3969/j.issn.1000-1220.2018.01.022]

19 El Amri A, Meddeb A. Optimal server selection for competitive service providers in network virtualization context. Telecommunication Systems, 2021, 77(3): 451–467. [doi: 10.1007/s11235-021-00764-3]