華中科技大學

《PVM: Efficient Shadow Paging for Deploying Secure Containers in Cloud-native Environments》

姓　　名：　　方子豪

学　　院：　　计算机科学与技术

专　　业：　　计算机科学与技术

班　　级：　　CS2008

学　　号：　　U202215628

课程名：　　云计算与虚拟化

指导教师：　　吴松

2025 年 6 月 4 日

前言

本学期我唯一选修的课程是《云计算与虚拟化》，我对这门课程给予了极高的重视。通过老师在课程上的讲解，我对虚拟机的运行机制有了整体性的认知，特别是对虚拟化如何实现资源隔离与共享的原理有了较深理解。同时，课程中对 Docker 容器技术的讲解让我初步建立了容器与传统虚拟化方式的对比视角。虽然这门课的内容专业性很强，但我通过积极思考和资料查阅，拓展了自己在云平台架构方面的认知，对未来的技术发展方向也有了更多思考。

在吴松老师的主页翻阅了众多文章之后，我最终选择了发布在第 29 届 Symposium on Operating Systems Principles（*SOSP '23*）的会议记录中的文章《*PVM: Efficient Shadow Paging for Deploying Secure Containers in Cloud-native Environment.*》。一方面原因是这篇文章发表在顶级会议 *SOSP* 上，足以证明其含金量，另一方面在于我本人对虚拟化的技术也比较感兴趣，很想知道最前沿的虚拟化技术关注的方向是什么。

阅读体会

论文总体上提出了 PVM，一种基于页表的高性能嵌套虚拟化框架，构建于 KVM 虚拟机监控器之上。具体的目的是将安全容器托管与宿主机虚拟机监控器和硬件虚拟化支持完全解耦，以实现以下两个目标：

（1）在不影响云平台安全性、灵活性和复杂性的前提下，实现与任何 IaaS 云的嵌套虚拟化

（2）避免代价高昂的退出到宿主机虚拟机监控器，并设计高效的 world switching 机制

为了能够细致描述 PVM 所做的具体工作，我将沿着论文本身的脉络来阐述我的理解，在此之前需要进行一些前置知识的准备。

Prerequisite knowledge

- 为什么虚拟化很重要？

虚拟化之所以重要，是因为它**极大地提升了资源利用率、灵活性和可管理性**，在现代 IT 基础设施、云计算和数据中心中扮演着关键角色。虚拟化允许在一台物理机上运行多个虚拟机，让资源“吃满”，避免浪费。同时能够用更少的硬件设备达成要求服务器整合后，IT 基础设施更加简洁、节能、环保。相关优点很多，在此就不再赘述。这些优点在前几学期的必修课中涉及到虚拟机的部分我已深有体会，代码运行环境可以和我电脑本机的环境分离开来可以极大程度上避免崩溃带来的问题。同时在实验室的服务器使用上，虚拟化也能支持多个人同时访问服务器利用不同资源进行实验，不可谓不重要。

- 什么是 Hypervisor？

Hypervisor 可以说是虚拟化中重中之重的一个关键层。有时也称作虚拟机监视器，简单的说 Hypervisor 是管理虚拟机的“控制器”，协调硬件资源的分配，让多台虚拟机共享同一套物理资源。

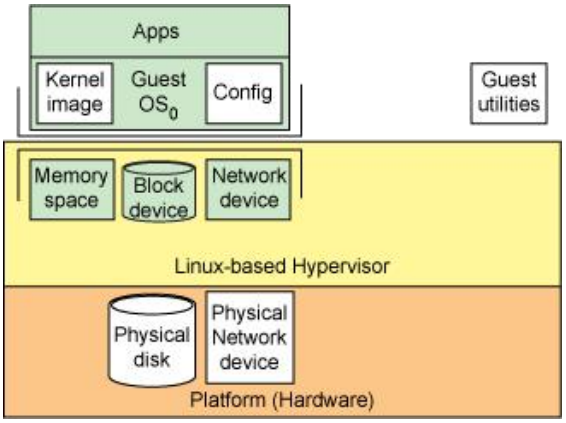

图 1　Hypervisor 示例图

- KVM 是什么？

KVM，即 Kernel-based Virtual Machine，是 Linux 内核的一种虚拟化技术，把 Linux 操作系统本身变成了一个功能完整的 Hypervisor。它使用 Linux 内核的虚拟化模块，将物理服务器划分为多个虚拟机。在 KVM 中虚拟机被实现成一个 Linux 的进程，由 Linux 的调度进程进行调度，虚拟机的每个虚拟 CPU 被实现为一个常规的 Linux 进程。这使得 KMV 能够使用 Linux 内核的已有功能。

从 Linux2.6.20 其 KVM 作为内核的一个模块集成到了 Linux 的主要发行版本中，可见 KVM 最为当前最主流的开源服务器虚拟化技术的重要性。

- KVM 的缺点是什么？

既然想要改进，那么一定需要从缺点出发。本文中写到：Stateof-the-art nested virtualization in the x86 architecture relies heavily on the host hypervisor to expose hardware virtualization support to the guest hypervisor, not only complicating cloud management but also raising concerns about an increased attack surface at the host hypervisor.

也就是说问题主要在于**嵌套虚拟化**很大程度上依赖于宿主机 Hypervisor，这会使**云管理复杂化**同时可能会引起宿主机 Hypervis 受**攻击面增加的问题**。这也对应了一开始提出的两个改进目标——在不影响云平台**安全性、灵活性和复杂性**的前提下，实现与任何 IaaS 云的嵌套虚拟化；避免代价高昂的退出到宿主机虚拟机监控器，并设计高效的 world switching 机制。

Key idea of 2-level nested virtualization.

首先明确一点，PVM 是一种**针对嵌套内存虚拟化优化的 KVM 客户虚拟机管理程序，**是一个高性能的客户虚拟机 hypervisor，它对于宿主机 hypervisor 是透明的，并且不依赖于硬件虚拟化支持。PVM 利用了两个关键设计：

（1）客户虚拟机和客户虚拟机 Hypervisor 之间的一个最小共享内存区域，以方便不同特权级别之间的状态转换

（2）一种高效的影子页表设计，以降低内存虚拟化的成本。

既然要针对嵌套内存虚拟化进行优化，分析当前最先进的嵌套虚拟化方法是必不可少的一步。在本论文中得到的结论如下：虽然硬件虚拟化相对于软件虚拟化在单层虚拟化的前提下有着更卓越的性能和更低的开销，但是在**双层嵌套虚拟化中可能会导致次优的性能和过多的 trap**（这里我理解为用户态到内核态的陷入，即状态切换，后文中也确实描述为了 world switching）到宿主机管理程序。同时我查阅了论文中提到的相关的论文《*A comparison of software and hardware techniques for x86 virtualization*》，该论文中比较了软件和硬件虚拟化以及他们在单层虚拟化中的开销问题。得到的传统解决方案是**混合虚拟化方法，**也就是从客户机虚拟地址到客户机物理地址（L2）到 VM 物理地址（L1）到宿主机物理地址（L0）的多路复用。传统方法和 KVM 中的方法略有不同但是大体上都涉及到过于繁杂的 world switch 因此需要进行改进。

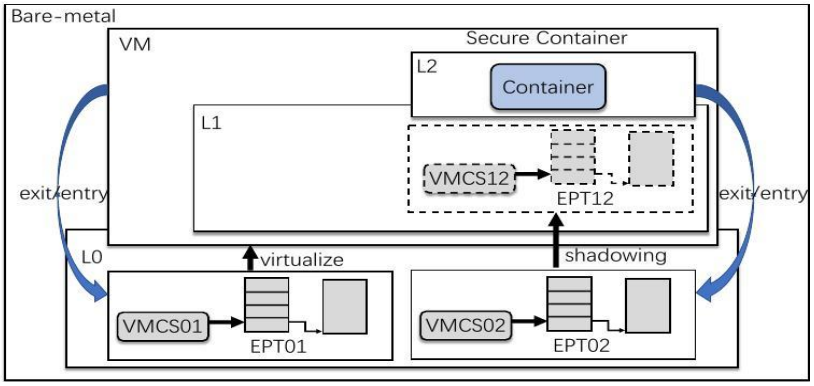

图 2 具有嵌套虚拟化的安全容器示例

Key ideas of PVM memory virtualization

分析完了嵌套虚拟化的思路，就该来考虑内存虚拟化的问题了。传统的内存虚拟化方式，虚拟机的每次内存访问都需要 VMM 介入，并由软件进行多次地址转换，其效率是非常低的。因此才有了影子页表技术和 EPT 技术。

在下面的论述中沿用论文中原本的代称：

L2 —— 客户机虚拟地址　L1—— VM 物理地址　L0 —— 主机物理地址

由于 PVM 是基于 L1 虚拟机中实现的 Hypervisor，因此与 L0 中的 KVM 技术无关。而在具体实现中，为了完全控制 L2 中的内存虚拟化，PVM 采用基于软件的**影子分页技术**将 L2 虚拟地址转化为 L1 物理地址，这也就是我下文提到的**L2 的含义从客户机物理地址变为客户机虚拟地址**的根本原因。后续依靠硬件 EPT 或者 NPT 将剩余的转换转换为 L0。所以这允许 PVM 与未修改的 KVM 主机虚拟机管理程序一起工作，并允许嵌套虚拟机和普通虚拟机共存。

The key idea of PVM nested memory virtualization

PVM 的具体思路也没有脱离混合软硬件的底层设计。在对于 L2 和 L1 的转换中使用基于软件的影子分页将 L2 转换为 L1，在后续转换中依赖硬件 EPT 或 NPT 进行剩余的转换，将其转换为 L0。不过相对于上面分析的传统嵌套虚拟化方案有一个明显的区别，**这里的 L2 表示为客户的虚拟机地址，也就是说客户的物理机地址这一个中间层是被省略掉了或者说可以被化简掉。（上文中有提到，其实是基于影子页表的方案，简化了地址转换的过程）**

由于 L2 转换为 L1 和将 L1 转化为 L0 分别用**软件**和**硬件**实现，其中 L1 转换为 L0 也是基于原本的未经修改的 KVM 主机虚拟机实现的，因此 PVM 一个很大的优点在于其能够与未经修改的 KVM 一起工作同时允许嵌套虚拟机和普通虚拟机的共存。另一个优点则是 L2 转换为 L1 这种基于软件模拟的嵌套 CPU 和内存虚拟化有效的将 L2 客户隔离在 L1 虚拟机中，确保了安全性。

在后续的论文中提到，PVM 证明了对于嵌套虚拟化 PVM 的软硬件结合方法显著由于仅由硬件辅助的方法，基于这一点 PVM 实现了三个重要的设计，也就是对上文中提到的**两个关键设计**的具体阐述：

（1）将 L2 客户机的用户和内核空间完全放置在 L1 虚拟机的特权级别 3（Ring 3）中，以确保所有特权指令、系统调用和异常都陷入 PVM 进行模拟。这种设计在最新的 AMD 处理器和 Intel 的 x86-s 架构中是必要的，因为它们都移除了 Ring 1 和 2，以简化 CPU 指令集。

（2）一段高效的汇编代码，称为切换器，驻留在 L2 客户机和 L1 虚拟机之间的共享内存区域中，以方便 L2 客户机的用户和内核空间以及 PVM（L1 虚拟机监控器）之间的 world switching。

（3）一种由 PVM 的嵌套虚拟化实现的、高效且可扩展的影子页表设计。PVM 具有并行 SPT 和预先缺页机制，以加速客户机页错误的处理。

PVM: Design and Implementation

PVM 的总体架构如下图所示：

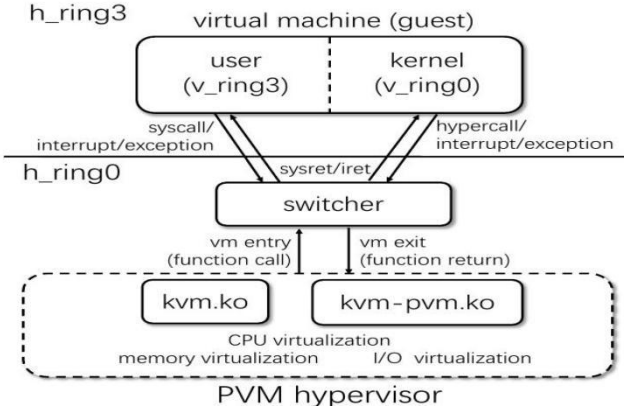

图 3 PVM 的架构图

该架构包含三个关键组件：L2 客户机，切换器和 PVM 虚拟机管理程序。

- 对于 **L2 客户机**：可以看到 L2 完全运行在 Ring3 中，仅具有最低的权限。
- 对于**切换器（switcher）**：可以看到切换器负责在 L1 的 Hypervisor，L2 客户机用户和 L2 客户机内核之间切换“world”，值得一提的是 PVM 在切换器中设计了一种新颖的直接开关机制，允许 L2 客户机通过系统调用实现快速的用户/内核切换，而无须 Hypervisor 的干预。这也是我们之前提到的一大优化之一。
- 对于 **PVM 的管理程序本身**：可以看到框图中包含两个重要模块 kvm.ko 和 kvm-pvm.ko。可以理解的是 kvm.ko 集成的是原本的 kvm 的基础功能，而 kvm-pvm.ko 则在 kvm 的基础上集成了 pvm 的功能。该部分采用了基于指令仿真和半虚拟化的混合方法进行 CPU 虚拟化、基于软件的方法（影子页表 SPT）进行内存虚拟化，以及仅依靠 L0 向 L1 注入中断然后采用基于软件的中断虚拟化。

最后论文经过实验也证实了 PVM 对于所有应用程序，提供的性能与单层虚拟卷的硬件辅助方法接近，当并发性较高时，kvm-ept 的性能会由于 L0 的 Hypervisor 的性能瓶颈而崩溃，但是相比之下 PVM 实现了始终如一的良好性能，并且在许多情况下能做到接近于单层虚拟化的性能。

The limitations of PVM

虽然 PVM 通过软硬件结合的嵌套虚拟化方法，在 KVM 的基础上进行了很大的改进并且有了较为显著的结果，但是 PVM 依旧是有一些缺陷和局限性的。

（1）首先是 PVM 的安全性问题：在 PVM 中运行的安全容器的假设为——信任基于硬件的保护，但是软件的缺陷可能通过暴露的接口（如系统调用等）被利用。论文中提供的例子为假设一个恶意租户（L2）中运行可能试图通过破坏暴露的接口并利用其主机内核或虚拟机监控器（L1 的 Hypervisor）的漏洞来突破边界导致信息泄露等问题。虽然 PVM 比直接部署在主机上并共享主机内核的传统容器更加安全，但这并不代表 PVM 的安全性就一定得到了保障，还是有突破安全性的可能。

（2）其次由于 PVM 是软硬件结合的嵌套虚拟化方法，有基于软件的虚拟化部分，会随着某些工作负载的提高而引入相当大的开销。在论文中引入的例子为 fork 和许多小内存区域分配会导致大量的 L2 客户机页面错误从而导致 PVM Hypervisor 来处理这些错误从而引发极大开销。而相对的硬件辅助方式处理这些页面错误则会简单很多。

总结

（1）技术认知的深化与体系构建

通过对《***PVM: Efficient Shadow Paging for Deploying Secure Containers in Cloud-native Environments***》的系统性研读，本人对嵌套虚拟化技术的演进脉络与前沿方向形成了立体化认知。论文中提出的 PVM 框架以软硬件协同优化为核心，突破了传统嵌套虚拟化对宿主机 Hypervisor 的强依赖困境，其通过影子页表创新设计与高效状态切换机制，在保障性能的同时实现了安全容器与底层硬件的解耦。这一技术路径不仅为云原生环境下多租户隔离提供了新范式，更揭示了虚拟化领域 "性能 - 安全 - 兼容性" 三角平衡的关键矛盾 —— 例如 PVM 通过软件模拟实现 L2 客户隔离，虽牺牲部分硬件加速性能，却换来了与原生 KVM 的兼容性及安全边界的清晰化，这种取舍思维对理解复杂系统设计具有重要启发。

在知识体系构建层面，论文研读过程促使本人完成了从 "碎片化概念" 到 "系统化框架" 的认知升级。以 Hypervisor 为例，通过对比 Type 1（如 ESXi）与 Type 2（如 VirtualBox）Hypervisor 的架构差异，深刻理解了其在虚拟机调度、资源分配中的核心作用。这种理论认知直接解释了实践中实验室服务器（基于 Type 1 Hypervisor 的裸金属虚拟化）与个人电脑虚拟机（基于 Type 2 Hypervisor 的宿主虚拟化）在连接流程、性能表现上的同源性 —— 二者均通过 Hypervisor 层实现物理资源抽象，尽管部署场景不同，但本质上均遵循 "虚拟地址 - 物理地址 - 主机地址" 的多层地址转换逻辑。

（2）研究方法论的启发与实践导向

论文研读暴露出的 "无源码复现困境"，本质上反映了学术研究与工程实践的天然鸿沟，但也倒逼本人形成更务实的技术探索思维：

PVM 的成功源于对嵌套虚拟化核心痛点的精准定位——传统方案中 L0-L1-L2 的多层硬件依赖导致攻击面扩大与性能损耗。这启示在未来研究中，需始终以 "解决实际问题" 为导向，跨越软硬件层隔，从体系结构视角寻求破局点。例如在容器安全领域，可借鉴 PVM 的 "最小共享内存 + 影子页表" 思路，设计轻量级内存隔离机制，平衡安全增强与性能损耗。论文对 PVM 局限性的坦诚分析（如软件虚拟化引入的 fork 操作开销、依赖硬件保护的安全假设），彰显了学术研究应有的客观理性。这种思维要求在吸收前沿技术时，需建立 "优势 - 缺陷 - 场景适配" 的三维评估框架。以 PVM 为例，其在高并发场景下的稳定性优势适用于微服务密集型云平台，但在 IO 敏感型负载中可能因软件模拟开销反而不及传统硬件辅助方案。

为理解论文中 "影子页表并行化"" 预缺页机制 "等关键技术，本人系统梳理了内存虚拟化的发展脉络：从早期纯软件模拟的影子页表（Shadow Page Table），到硬件辅助的 EPT/NPT 技术，再到 PVM 提出的软硬混合方案。这一过程构建了" 问题提出 - 技术演进 - 方案优化 "的认知链条，提示在后续学习中需定期对知识体系进行" 版本迭代 "，通过追踪顶会论文（如 SOSP、OSDI）与开源项目（如 KVM 社区），保持对技术前沿的敏感度。

（3）未来研究的方向锚定

对于 PVM 的技术框架，我觉得后续探索可能有以下几种方向：

- **性能优化维度：**针对论文指出的 "小内存分配导致页错误激增" 问题，

可研究基于机器学习的页错误预测模型，通过历史访问模式预判缺页风险，提前触发预加载机制，降低软件处理开销。

- **安全增强维度：**鉴于 PVM 对硬件保护的依赖，可引入可信计算技术（如 TPM），构建 "硬件隔离 + 软件验证" 的双重安全屏障，抵御针对共享内存接口的攻击。
- **场景拓展维度：**将 PVM 的嵌套虚拟化思路迁移至边缘计算场景，设计轻量化 Hypervisor 以支持 "边缘节点 - 云中心" 的分级虚拟化架构，满足物联网设备的资源隔离与弹性调度需求。

（4）结语

本次论文研读不仅是一次专业知识的深度学习，更是一次研究思维的系统训练。PVM 的论文揭示了计算机系统领域 "复杂问题拆解 - 关键路径创新 - 工程实现验证" 的标准研发范式，而其未竟的技术挑战（如解决 PVM 的相关痛点）则化为持续探索的动力。在未来的学术道路上，本人将秉持 "从实践中发现问题，在理论中寻找方案，于验证中迭代优化" 的研究理念，以更开放的技术视野与更严谨的批判思维，投身于所在领域的创新实践。