基于容器的操作系统虚拟化：

一种高性能，可扩展的虚拟机监视器（Hypervisors）替代者

翻译：黎润

Mail: yijunzhu@live.com

Blog: blog.sina.com.cn/yijunzhu

摘要：

虚拟机监视器（即Hypervisors，下文不再翻译）伴随着Xen和VMware的流行而快速商业化。的确，Hypervisors解决方案（即全虚拟或者半虚拟）满足绝大部分的应用场景，但是在一些对于对于隔离和性能都敏感的业务领域，它却显得有点力不从心。例如高性能计算集群，网格计算，主机中心等等。本文提出一种更适合这种场景的解决方案，该方案集成了以前通用分时操作系统的资源容器和安全容器的理论精华。Sloaris10和linux项目里的VServer，Virtuozzo就是集大成者。本文详细描述了VServer的设计和实现理念，另外还和Xen类的虚拟机做了一些架构上的比对，最后论述了VServer为什么能够在隔离和效率之间找到合适的平衡点。

1 简介

操作系统在设计的时候就面临着隔离和共享的根本性矛盾，即要让每一个应用程序都感知到自己是完全占有硬件资源的，同时又要方便的在应用程序之间共享对象（例如管道，文件）。我们当前所使用的个人电脑操作系统是由老的分时系统改造而来，它在资源共享方面有先天优势。另外一方面，基于hypervisors的虚拟机解决方案首先解决的是数据隔离问题，带来的后果就是两台虚拟机之间只能通过网络共享数据。这就要求我们能够在两者之间找到一个权衡。

来看一个常识，个人电脑常常是一个用户运行多个应用程序，和隔离比起来，追求共享是它的天性。对比来看，hypervisor主要用来在一台机器上运行多个不相干的应用程序，这些不同的应用程序可能属于各个不同的独立组织。这样的话，这些应用程序之间就无序共享数据，我们也就较为容易得出结论，比起共享，hypervisors更擅长隔离。但是假如每一个虚拟机都运行同样的内核和操作系统，那么hypervisors所带来的隔离是凌驾于所有应用都跑在一个单一内核的效率降低基础之上的。

许多新型的应用（例如高性能计算机群，网格等等）得益于虚拟技术，虚拟化就是把用户和应用彼此隔离的一种技术。这些新型的应用很看重资源共享的高性性，这种高效性体现在虚拟机计算的裸性能或者大量运行虚拟机之间的扩展性。

本文描述的虚拟化技术能够在增强隔离性的基础上，仍然保证系统资源共享的高效性。这种技术集成了以前通用分时操作系统的资源容器和安全容器的理论精华。Sloaris10和linux项目里的VServer，Virtuozzo就是集大成者。

本文有两个主要贡献：

第一：本文是第一次透彻的面对科研工作者描述VServer的技术。我们选择VServer作为基于容器的虚拟机代表出于如下的几点原因。1)它是开源的，2)它在生产环境中使用，3)我们有实际操作运营VServer的经验和数据。

第二：我们对比了VServer和最新进行了重大修正的Xen。就性能而言，两者对于CPU的调度使用基本是类似的，但是一些IO密集的业务，VServer在系统资源使用上更为高效，故总体而言，VServer的性能更佳。那么扩展性呢，在一些可能存在业务透支（Overbook）的应用场景下，VServer的表现远远超过Xen。

下一章主要描述基于容器的虚拟机研发动机和作用。第三章从细节上描写基于容器的虚拟

技术，同时提及了VServer的设计和实现机制。第四章则重点关注Xen和VServer对比数据的基线。第五章是实验方法论，第六章是结论。

2 动机

虚拟机技术不同的人有不同的见解。为了简单起见，我们决定统一起见，定义为运行于提供虚拟功能的系统之上的隔离执行环境。虚拟架构从全硬件到全软件都有，全硬件的例如Intel的VT技术；全软件但是提供硬件抽象功能的虚拟机，例如Xen，VMWare的ESX；系统调用层的虚拟机，例如Solaris和VServer；模拟器，例如QEMU；高级语言的虚拟机，例如JAVA；或者应用层程序的虚拟机，例如Apache的虚拟主机。那么多类型里面，我们集中比较hypervisor和基于容器的虚拟机。

本章的剩余部分首先介绍我们重点提及虚拟机的使用场景，对应的比对虚拟化的不同方法。接着我们创建一个基于容器的虚拟技术的应用场景。

2.1 使用场景

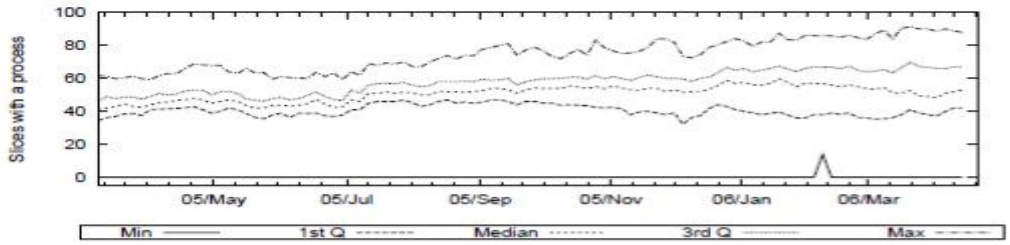
对于虚拟机的使用，我们有很多创新的主意。例如加固桌面电脑的工作安全环境；实时检测病毒攻击；找出电脑被入侵的原因并且调试系统错误。现在虚拟机更是大量的被程序员用于开发和测试，被IT数据中心用于整合业务，被传统的主机供应商用来卖虚拟主机。另外一些新型业务也在积极的考虑，评估使用虚拟主机来进行高性能计算，网格计算等。本文主要关注这些新型应用，性能是他们考虑的第一要素。

机器群，理想化的实现是基于网格的，但是在实际生产环境里面，往往都是通过高性能集群来实现。机器群通过批量调度的方案来支持多用户作业。其实机器群不需要运行太多的虚拟机（经常是一台物理机器运行一个虚拟机），但是他们对于系统总体的性能是相当的敏感。同时根据经验，机器群的配置错误也大部分是由于使用和内核不兼容的虚拟操作系统跑应用软件引入的。因此，让用户能够在虚拟机里面使用他们自己的版本库或者分发版本可以解决这个问题。

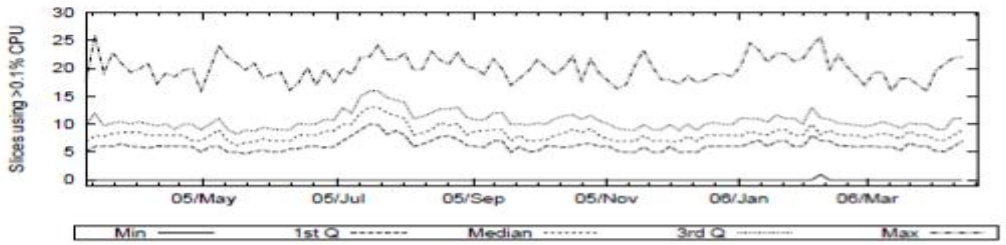
而主机托管则不一样，他们一般是在不同的虚拟机里面跑同样的内核，同样的软件。在主要关注盈利的应用场景下，主机托管商努力的从规模经济以及降低虚拟机的迁移成本里面获利。所以主机托管商总是想在不降低服务质量的前提下，最大化现有硬件的运行能力。不过不幸的是主机托管商自己也不清楚他们在一个硬件平台上可以跑多少个虚拟机。

幸运的是，CoMon发布了基于PlanetNet的虚拟机性能监控程序。（接下来就是介绍PlaneNet的东东，不翻译了。）

CoMon把虚拟机分成活跃的和运行的。活跃的主要指虚拟机里面有进程。而运行的主要指在最近的5分钟内，至少1%的CPU时间被程序消耗。看两个例子。



(a) Active slices, by quartile



(b) Live slices, by quartile

上图我们可以看到5条线，25%的阶段落在第一条和第二条之间，等等。我们也注意到，经常在5分钟的间隔内，有10-15个运行主机和60个活跃主机。我们要注意到，在这个网络里，都是普通的计算机，如果你也遇到我们一样的负载，请考虑虚拟机吧。

2.2 基于容器的虚拟机应用场景

适合基于容器的虚拟机一个重要的条件就是我们要在隔离和性能之间做平衡，第四节和第五节主要定量的分析基于容器的虚拟机比Xen在性能上更胜一筹。所以我们时刻需要问自己的是，隔离和性能哪一个更重要。

虚拟机的性能可以通过吞吐量，延迟等等衡量，而扩张性则可以借助并发的虚拟机数量来做一个评判，但是隔离性却难以界定一个数量标准。系统在具备了错误隔离，资源隔离以及安全隔离的特点后，我们就能够大致说这个系统是完全隔离的。下面也还会提及，基于容器的虚拟机和基于hypervisor的虚拟机两者是存在重叠的。

错误隔离主要指一个虚拟机出问题了之后，他能够控制对其他虚拟机的数据，状态以及其他操作的影响。为了达到这个目的，两台虚拟机之间必须完全直接的数据和代码共享。在基于容器和基于hypervisor的虚拟技术里面，虚拟机之间通过地址空间就能够达到错误隔离，不过他们在底层的虚拟层存在代码和数据的共享，如果在这个模块里面出bug，所有的系统都会收到影响。

历史经验证明，只有不足8w行代码的hypervisor层能够提供足够的可靠性。但是我们还需要关注一下虚拟机的主机操作系统（Host OS），它往往是一个完整的系统，虚拟机（Guest OS）对于设备的访问和认证，都需要借助主机操作系统完成，而主机操作系统宕机，所有虚拟机都挂了，所以有人建议把设备的驱动分割成独立的驱动域（IDD）。

Linux的体积巨大，很大程度上是设备驱动，文件系统以及网络协议栈造成的，它的内核核心部分只有不足14w的代码行。为了提高错误处理的弹性和灵活性，对于设备驱动可以采用IDD以及，或者在Linux内核里面采用Nook技术。不过不巧的是，目前还没有直接的数据来比较Xen+IDD和Linux+Nooks的性能。

那我们就假象一下设备驱动，文件系统以及网络协议栈都是极其安全的，那么错误恢复的难易就取决于暴漏给虚拟机的接口数目了。虚拟机之间看到的接口越多，他们被感染的可能性就越大。基于hypervisors的系统只提供了少数的时间和驱动接口，而基于容器的操作系统则提供了一些列的ABI接口。那么我们就能够推导出一个结论，基于hypervisors的虚拟机稍微安全一些。

资源隔离主要是指对一台虚拟机能够进行资源使用的审计以及执行资源的使用。从而确保其他虚拟机对于资源的公平使用。虚拟机之间未预料的交互我们常常称呼为干扰。提供资源隔离，从总体来说，我们主要就是能够对物理资源进行精度的分配和调度，例如CPU周期，内河大小，带宽，硬盘空间等等。实际上我们虽然实行了资源隔离，但是总有一些资源是共享了，他们之间也会互相干扰，例如文件描述符，端口，PID，共享缓存。资源共享的一个极端是任何资源都定量化，例如一个虚拟机可以独立的使用100Mcps，1.5Mbps的带宽；另外一个极端则是创建一个全局的共享资源池，所有人共享。当然在使用中我们可以选择两者的混合体作为一个较为完美的解决方案。

安全隔离主要指限制虚拟机对逻辑对象的访问，例如文件，虚拟内存地址，端口数目，用户的标志，进程ID等等。通过这些控制，安全隔离可以做到（1）配置独立，因此一个虚拟机的全局命名文件不会和其他机器的文件冲突。（2）安全，通过安全隔离，一台虚拟机都不能修改另外一台虚拟机的数据和代码，这样就减少了虚拟机之间互相影响的可能性。一个拥有完全安全的虚拟系统，虚拟机之间是互相卡不到文件或者进程的信息，这样他们只能修改自己的文件对象。但是一个只具备部分安全隔离的虚拟系统，他们就可能共享一个命名空间，例如全局的文件系统，通过这种方式，一台虚拟机就能够修改另外一台虚拟机的对象。后面我们就会看到，基于容器的虚拟机一般是后者，即访问控制应用在全局对象上，而不是维护每一个虚拟机的自知域和命名空间，选择这种方式，主要也是处于性能上的考虑。在这种半隔离的环境下，信息泄漏是可能存在的。例如一个非法的用户可能会潜在的获取已经在使用的端口，名称，进程等等信息。但是总而言之，两者技术（基于容器和基于hypervisors）都能够在虚拟机之间隐藏逻辑对象，做到配置独立和系统安全。

讨论: 人们往往因为强健的隔离技术以及其他增值属性而采用虚拟化技术。下表提供了流行的增值业务，例如多内核，每个虚拟机都有管理权限，定时监控，挂起以及在物理机器之间迁移虚拟机等等。

Features	Hypervisor	Containers
Multiple Kernels	✓	✗
Administrative power (root)	✓	✓
Checkpoint & Resume	✓	✓ [15, 22, 17]
Live Migration	✓	✓ [22, 17]
Live System Update	✗	✓ [17]

Table 1: Feature comparison of hypervisor- and COS-based systems

因为基于容器的虚拟机是跑在一个单一的内核镜像上，故他们不具备hypervisors多内核支持特性。也因为同样的原因，基于容器的虚拟机不具备很多人们渴求的特性，例如内核模块的加载等等底层操作。但是其余的操作基本都能够得到有力的支撑，我们在上面的表格里面已经看到。我们不能只看到不足，基于容器的虚拟机带来了一个全新的技术，那就是虚拟机迁移，通过这个技术，管理员能够做到在线升级。例如更换内核，性能增强或者新特性释放都不需要重启虚拟机。

表二（下表）给出了虚拟技术的不同纬度衡量标准，x轴用来描绘虚拟技术的隔离强度，y轴是交互的能力。

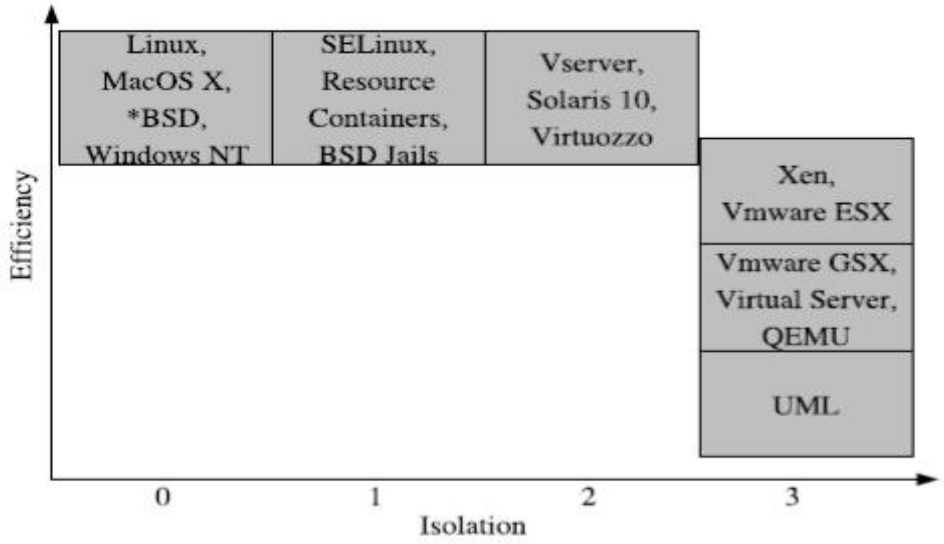


Figure 2: **Summary of existing hypervisor- and COS-based technology**

通过上图我们也能够看出，目前为止，没有一个完美的技术能够解决隔离性和效率的结合。我们可以简单的说一下，在效率要求比较严格的领域里，VServer更为适合一些。相应的，更看重隔离的场景，他们应该选择基于hypervisor的技术。最后要指出的是，两种技术并不是互斥的，用户完全可以在基于hypervisor虚拟机里面跑基于容器的虚拟机。

3 基于容器的操作系统

本节我们主要总体上看一下基于容器的操作系统是如何实现隔离的，以及VServer是通过什么机制来实现虚拟机的。

3.1 总览

基于容器的虚拟机提供了一个共享的，虚拟化的OS镜像，包括独一的根文件系统，一系列可执行文件和库以及其他创建虚拟机时候添加的东西。每一个虚拟机都可以像正常的操作系统一样关闭，挂起，并且启动只要数秒钟。对于应用和用户来讲，每一个虚拟机看起来就像一个独立的主机。正如下图所示，这儿有三组虚拟机，每一个主机平台都有共享的OS镜像和一个私有的主机虚拟机构成。

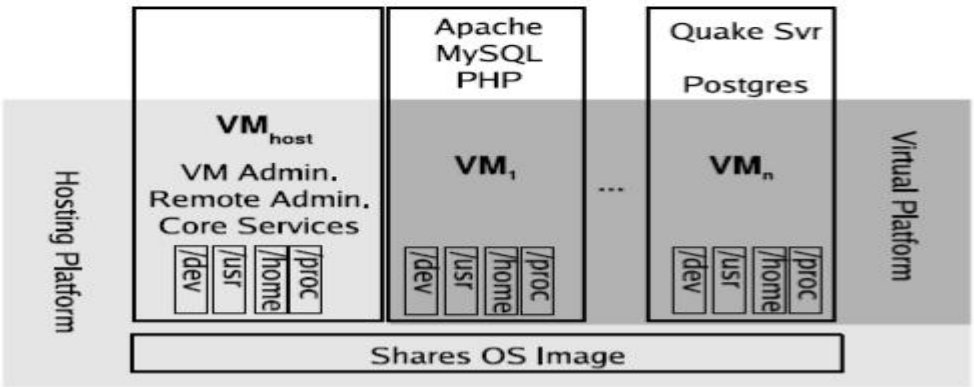


Figure 3: **COS Overview**

主机虚拟机（Host VM）主要用来管理其他虚拟机。而虚拟平台我们看到只有guest VM。跑在客户虚拟机里面的应用程序以为自己就是在一个正常的操作系统里面运行。从这边上看来，基于基于容器的和基于hypervisor的虚拟机存在一些差异，但是他们实质上的差异在于实现隔离的技术。

下图（图四）分别就他们的安全和资源隔离做了分类对比，基于容器的虚拟机主要通过隔离操作系统内核的对象完成安全性的隔离，例如（PID，UID，系统共享内存，IPC，ptys等等）。那么具体来讲应用了下列的一些技术(1)隔离命名空间(2)访问权限控制，以前的全局对象（句柄，用户ID等等）是在完全不同的空间里，不同虚拟机之间是完全看不到的，因此他们就不能访问到命名空间之外的对象。但是在上下文虚拟的应用场景下，全局的对象标记变成了每个虚拟机本地化的了，换句话讲就是全局的对象标记仅仅是在每个虚拟机内部全局。另外要提及的就是过滤器，它要在内核对象运行的时候，实时检查虚拟机是否有权限访问内核对象。基于hypervisor的虚拟机也是通过上下文虚拟化以及过滤器来实现隔离，不过它们往往是通过指令来完成的。这些指令主要应用在硬件抽象层，例如虚拟地址空间，PCI总线空间，设备以及特权指令。基于容器和基于hypervisor的虚拟机对于资源隔离所使用的方法是相近的。两者都需要复用物理资源，例如CPU周期，IO带宽，内存和存储空间。最新的Xen花了很大力气优化CPU的复用（其实就是guest VM可以直接操作CPU）。不过其他的物理资源，还是需要通过host VM来访问。让我们感兴趣的是，Xen的host VM是基于Linux，这样guest VM的资源控制器和Linux VServer就是一致的了。两者的区别就在于他们如何在虚拟机和资源之间做的映射关系。

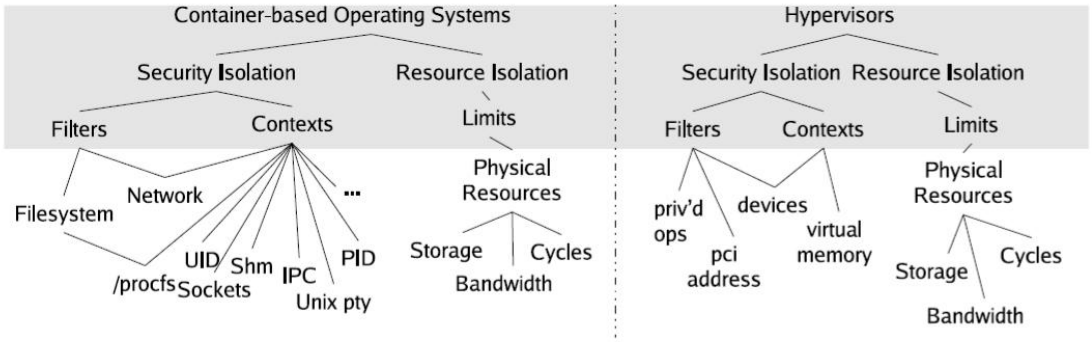


Figure 4: **Isolation Taxonomy of COS and Hypervisor-based Systems**

作为一个参考，基于hypervisor的虚拟机，他们在i32架构下，hypervisor大概有8w行代码，另外Linux不同分支的半虚拟化需要额外的1.5w行代码。还是看看VServer，它只需要不到9千行代码打到内核里面，也因为这个架构无关的特性，它仅仅需要微小的修正就很容易就能够运行在8种不同的架构上。VServer新建了50多个新文件另外修改了300多个老文件，真是不容易啊。

3.2 VServer 资源隔离

本节主要来说说VServer是如何做到资源隔离的。总体来看，它就是综合利用了Linux已经存在的资源管理和记账功能。不管物理的还是逻辑的资源，VServer都是简单的对VM能够消费的量做一个限制。

3.2.1 CPU调度：公平共享和资源预留

VServer实现的CPU资源隔离，在Linux标准的O(1)调度基础上增加了TBF（令牌桶过滤器）。每一个虚拟机都以一个固定的速度累加令牌，同时每一个tick，有运行进程的虚拟机都会消费令牌。当一个虚拟机把他的令牌消费完毕后，就会被移出运行队列，直到它又有了空闲的令牌。最初VServer的令牌桶用来设定CPU调度的上边界，但是我们也可以用这种方式来扩张资源隔离。后来我们通过修改令牌桶来提供CPU调度的公平共享和资源预留。令牌桶的发放速度取决于虚拟机是预留还是共享的方式运行，例如预留10%CPU的虚拟机每秒钟分发100个令牌（因为一个令

牌能够让一个进程运行1毫秒）。而共享CPU的虚拟机（有运行进程）在任何时候都会被调度，除非预留CPU的虚拟机被调度了。最后，CPU的运行总时间受到虚拟机类型的影响，预留CPU的虚拟机肯定能够得到预留值，而共享CPU则分享剩下的所有CPU时间。这样计算一下，一个虚拟机能够获得的CPU时间就是自己预留的（例如10%）加上共享的剩下CPU时间（例如1/10的剩余CPU时间）。

3.2.2 I/O QoS: 公平调度和资源预留

层次化的令牌桶（HTB）缓存了Linux的TC请求，通过这种方式来完成在虚拟机之间的带宽的预留和共享。每一个虚拟机都需要指定预留的带宽和共享的带宽。预留带宽指定本机发送出去的总带宽，而共享数值则用于超出预留值之后，如何分配共享资源。VServer发送出去的报文会被打上Tag，这个Tag会和每一个VServer的令牌桶关联上，层次化的令牌桶能够保证每一个VServer按照预留的资源发送报文，超出部分则是公平调度资源。所以我们创建一个虚拟机的时候可以仅仅制定预留或者共享的数值，也可以赋予两者的混合体。

硬盘的IO是通过Linux的标准CFQ（完全公平队列）进行调度，CGQ能够最大限度的保证每一个虚拟机共享块设备的IO带宽。

3.2.3 Storage Limits

VServer对于每一个虚拟机都提供了内存和存储空间的限制接口。硬盘空间的限制可以通过最大打开的磁盘block和inode节点数目来完成。而内存空间的限制则可以通过如下的一些途径a)控制RSS（resident set size）大小b)匿名内存页面的多少c)特定内存页面的数目，例如在虚拟机里面通过mlock和mlockall映射的页面。当然用户也可以声明虚拟机的共享内存的多少。请注意固定RSS的数目上限一般是不合适的，管理员希望能够在虚拟机里面透支页面数目。这样的话，其中一个可选方案就是让虚拟机完全占用内存，但是开启一个监控进程用于透支情况的恢复，例如可以死掉利用物理内存最多的进程。在我们的测试环境里面，一台机器上面跑了90个活跃的虚拟机，假定每一个虚拟机占用10M的内存，那么总体也要花销1GB的内存。所以我们就没有给每一个虚拟机分配独立的内存，而是全部共享，但是开启了一个监控进程，在交换空间快满的时候，死掉占用物理内存最多的应用作为惩罚。

3.3 VServer安全隔离

VServer在内核里做了大量的修改来加固安全隔离。

3.3.1 进程过滤

VServer在所有的虚拟机里面重用了全局PID命名空间，相对比而言，OpenVZ则是使用了每个虚拟机独立的全局命名空间。显而易见，后者更容易实现安全检查，恢复，迁移。以后VServer也会慢慢的往这个方向改，但是处于精确和完整性，我们会着重谈谈现在的老模式。

VServer通过过滤器来把进程隐藏在每个虚拟机的范围内，同时禁止不同虚拟机之间的异常交互。这样就要求我们要更改一些内核结构来达到如下目的：a)每个进程知道自己所属的虚拟机b)不同虚拟机使用的进程号存在差异。

为了规避一些用户空间程序（例如pstree）的假定（init进程都是存在而且PID为1），VServer提供了一个映射，会把一个进程映射为PID为1的INIT进程。

当运行有VServer的系统启动的时候，所有的进程都属于一个默认的host VM。为了简化管理，这个host VM和其他的guest VM没有任何的不同，但是在host VM里面能够看到全局的进程状况和操作所有的进程。相当于VServer定义了一个观察者，他能够立刻看到所有的进程而已。

这种方法的副作用就是我们在同一主机的虚拟机上面迁移进程的时候，仅仅需要更改了一些映射关系和统计值就可以了。

PDF-XCHANGE Click to buy NOW! www.docu-track.com

3.3.2 网络隔离

目前为止，VServer还没有和其他基于容器的虚拟机一样做到网络层的完全虚拟化，它在所有虚拟机之间共享网络层的资源，例如路由表，IP过滤表等等。但是虚拟机里面的应用程序只能绑定本机的IP地址和host VM的动态地址。这样的缺点就是每一个虚拟机不能修改路由表或者IP过滤表。但是这也是精心思考的结果，通过这种方式我们能够达到最佳的Linux网络性能。

VServer网络隔离有很多的细节值得斟酌，例如一个虚拟机绑定IPADDR_ANY或者本地主机地址（Loopback），这种情况要仔细处理，防止别的虚拟机的报文也会被本机收到。解决的方式是对于每个收到的报文都打上标签，在协议栈分发的时候，通过过滤器保证只有正确的虚拟机才能收到特定标签的报文。我们后面会看到，这种处理方式带来的额外开销和Linux的原生处理比起来可以忽略不计，总体的吞吐量相当喜人。

3.3.3 Chroot隔离

Chroot调用的一个主要问题是他的信息是不稳定，每次都可能存在震荡。一个简单的从chroot环境里退出的方法如下：

打开或者创建一个文件，保留文件句柄，接着用相同或者低级别的等级chroot到子目录，这样root就从文件系统中被“删除了“。

接着再用fchdir切换到上次打开的文件句柄，这样就从脱离了新的root，当然也没有了老的root（上次操作已经删除了）。

VServer用了一个特别的文件属性，称之为Chroot隔离，这样在每一个虚拟机里面，每一级的父目录都会阻止未认证的修改和擅自脱离chroot限制。

3.3.4 Linux能力的上限

因为目前的Linux能力没有实现部分POSIX相关的文件系统部分，这样setuid和setgid对于处于一个context里面的进程来说都有了安全上限。为了继续沿用这种安全能力，每个虚拟机的进程也有了和现存的能力集一样的控制。

3.4 文件系统的一致性

VServer的一个中心目的就是尽可能的减少系统总体开销，VServer用一种简单的办法来实现磁盘空间的节约，方法也很简单，对于一些几乎不变化的文件（例如库文件，二进制文件等等。），建立一个链接文件到虚拟机，这样虚拟机就能够使用这些文件了，从而节约了磁盘空间，内存使用等等。

但是这样做的坏处是，一个虚拟机有意或者无意的删除了这些共享的文件，其他的虚拟机也就会受到影响。VServer采用的这种技术也叫做写时拷贝。当一个虚拟机尝试修改这个文件的时候，虚拟机会拷贝一份私有的文件给他。

基于容器的虚拟机这种共享方法，我们称之为一致性，但是需要特别说明的是我们提出这种策略不是为了管理上的方便，而是为了减少资源的消耗。例如一个典型的Linux Server会消耗500M内存空间，但是10个这样的Server也只需要700M空间，并且需要更少的内存用户cache，这绝对是个不错的选择。

（第四章和第五章主要就是通过比较结果说明VServer性能和隔离性做的如何如何好，不翻译。）

4 System Efficiency

4.1 Micro-Benchmarks

4.2 System Benchmarks

4.3 Performance at Scale

4.3.1 OSDB

PDF-XCHANGE Click to buy NOW! www.docu-track.com

5 Resource Isolation

5.1 Single-dimensional Isolation

5.2 Multi-dimensional Isolation and Resource Guarantees

6 结论

虚拟化技术在大量的场合都给我们带来了好处。他提供了多种不同的技术来增加系统的安全性，例如隔离，配置独立，软件兼容，更好的系统利用率，可审计和预测的性能等等。关于隔离和性能，我们希望通过在建的项目能够共同提高基于容器和基于hypervisor虚拟机的规格。

同时，针对特定的应用，需要在隔离和性能之间做出权衡。实验表明基于容器的虚拟机理想情况下能够提供2倍的基于hypervisor的虚拟机性能，现在市面上有大量的基于容器和基于hypervisor的虚拟机存在，选择哪一个虚拟机完全是在特定的应用情境下更看重哪个特性。不管怎么样，我们都希望基于容器类的虚拟机能够成为基于hypervisor虚拟机（例如Xen）的强有力竞争者。