计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20180720
56(1):58-68, 2019

操作系统虚拟化的研究现状与展望

吴 松 王 坤 金 海
(服务计算技术与系统教育部重点实验室(华中科技大学) 武汉 430074)
(集群与网格计算湖北省重点实验室(华中科技大学) 武汉 430074)
(华中科技大学计算机科学与技术学院 武汉 430074)
(wusong@hust.edu.cn)

Research Situation and Prospects of Operating System Virtualization

Wu Song, Wang Kun, and Jin Hai
(*Services Computing Technology and System Lab* (*Huazhong University of Science and Technology*), *Wuhan* 430074)
(*Cluster and Grid Computing Lab* (*Huazhong University of Science and Technology*), *Wuhan* 430074)
(*College of Computer Science and Technology*, *Huazhong University of Science and Technology*, *Wuhan* 430074)

Abstract As a kind of lightweight virtualization technology, container has not only been widely used in resource management and DevOps of cloud computing platform and data center in recent years, but also gradually applied to some new fields such as edge computing and Internet of things. Container has shown a good development trend and application prospect. So, operating system virtualization as a core technology of container has received widespread attention in both industry and academia. Operating system virtualization allows multiple applications to run in a set of isolated runtime environment by sharing the same host operating system kernel. It has the advantages of fast startup, convenient deployment, low resource consumption, high running efficiency. However, there are also deficiencies such as weak isolation. And the deficiency has become a research hotspot in the field of virtualization. In this survey, we first introduce the technical architecture of operating system virtualization and compare it with traditional virtualization technology to summarize its characteristics. Then we analyze the current research status of operating system virtualization from container instance layer, container management layer and kernel resource layer. Finally, the paper lays out several challenges and research prospects of operating system virtualization.

Key words operating system; container; virtualization; container management; resource isolation

摘 要 容器技术作为一种轻量级虚拟化技术,近年来不仅广泛应用于云计算平台和数据中心的资源管理、系统运维和软件部署中,也逐步应用于包括边缘计算、物联网等在内的新领域,表现出了良好的发展态势和应用前景.在此背景下,操作系统虚拟化作为容器的核心技术引起了广泛的关注.操作系统虚拟化允许多个应用在共享同一主机操作系统内核的环境下隔离运行,具有启动快速、部署方便、资源占用少、运行效率高等优点,但是也存在隔离性较弱等不足之处,后者也成为了虚拟化领域的研究热点.首先

收稿日期:2018-11-01;**修回日期**:2018-11-27
基金项目:国家重点研发计划项目(2016YFB1000501);国家自然科学基金项目(61732010,61872155);中央高校基本科研业务费专项资金(HUST:2016YXZD016)
This work was supported by the National Key Research and Development Program (2016YFB1000501), the National Natural Science Foundation of China (61732010, 61872155), and the Fundamental Research Funds for the Central Universities (2016YXZD016).

介绍操作系统虚拟化的历史背景和技术架构,并与传统虚拟化技术对比总结操作系统虚拟化技术的特点;随后分别从容器实例层、容器管理层和内核资源层梳理和分析操作系统虚拟化当前研究现状;最后阐述了操作系统虚拟化领域的技术挑战和研究展望.

关键词　操作系统;容器;虚拟化;容器管理;资源隔离

中图法分类号　TP316

近年来,容器技术广泛应用于云计算平台和数据中心的资源管理、系统运维和软件部署中,成为了云计算领域的业界热点.作为容器代表系统之一的Docker[1],从其2013年诞生以来便获得了持续关注和快速发展,相关开源社区的活跃度极高,大批IT企业实际部署.同时,作为业界最具代表性的云计算平台,亚马逊云、谷歌云、微软云和阿里云等都先后在云计算业务中推出以操作系统虚拟化技术为基础的容器云服务.最近操作系统虚拟化技术开始走出数据中心,除了传统的微服务(micro service)、软件开发运维(DevOps)等领域,已逐步应用于包括边缘计算、物联网等在内的新领域,表现出了良好的发展态势和应用前景.

在这种背景下,操作系统虚拟化作为容器的核心技术支撑,得到了研究者的广泛关注.最近几年,无论是在以SOSP/OSDI为代表的计算机系统领域顶级学术会议上,还是以Google为代表的重要互联网企业中,都陆续出现了一批操作系统虚拟化的最新研究成果,并且成果数量呈现出逐年增加的总体趋势.操作系统虚拟化及其支持的容器技术已逐渐成为了研究热点.本文通过对操作系统虚拟化代表性研究工作涉及的主要研究内容以及技术方法进行梳理;继而对操作系统虚拟化可能存在的问题进行分析和总结;在此基础上,展望操作系统虚拟化未来的研究方向.

1　操作系统虚拟化的历史、架构和特点

虚拟化是一种资源管理技术,通过对计算机资源的抽象和模拟,提供一个个隔离执行环境,即虚拟机.虚拟化可以实现计算机资源的动态分配、灵活调度、有效共享,提高计算机资源利用率.虚拟化技术分布在计算机系统的不同层次.应用在云计算平台和数据中心中的虚拟化技术主要包括传统虚拟化技术和操作系统虚拟化技术.在本文中,传统虚拟化技术是指以Xen和KVM等系统为代表的硬件虚拟化技术.由于近年来操作系统虚拟化技术获得了广泛的关注并表现出了良好的发展趋势,本文将重点讨论操作系统虚拟化技术.本节首先简单介绍操作系统虚拟化历史背景和技术架构,然后通过将操作系统虚拟化技术与传统虚拟化技术进行比较从而总结其优缺点.

1.1　操作系统虚拟化的历史背景

操作系统虚拟化技术有近40年的历史.思想起源于1979年UNIX系统中的chroot技术[2],该技术可以切换系统的根目录位置,从而实现简单的文件系统资源隔离.2000年FreeBSD Jails[3]吸收并改进了chroot技术,在文件系统隔离的基础上添加了用户和网络资源的隔离.2004年Solaris Container[4]作为Solaris 10中的特性发布,包含了Resource Management提供的系统资源管理和Zones提供的运行环境隔离.2005年OpenVZ[5]采用了和Solaris Container类似的思想,通过为内核打补丁的方式来提供资源管理和隔离等功能.2007年Cgroup(control group)[6]被加入Linux内核中,实现了统计和控制每个进程的资源使用(包括CPU、内存、I/O、网络等)功能.2008年LXC[7]基于Cgroup和Linux namespace[8]实现,进一步完善了操作系统虚拟化技术.2013年Docker围绕操作系统虚拟化技术构建了一套完整的生态,包括镜像标准、REST API、CLI、集群管理等,具有配置简化、部署便捷、效率更高等优点,使得操作系统虚拟化技术得到了更加广泛的应用.

1.2　操作系统虚拟化的技术架构

操作系统虚拟化技术允许多个应用在共享同一主机操作系统(Host OS)内核的环境下隔离运行,主机操作系统为应用提供一个个隔离的运行环境,即容器实例.操作系统虚拟化技术架构可以分为容器实例层、容器管理层和内核资源层.图1展示了Docker系统架构对应各层次的划分.

容器实例层指所有运行于容器之上的应用程序以及所需的辅助系统,包括监控、日志等.容器管理层可以分为运行时引擎和管理程序2个模块,所起作用分别是运行容器应用和管理容器实例.内核资源层是指集成在操作系统内核中的资源管理程序,

例如 Linux 内核中的 Cgroup,其目标是对 CPU、内存、网络、I/O 等系统资源进行控制和分配,以支持上层的容器管理.

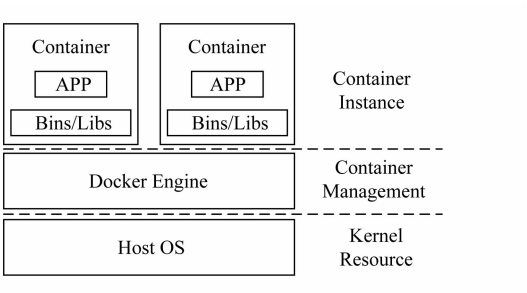

Fig. 1 Hierarchical structure of operating system virtualization

图 1 操作系统虚拟化层次架构图

1.3 操作系统虚拟化的主要特点

目前,云计算平台和数据中心中最常用的虚拟化技术是 Xen,KVM 等传统虚拟化技术和 Docker,LXC 等操作系统虚拟化技术. 因此,本节将操作系统虚拟化与传统虚拟化相比较以总结其主要特点.

操作系统虚拟化与传统虚拟化最本质的不同是传统虚拟化需要安装客户机操作系统(Guest OS)才能执行应用程序,而操作系统虚拟化通过共享的宿主机操作系统来取代 Guest OS. 因此容器启动时不需要等待操作系统加载,从而使得容器的启动速度远快于虚拟机的启动速度. 同时,与传统虚拟化相比较,操作系统虚拟化减少了虚拟机监控器(virtual machine monitor, VMM)和 Guest OS,所以在保证应用性能的同时,虚拟化的系统开销很小,并且有效地提升了系统资源利用率,使得在单一主机上同时运行数千个容器成为可能. 然而,传统虚拟化通过 VMM 层虚拟硬件资源以保证虚拟机之间的隔离性,而操作系统虚拟化只是利用内核的特性实现了部分资源的限制,因此传统虚拟化相较于操作系统虚拟化拥有更强的隔离性和安全性. 表 1 对操作系统虚拟化技术与传统虚拟化技术的特性进行了比较.

Table 1 Operating System Virtualization vs Traditional Virtualization

表 1 操作系统虚拟化与传统虚拟化的比较

Features	Operating System Virtualization	Traditional Virtualization
Startup	Second-Level	Minute-Level
Resource Utilization	High	Low
Performance	Close to Native	Worse than Native
Deployment Density	Thousands	Tens
Isolation	Weak	Strong
Running Form	Running on Host OS	Running on Hypervisor

基于以上特性,操作系统虚拟化技术和传统虚拟化技术都存在各自比较适用的应用场景. 操作系统虚拟化适用于追求高性能且对隔离性及安全性要求不高的场景,例如在进程多、更新快的微服务以及环境配置变化多、更新频繁的 DevOps 场景中,操作系统虚拟化可以发挥出其镜像小、启动快、部署便捷的优点. 传统虚拟化适用于对隔离性要求较高或者应用程序需要运行于不同操作系统的场景,例如面向多租户的云服务提供商多数是采用传统虚拟化技术以保证用户性能和安全需求. 在一些特定场景中,也会让容器运行在传统虚拟机上,用传统虚拟化的良好隔离性来弥补操作系统虚拟化隔离性的不足,例如现在主流的云计算平台皆采用这种方式提供容器服务.

综上所述,操作系统虚拟化具有启动快速、部署方便、资源占用少、运行效率高等优点,但也存在隔离性较弱的不足,因此适用于追求高性能但是对隔离性要求不高的应用场景.

2 操作系统虚拟化的研究现状

随着操作系统虚拟化技术的快速发展和广泛应用,诞生了大量相关的研究工作. 本节将分别从容器实例层、容器管理层和内核资源层介绍操作系统虚拟化相关研究工作从而梳理和分析其研究现状. 本文将介绍的代表性研究工作所涉及的研究内容如表 2 所示.

2.1 容器实例层

目前针对操作系统虚拟化容器实例层的研究工作主要集中在容器应用场景适用性和容器应用性能分析 2 个方面,从这 2 个方面梳理应用层的研究现状.

Table 2　Research Concerns Status to Operating System Virtualization

表 2　操作系统虚拟化研究内容统计

Research Work	Container Instance		Container Management		Kernel Resource	
	Application Scenarios	Performance Analysis	Image Management	Container Orchestration	Resource Isolation	Security Isolation
Docker[1]			√	√	√	√
FreeBSD Jails[3]					√	√
Solaris container[4]					√	√
Contain-ed[9]	√	√				
Elastic resource provision[10]	√					
Container-based emulation[11]	√	√				
Container and microservice[12]	√	√				
Elatic container[13]	√	√				
Container-based edge cloud[14]	√	√				
Skyport[15]	√	√				
Container-based mobile cloud[16]	√	√				
Customization mobile cloud[17]	√					
SOCK[18]	√				√	√
Container and cloud[19]	√	√				
Performance survey[20]	√	√				
Isolation analysis[21]		√			√	
HPC performance[22]		√				
I/O performance analysis[23]	√	√				
Comparison of VM and container[24]		√				
Container network[25]		√				
Performance study[26]		√				
Slacker[27]			√			
Wharf[28]			√			
Docker registry design[29]			√			
Essential docker for ASP. NET[30]				√		
Omega[31]				√		
Borg[32]				√		
Mesos[33]				√		
Container-based OSV[34]	√	√			√	√
OS virtualization[35]					√	√
Two-Tiered Approach to I/O[36]					√	
Dune[37]						√
Nested kernel[38]						√
okernel[39]						√
SCONE[40]						√
Advances on virtualization[41]	√	√				
Quantify performance[42]		√			√	√

Notes: In this table, check mark "√" indicates the specific concern is discussed in corresponding publication.

1) 应用场景

操作系统虚拟化技术的出现、成熟和走向主流,对于应用而言,意味着除了传统虚拟化技术之外又多了一种运行环境的选择.从技术特点来看,操作系统虚拟化技术能在一些应用场景中提供快速部署能力并提高资源利用率.

操作系统虚拟化技术的传统应用场景包括微服务、DevOps、云环境下的资源管理等.例如 Sheoran 等人[9]利用基于容器的微服务系统解决分布式部署带来的端到端延时问题;然而,在微服务场景中也存在资源供给时效性差难以应对负载突变的问题,郝庭毅等人[10]提出了一种服务质量敏感的、基于前馈的容器资源弹性供给方法可有效保障应用的服务质量.针对自动化开发与测试应用场景,Handigol 等人[11]基于容器技术开发网络系统模拟器,使得对以往研究成果的实验复现变得更加简便.Kang 等人[12]利用容器技术和微服务架构在云计算环境中实现基础设施的自动化运维.He 等人[13]在云计算环境中基于容器技术实现弹性的资源管理策略,相较于基于传统虚拟化的方案有效提升了资源灵活性以及资源利用率.

操作系统虚拟化也逐步应用到一些新型应用场景之中,包括边缘计算、移动云、无服务器计算等应用场景.例如 Pahl 等人[14]和 Gerlach 等人[15]分别利用容器技术来解决边缘计算平台和科学计算平台的软件部署以及资源利用率低下的问题.但是在容器技术应用的过程中,也存在容器必须运行相同操作系统的局限性.针对该问题,吴松等人[16]通过在主机操作系统内核中实现安卓容器驱动以打破操作系统虚拟化的内核限制,从而构建启动快速、系统资源开销较低的云端安卓代码卸载运行环境.Hu 等人[17]在以上研究的基础上提出基于容器技术的安卓内核定制工具 AndriodKit,以适应云端大规模、多样化的应用场景.针对物联网等新型应用场景,OpenLambda[43]基于容器技术实现比微服务架构更细粒度的无服务器架构,SOCK[18]在此基础上根据无服务器架构的需求针对容器的初始化延迟和可扩展性进行了优化.

现有的针对操作系统虚拟化应用场景的研究涉及领域比较广泛,从传统的云计算平台到最新的无服务器架构都存在相关的研究工作.由于操作系统虚拟化具有良好的性能并且正在快速发展,因此未来将会有更多的关于容器应用场景的研究工作诞生.

2) 性能分析

现有针对操作系统虚拟化性能分析的研究主要是与传统虚拟化环境下的性能做比较,以总结操作系统虚拟化的优缺点.在针对吞吐量等应用性能的研究中,结果均显示应用运行在容器环境中较运行在传统虚拟化环境中有一定的性能增长.例如,Bernstein[19]分别在容器环境和传统虚拟机环境中测试 NewSQL 系统性能,结果显示容器环境应用性能较传统虚拟机环境有 5 倍的增长.Plauth 等人[20]将 HTTP 服务器和键值存储应用程序分别运行于 Docker 和 XEN 环境中,测试结果显示在吞吐量等应用性能方面,Docker 明显优于 XEN.在针对隔离性等系统性能的研究中,作者也都得结论:容器系统隔离性相较传统虚拟化更弱.例如,Xavier 等人[21]分别在 LXC 和 KVM 上对磁盘敏感的负载进行压力测试,得出结论:LXC 的 I/O 隔离远弱于 KVM;他们的另一项研究[22]证明在高性能计算(HPC)环境中,虽然容器技术的性能开销较小,但是其隔离性目前还不成熟,实验结果显示只有 CPU 资源能做到较好的隔离,内存、磁盘和网络等资源的隔离性均较弱.Guedes 等人[23]分别在容器和传统虚拟化环境中运行 I/O 密集型的应用,结果表明容器环境虽然有更短的应答时间、更小的失败率,但是其隔离性不如传统虚拟化强.Felter 等人[24]在 Docker 和 KVM 上分别对 CPU、内存、I/O 做压力测试,也通过实验得出 Docker 隔离性更弱的结论.

还有一些研究工作针对网络、I/O 等容器系统性能做出详细的测试与分析.例如,Suo 等人[25]分别详细分析了单主机和多主机环境下各种容器网络连接模式的性能优缺点.在单主机和多主机环境下都存在安全隔离和网络性能的权衡,因此容器系统为追求良好的安全隔离都会损耗网络性能.另外,各种容器网络模式的启动时间也存在差别,因此短生命周期的应用和延迟敏感的应用在选择容器网络模式时都应将启动时间考虑在内.阮博文等人[26]将现有的容器系统分为应用容器(例如 Docker)和系统容器(例如 LXC)2 类,通过一系列实验得到结论:系统容器比应用容器更适合 I/O 密集型应用场景,这是因为应用容器中的分层文件系统会增加 I/O 延迟.

综上所述,在对操作系统虚拟化与传统虚拟化应用性能比较的研究中,均可得出结论:操作系统虚拟化技术具有较高的性能效率,但是隔离性仍存在不足.在对容器系统测试与分析的研究中,可以发现容器系统的不同网络模式以及不同容器系统之间

都存在性能差异,因此运行在容器环境中的应用程序应根据自身性能需求进行网络模式配置与容器系统选择.

2.2　容器管理层

目前针对操作系统虚拟化容器管理层的研究工作主要集中在容器镜像管理和容器编排2个方面.具体研究现状如下.

1) 镜像管理

容器的启动时间对于容器应用性能有很重要的影响,如果容器启动时间更快的话,很多工作将会因此受益.例如集群调度器将会以一个更小的开销频繁地进行负载均衡;软件也可以更加快速地更新来解决重要bug等.然而相关实验结果表明,76%的容器启动时间是花费在镜像拉取这个操作上面[27].因此,对于容器镜像管理的研究具有重大意义.Slacker[27]通过分析发现容器启动时必需的数据仅占容器镜像的6.4%,从而采取优先读取必需镜像数据、延迟读取不重要镜像数据的方案以加快容器启动速度;并且通过修改Docker容器的存储驱动,充分利用VMstore的snapshot和clone等基本API来实现写时复制功能.以上改进工作使得镜像上传和镜像拉取操作执行速度分别提升了153倍和72倍,有效降低了容器的启动和部署时间.然而,该工作并没有考虑大规模环境下,容器守护进程对于少量镜像的竞争访问问题.针对以上问题,Wharf[28]把容器存储驱动拆分为全局和私有部分,全局部分存储镜像层和元数据等容器守护进程之间共享的内容,私有部分存储网络和数据卷等私有信息,从而减少容器守护进程对于共享镜像的访问;对于共享的全局部分,也采取细粒度锁和元数据缓存等机制来实现容器守护进程间的同步问题,最终有效减轻了大规模分布式存储系统中容器镜像共享竞争问题.Anwar等人[29]通过分析大量实际生产环境中的镜像管理数据,总结镜像请求类型分布、镜像大小分布、各阶段响应时间、请求之间的关系、镜像请求的空间和时间局部性等特征,设计缓存和预取策略,提高镜像管理性能.

针对容器镜像管理的研究对于容器的发展具有重要意义,现有的该方面研究刚处于起步阶段,可能存在的研究问题和挑战将在第3节详细阐述.

2) 容器编排

容器编排是指对容器完整生命周期包括容器准备、部署和监控等多个方面的自动化管理.容器编排包括单主机的容器管理以及容器集群的管理.目前对于容器编排的研究已经相对比较成熟并在工业界得到广泛的应用.

现有针对单主机环境下容器管理的研究使用户对容器应用的操作更加方便.例如当需要部署一个Web应用时,首先需要启动数据库容器,然后再启动应用容器,最后可能还要启动反向代理容器.以上步骤需要3句命令才能部署而且不能把3个容器统一起来管理,使得部署和管理变得极为繁琐.针对以上问题,Docker Compose[30]实现了单主机多容器应用的管理,它可以根据配置文件自动构建、管理、编排一组容器,为容器应用的定义和运行带来极大的便捷.

以上研究很好地解决了单主机多容器的管理问题,然而应该如何高效地抽象和管理一个大规模的容器集群?Omega[31],Borg[32],Kubernets[44],Docker Swarm[45]以及Mesos[33]分别针对该问题进行研究,提出了高效的容器集群资源管理策略,实现了容器集群生命周期管理、动态负载均衡、应用配置管理、资源额度管理等功能,为容器集群的运行提供了高效性和稳定性.因为容器集群编排不属于操作系统虚拟化架构中容器管理的范畴,在此仅以业界目前最常用的Kubernets为例简介容器集群管理的现状.Kubernets提出pod的概念来作为容器的逻辑单元.所有的容器均在pod中运行,一个pod可以承载一个或者多个相关的容器,这些容器被共同部署和调度,由此简化对容器集群的管理.典型的Kubernets集群包含一个Master节点和多个Node节点.Mater是控制集群的中心,上面运行着多个进程,包括面向用户的API服务、负责维护集群状态的Controller Manager、负责调度任务的Scheduler等.Node是提供CPU、内存等资源的节点,每个Node节点上运行着维护Node状态并和Master通信的kubelet,以及实现集群网络服务的kube-proxy.综上,Kubernets通过对容器进行细致地组合,配合以上功能组件的使用,实现了简单高效的容器集群管理机制.

2.3　内核资源层

操作系统虚拟化内核资源层的主要作用是对共享的CPU、内存、网络等系统资源进行隔离,为上层容器的资源访问提供保障.具备良好的隔离性是促进容器技术进一步发展的基础.但是,容器隔离性是一个非常复杂的问题,从不同角度来看,隔离性具有不同的涵义和要求[34].下面分别从资源隔离和安全隔离2个方面介绍操作系统虚拟化隔离性研究现状.

1) 资源隔离

资源隔离主要是指统计和限制一个容器的资源消耗能力,从而为其他容器保证资源公平共享.在操作系统虚拟化环境中,最常用的资源隔离方法是使用 Cgroup[6]资源管理框架.它是 Linux 内核提供的一种可以限制、记录、隔离进程组所使用的物理资源(如 CPU、内存、I/O 等)的机制.Cgroup 为资源管理提供一个统一的框架,现有 CPU,memory,blkio 等子系统分别负责相应物理资源的管理,并且 Cgroup 也为未来开发新的子系统提供接口.Cgroup 为操作系统虚拟化提供了 3 项功能:①限制容器可以使用的资源数量,比如:memory 子系统可以为容器设定一个 memory 使用上限,一旦容器应用使用的内存达到限额再申请内存就会触发 OOM(out of memory);②记录容器使用的资源数量,比如:可以使用 cpuacct 子系统记录某个进程组使用的 CPU 时间;③进程组控制,比如:使用 freezer 子系统可以将容器挂起和恢复.基于以上功能,容器能够对物理资源进行精确的分配和调度.

在 Cgroup 出现之后,Laadan 等人[35]探索了操作系统虚拟化在实现过程中存在的资源竞争问题,着重分析了针对进程标识符(PID)、文件系统等逻辑资源的竞争,但并未提出有效的解决方案.Mcdaniel 等人[36]通过修改 Docker API 实现了容器对 Cgroup 中 blkio 子系统的访问,设计了一个集群级和节点级的两层控制方案,以完成容器集群中对容器 I/O 访问的简单隔离.也有一些开源社区尝试从容器引擎层面解决资源隔离问题,例如 LXCFS[46]试图利用内核的用户态文件系统模块在容器内实现虚拟 sys 和 proc 文件系统,以增强容器的资源视图隔离性.

综上所述,Cgroup 资源管理框架是目前最常用的操作系统虚拟化资源隔离机制,并且能够有效保证容器之间物理资源的公平共享.在 Cgroup 之后的研究工作着重对没有得到 Cgroup 支持的资源进行隔离需求分析或者简单隔离.

2) 安全隔离

安全隔离主要指限制容器对逻辑对象的访问,例如文件、内存地址空间、端口号、用户标志、进程标志等.例如,通过安全隔离,一个容器的文件命名不会与其他容器的文件冲突并且容器之间互相看不到文件或者进程的信息,因此它们只能修改自己的数据和代码,从而减少容器之间互相影响的可能性.

操作系统虚拟化环境下安全隔离最初始的实现是 chroot 命令[2],通过它可以实现文件系统隔离.但是在分布式环境下,容器必须要有独立的 IP、端口和路由等,自然就有了网络隔离.同时容器还需要一个独立的主机名来标识自己,进程通信隔离、权限隔离等也需要考虑到.Linux 内核提供的 namespace[8]完整支持了以上容器所需的安全隔离.在同一个 namespace 下的进程可以感知彼此的变化,但对外界的进程一无所知.这样就可以让容器中的进程感觉自己置身于一个独立的系统环境中,以此达到配置独立和安全隔离的目的.然而,在一些特定的应用环境中,有些 namespace 隔离是不需要的,甚至会带来额外的开销.SOCK[18]通过实验分析无服务器架构环境下 namespace 带来的性能开销,发现 network namespace 会为整个系统带来额外的延迟,mount namespace 会影响系统的可扩展性.同时,由于无服务器架构本身的特性,network namespace 不是必要的,mount namespace 也可以用 chroot 代替,以此减少安全隔离带来的性能开销.

还有一些研究是利用体系结构技术来辅助实现操作系统虚拟化安全隔离.Dune[37]提出了应用程序直接访问底层硬件的想法.Nested kernel[38]采用嵌套内核的方式来增强操作系统内核的安全性.okernel[39]在以上 2 个研究的基础上,通过嵌套内核并利用扩展页表(extended page table, EPT)技术,实现内存地址空间的隔离.SCONE[40]为增强文件系统隔离和系统权限隔离,通过减少可信计算基以及抽象带来的开销,有效利用 Intel SGX[47]技术保护容器,加强其隔离性.

由于容器安全隔离性较弱的根本原因是容器共用主机内核,因此增强容器安全隔离性的另一种思路是阻断容器内应用程序对主机内核的依赖.一种方法是让容器运行在传统虚拟机上,用虚拟机的良好安全隔离性来弥补容器安全隔离性的不足,现在主流的云计算平台皆采用这种方式提供容器服务[41].但是,这种方式会造成冗余繁复的虚拟化资源管理,不仅增大了虚拟化的资源开销,而且使得系统软件栈复杂低效.Kata Containers[48]针对以上问题,将容器运行在一个优化过的内核之上,基于 Intel VT 技术提供对网络、I/O、内存等硬件资源的安全隔离,从而将虚拟化的安全隔离优势和容器的快速启动特点结合起来.另一种方法是 gVisor[49]在主机内核之外实现了一个“内核进程”,提供了大部分 Linux 内核的系统调用;然后通过在用户空间内拦截容器内应用程序系统调用并转发至“内核进程”,以此阻断容器内的恶意代码对主机内核的访问,增强安全隔离.

综上所述,操作系统虚拟化的安全隔离主要依赖于 Linux 内核中的 namespace 隔离机制,但是还存在较多缺陷.随着体系结构技术的发展,一些前沿的研究工作开始探索利用体系结构特征来增强操作系统虚拟化的安全隔离.工业界也针对虚拟机内部嵌套容器存在的问题开展了系统优化研究.

3　问题与展望

目前,关于操作系统虚拟化的研究虽然已经取得了一定成果,但从实际应用来说,还存在很多问题需要研究,下面对其中的 3 个主要问题进行分析.

3.1　应用兼容性

原本基于物理机或传统虚拟机开发的应用,迁移到容器中运行时会存在兼容性问题.操作系统虚拟化技术由于其高效的性能得到了业界的青睐,越来越多的应用运行于容器中.然而,由于容器本质上只是一个轻量级的隔离环境并且与其他容器共享同一个操作系统内核,原本基于物理机或者传统虚拟机开发的应用会在配置时出现兼容性问题.出现应用兼容性问题的根本原因是容器内应用无法准确地获取资源视图信息.例如在 Linux 主机环境下,应用通过访问/proc 和/sys 虚拟文件系统即可获得资源视图信息;然而由于容器系统没有独立的 OS 内核,容器内应用获得的资源视图实际上是主机的总体资源状态信息,从而导致容器内应用做出错误决策并对应用性能造成严重影响.具体来说,以 Java 虚拟机(JVM)为例,JVM 会根据所在环境的内存量大小来决定其内存管理策略,当 JVM 运行在容器中时,由于没有准确的资源视图,JVM 经常会申请超过容器资源限额的过量内存,从而产生大量的 swap 操作,严重影响运行性能.所以,由容器内应用资源视图信息不准确引起的应用兼容性问题会在一定程度上制约操作系统虚拟化技术的发展.综上,探索增强应用兼容性的方法、扩大容器应用范围,对于操作系统虚拟化的发展具有重要意义.

3.2　镜像体积

随着操作系统虚拟化的广泛应用,不同的用户对容器镜像有不同的需求,基于容器镜像的分层特性,每种需求都会增加一个镜像层,因此出现了容器镜像数量越来越多、容器镜像体积越来越大的现象.当容器镜像不在本地时,容器镜像体积的增大会使得容器镜像的拉取时间或者构建时间变长.由于容器的部署时间大部分来自于容器镜像的拉取或构建,所以容器镜像体积的变大会增加容器的启动部署时间并且增加磁盘空间占用.这些额外开销在边缘计算等场景中会严重影响系统性能,甚至会抵消使用容器带来的好处,失去快速迭代开发和部署应用的能力.但是对于某一特定应用来说,并非所有的镜像数据都是有用的,容器每一层镜像中都存在一定的冗余数据.综上,轻量化容器镜像的研究具有可行性,并且对容器技术的进一步发展具有重要意义.

3.3　资源隔离

相比于传统虚拟化技术,现有操作系统虚拟化技术具有开销小、效率高等优点,但是由于多个容器共享同一个操作系统内核,导致容器环境的隔离性相对虚拟机而言较弱[42].随着服务器硬件性能的不断增强和各种应用逐步部署于操作系统虚拟化环境,不仅单一宿主机上的容器数量越来越多,而且面临的应用场景和需求也更加复杂多样,容器部署的高密度和应用的复杂性使得解决容器隔离性问题的需求日益迫切.尽管隔离性问题是容器领域的研究热点,获得了工业界和学术界的广泛关注,但是目前的研究成果非常有限,还存在以下诸多研究挑战与重要机遇.

1) 统一的资源隔离模型

操作系统虚拟化资源隔离问题涉及因素较多、复杂性高,目前缺乏较为完整统一的资源隔离模型.针对容器隔离性问题,业界主要采用的解决思路是针对应用特征进行容器资源隔离性的完善,目前学术界的研究工作较多基于这种思路开展.这种方式的问题在于:仅依据某类应用的具体需求对操作系统内核进行修补和完善,一方面会增加操作系统的复杂性,使得内核变得越来越笨重,难以实际推广使用;另一方面缺乏具有通用性的整体解决思路,难以实现运行环境的全面隔离,也无法满足复杂应用场景的需求.另外,在容器系统中,运行环境隔离需要考虑一系列的系统资源.除了 CPU、内存、I/O 等“硬资源”之外,容器运行环境的隔离还需要考虑 OS 内核中的一系列“软资源”,包括进程号(pid)、文件描述符(fd)、索引节点(inode)、套接字(socket)等.由于容器共享 OS 内核,如果不对这些内核软资源的使用进行隔离限制,可能会带来意想不到的负面效果.例如,如果某个容器大量重复执行创建文件的操作,消耗了主机上所有的 fd 或者 inode 资源,那么同一主机上的其他容器就无法再正常创建文件,这不仅影响应用正常运行,而且存在严重安全隐患.上述分析说明,容器资源隔离是一个涉及诸多因素、

诸多方面的复杂问题,目前业界还缺乏清晰完整的隔离模型.因此设计简洁、高效、统一的容器资源隔离模型存在较大的研究前景.

2) 软硬协同的资源隔离机制

计算机体系结构技术的发展为解决操作系统虚拟化资源隔离问题带来了新的挑战和机遇.随着计算机体系结构技术的发展,为了满足云计算环境的需求,近年来陆续出现了一系列可对硬件资源进行灵活管理配置的技术.例如英特尔的 RDT(resource director technology)技术[50]为管理配置主机的 CPU cache 和内存带宽提供了有效手段,Open-Channel SSD[51]为上层 OS 和应用开放了存储设备的管理控制细节等.这些技术的出现和发展,使得"软件定义"可以深入到数据中心基础设施的底层,推动了先进的资源感知型编排管理技术的不断发展,不仅提高了云计算环境的服务器资源利用率,保障了应用服务质量,而且深层次促进了行业变革.体系结构领域的上述发展趋势为解决容器资源隔离问题提供了很好的底层技术支持和研究契机,如何充分利用新型体系结构技术实现软硬件协同的容器资源隔离机制,是一个具有前沿性和新颖性的研究问题,具有较大研究空间.

4 总 结

操作系统虚拟化允许多个不同的应用在共享操作系统内核的环境中隔离运行,作为一种轻量级虚拟化技术,操作系统虚拟化在很多应用领域都具有巨大的潜力.本文介绍了操作系统虚拟化的技术架构层次,并通过与传统虚拟化技术对比,总结其在启动速度、资源利用率等方面的优势以及隔离性的不足;随后,分别从容器实例层、容器管理层和内核资源层介绍了操作系统虚拟化应用场景、性能分析、镜像管理、容器编排和隔离性的研究现状;最后,基于以上研究现状分析了操作系统虚拟化依然存在的问题和挑战,并展望了未来可能的研究方向.

参 考 文 献

[1] Linux Journal. Docker: Lightweight Linux containers for consistent development and deployment [EB/OL]. [2014-05-19]. https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment

[2] Linux Programmer's Manual. Linux Chroot [EB/OL]. (2018-04-30) [2018-09-14]. http://man7.org/linux/man-pages/man2/chroot.2.html

[3] Kamp P H, Watson R N M. Jails: Confining the omnipotent root [C] //Proc of the 2nd Int System Administration and Network Engineering Conf. Berkeley, CA: USENIX Association, 2000: 116-132

[4] Price D, Tucker A. Solaris zones: Operating system support for consolidating commercial workloads [C] //Proc of the 18th Large Installation System Administration Conf. Berkeley, CA: USENIX Association, 2004: 241-254

[5] SWsoft. OpenVZ: A container_based virtualization for Linux [EB/OL]. (2016-11-30) [2018-09-14]. https://wiki.openvz.org/Main_Page

[6] Red Hat Enterprise Linux. Introduction to control groups [EB/OL]. [2018-09-14]. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/ch01

[7] Linux Container. Introduction to LXC [EB/OL]. [2018-09-14]. https://linuxcontainers.org/lxc/introduction/

[8] Linux/UNIX System Programming Training. Linux Programmer's Manual. Linux namespace [EB/OL]. (2018-02-02) [2018-09-14]. http://man7.org/linux/man-pages/man7/namespaces.7.html

[9] Sheoran A, Sharma P, Fahmy S, et al. Contain-ed: An NFV micro-service system for containing e2e latency [J]. ACM SIGCOMM Computer Communication Review, 2017, 47(5): 54-60

[10] Hao Tingyi, Wu Heng, Wu Guoquan, et al. Elastic reource provisioning approach for container in micro-service architecture [J]. Journal of Computer Research and Development, 2017, 54(3): 597-608 (in Chinese)
(郝庭毅, 吴恒, 吴国全, 等. 面向微服务架构的容器级弹性资源供给方法[J]. 计算机研究与发展, 2017, 54(3): 597-608)

[11] Handigol N, Heller B, Jeyakumar V, et al. Reproducible network experiments using container-based emulation [C] //Proc of Int Conf on Emerging NETWORKING Experiments and Technologies. New York: ACM, 2012: 253-264

[12] Kang Hui, Le Michael, Tao Shu. Container and microservice driven design for cloud infrastructure devops [C] //Proc of 2016 IEEE Int Conf on Cloud Engineering. Piscataway, NJ: IEEE, 2016: 202-211

[13] He Sijin, Guo Li, Guo Yike, et al. Elastic application container: A lightweight approach for cloud resource provisioning [C] //Proc of the 26th Int Conf on Advanced Information Networking and Applications. Piscataway, NJ: IEEE, 2012: 15-22

[14] Pahl C, Helmer S, Miori L, et al. A container-based edge cloud PaaS architecture based on Raspberry Pi clusters [C] //Proc of IEEE Int Conf on Future Internet of Things and Cloud Workshops. Piscataway, NJ: IEEE, 2016: 117-124

[15] Gerlach W, Tang W, Keegan K, et al. Skyport: Container-based execution environment management for multi-cloud scientific workflows [C] //Proc of Int Workshop on Data-Intensive Computing in the Clouds. Piscataway, NJ: IEEE, 2014: 25-32

[16] Wu Song, Niu Chao, Rao Jia, et al. Container-based cloud platform for mobile computation offloading [C] //Proc of IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2017: 123-132

[17] Hu Jiahuan, Wu Song, Jin Hai, et al. Container-based customization approach for mobile environments on clouds [C] //Proc of the 13th Int Conf on Green, Pervasive and Cloud Computing. Berlin: Springer, 2018: 3-17

[18] Oakes E, Yang L, Zhou D, et al. SOCK: Rapid task provisioning with serverless-optimized containers [C] //Proc of the 2018 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2018: 57-69

[19] Bernstein D. Containers and cloud: From LXC to Docker to Kubernetes [J]. IEEE Cloud Computing, 2015, 1(3): 81-84

[20] Plauth M, Feinbube L, Polze A. A performance survey of lightweight virtualization techniques [C] //Proc of Service-Oriented and Cloud Computing: The 6th IFIP WG 2. 14 European Conf. Berlin: Springer, 2018: 34-48

[21] Xavier M G, Oliveira I C D, Rossi F D, et al. A performance isolation analysis of disk-intensive workloads on container-based clouds [C] //Proc of Euromicro Int Conf on Parallel, Distributed and Network-Based Processing. Piscataway, NJ: IEEE, 2015: 253-260

[22] Xavier M G, Neves M V, Rossi F D, et al. Performance evaluation of container-based virtualization for high performance computing environments [C] //Proc of the 21st Euromicro Int Conf on Parallel, Distributed, and Network-Based Processing. Piscataway, NJ: IEEE, 2013: 233-240

[23] Guedes E A C, Silva L E T, Maciel P R M. Performability analysis of I/O bound application on container-based server virtualization cluster [C] //Proc of Int Symp on Computers and Communications. Piscataway, NJ: IEEE, 2014: 1-7

[24] Felter W, Ferreira A, Rajamony R, et al. An updated performance comparison of virtual machines and Linux containers [C] //Proc of IEEE Int Symp on PERFORMANCE Analysis of Systems and Software. Piscataway, NJ: IEEE, 2007: 171-172

[25] Suo Kun, Zhao Yong, Chen Wei, et al. An analysis and empirical study of container networks [C] //Proc of IEEE Conf on Computer Communications (INFOCOM). Piscataway, NJ: IEEE, 2018: 189-197

[26] Ruan Bowen, Huang Hang, Wu Song, et al. A performance study of containers in cloud environment [C] //Proc of Asia-Pacific Services Computing Conf. Berlin: Springer, 2016: 343-356

[27] Harter T, Salmon B, Liu R, et al. Slacker: Fast distribution with lazy docker containers [C] //Proc of the 14th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2016: 181-195

[28] Zheng Chao, Rupprecht L, Tarasov V, et al. Wharf: Sharing Docker images in a distributed file system [C] //Proc of the ACM Symp on Cloud Computing. New York: ACM, 2018: 174-185

[29] Anwar A, Tech V, Mohamed M, et al. Improving docker registry design based on production workload analysis [C] //Proc of the 16th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2018: 265-278

[30] Freeman A. Essential Docker for ASP. NET Core MVC[M]. Berkeley, CA: Apress, 2017: 97-117

[31] Schwarzkopf M, Konwinski A, Abd-El-Malek M, et al. Omega: Flexible, scalable schedulers for large compute clusters [C] //Proc of the 8th European Conf on Computer Systems. New York: ACM, 2013: 351-364

[32] Verma A, Pedrosa L, Korupolu M, et al. Large-scale cluster management at Google with Borg [C] //Proc of the 10th European Conf on Computer Systems. New York: ACM, 2015: 18:1-18:17

[33] Hindman B, Konwinski A, Zaharia M, et al. Mesos: A platform for fine-grained resource sharing in the data center [C] //Proc of the 8th USENIX Conf on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2010: 429-483

[34] Soltesz S, Fiuczynski M E, Bavier A, et al. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors [J]. ACM SIGOPS Operating Systems Review, 2007, 41(3): 275-287

[35] Laadan O, Nieh J. Operating system virtualization: Practice and experience [C] //Proc of the 3rd Haifa Experimental Systems Conf (SYSTOR 2010). New York: ACM, 2010: 1-12

[36] Mcdaniel S, Herbein S, Taufer M. A two-tiered approach to I/O quality of service in Docker containers [J]. IEEE/ACM Transactions on Networking, 2015, 6(1): 42-55

[37] Belay A, Bittau A, Mashtizadeh A, et al. Dune: Safe user-level access to privileged CPU features [C] //Proc of the 10th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2012: 335-348

[38] Dautenhahn N, Kasampalis T, Dietz W, et al. Nested kernel: An operating system architecture for intra-kernel privilege separation [C] //Proc of the 20th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2015: 191-206

[39] Linuxkit. Projets of okernel [EB/OL]. (2018-08-30) [2018-09-14]. https://github.com/linuxkit/linuxkit/tree/master/projects/okernel

[40] Arnautov S, Trach B, Gregor F, et al. SCONE: Secure Linux containers with Intel SGX [C] //Proc of the 12th USENIX Symp on Operating System Design and Implementation. Berkeley, CA: USENIX Association, 2016: 689-704

[41] Wu Zhixue. Advances on virtualization technology of cloud computing [J]. Journal of Computer Applications, 2017, 37(4): 915-923 (in Chinese)

(武志学. 云计算虚拟化技术的发展与趋势[J]. 计算机应用, 2017, 37(4): 915-923)

[42] Matthews J N, Hu W, Hapuarachchi M, et al. Quantifying the performance isolation properties of virtualization systems [C] //Proc of the Workshop on Experimental Computer Science. New York: ACM, 2007: 6-16

[43] Apache. OpenLambda project [EB/OL]. [2018-09-14]. http://www.open-lambda.org/

[44] Google. Kubernets: Production-Grade container orchestration [EB/OL]. [2018-09-14]. https://kubernetes.io/

[45] Docker. Docker Swarm [EB/OL]. [2018-09-14]. https://docs.docker.com/swarm/

[46] Linux Container. What is LXCFS [EB/OL]. [2018-09-14]. https://linuxcontainers.org/lxcfs/introduction/

[47] Intel. Intel software guard extensions [EB/OL]. [2018-09-14]. https://software.intel.com/zh-cn/sgx

[48] Kata Container. Kata container: The speed of containers, the security of VMs [EB/OL]. [2018-09-14]. https://katacontainers.io/

[49] Google. Open-sourcing gVisor, a sandboxed container runtime [EB/OL]. (2018-05-02) [2018-09-14]. https://chinagdg.org/2018/05/open-sourcing-gvisor-a-sandboxed-container-runtime/

[50] Intel. Resource director technology: Unlock system performance in dynamic environments [EB/OL]. [2018-09-14]. https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html

[51] Wikipedia. Open-channel SSD [EB/OL]. [2018-09-14]. https://en.wikipedia.org/wiki/Open-channel_SSD

Wu Song, born in 1975. PhD, professor, PhD supervisor. Member of CCF. His main research interests include cloud computing, system virtualization, datacenter management, and in-memory computing.

Wang Kun, born in 1992. PhD candidate. Student member of CCF. His main research interests include cloud computing and system virtualization.

Jin Hai, born in 1966. PhD, professor, PhD supervisor. Fellow of CCF. His main research interests include computer architecture, virtualization technology, cloud computing, network storage, and network security.