基于容器虚拟化技术研究

汪 恺,张功萱,周秀敏
(南京理工大学 计算机科学与工程学院,江苏 南京 210094)

摘 要:虚拟化是构建云基础架构不可或缺的关键技术之一,它保证了云计算平台的高性能与高可用性,使云计算平台在实际应用中资源最大化、利润最大化。Linux 容器 LXC(Linux containers)是一种内核虚拟化技术,可以提供轻量级的虚拟化,以便隔离进程和资源,而且不需要提供指令解释机制以及全虚拟化的其他复杂性。LXC 的实现需要 Linux 内核的支持,理解 Cgroups 和 Namespace 是分析 LXC 的关键。文中介绍了这两种内核机制,从原理上解释了如何隔离进程的资源,并分析了开源项目 LXC 容器启动的过程,然后通过实验来验证 LXC 在资源隔离和分配中所起的作用。随着 LXC 的更加完善,不久的将来轻量级虚拟化一定会推动 PaaS 的发展,被更广泛地运用在部署中。目前 openstack 已经支持基于 LXC 的 docker 部署,容器虚拟化技术也将会在云计算领域发挥越来越大的作用。

关键词:虚拟化;容器;资源控制;隔离;云计算
中图分类号:TP393 **文献标识码**:A **文章编号**:1673-629X(2015)08-0138-04
doi:10.3969/j.issn.1673-629X.2015.08.029

Research on Virtualization Technology Based on Container

WANG Kai,ZHANG Gong-xuan,ZHOU Xiu-min
(School of Computer Science and Engineering,Nanjing University of Science and Technology,Nanjing 210094,China)

Abstract:Virtualization is one of the key technology to build cloud infrastructure,which ensures high performance and availability of cloud computing platform,in practical applications,it makes cloud computing platform to maximize resources and profits. Linux container virtualization technology can provide lightweight virtualization to isolate processes and resources. Unlike other full virtualization,it is not required to provide instruction interpretation mechanisms. LXC implementation needs the support of the Linux kernel,understanding Cgroups and Namespace is the key to analyze the LXC. Two mechanism based on kernel are introduced in this paper,which explains how to separate process from the resource,and analyzes the open source project LXC and starting process of container,and then through the experiment to verify the LXC's role in resource isolation and distribution. With the improvement of LXC,the near future lightweight virtualization will promote the development of PaaS,which will be more widely used in the deployment. Currently,openstack supports docker which is based LXC,Linux container will play an increasingly large role in cloud computing.

Key words:virtualization;container;resources control;isolation;cloud computing

0 引 言

虚拟化是构建云基础架构不可或缺的关键技术之一。云计算的云端系统,其实质就是一个大型的分布式系统。虚拟化通过在一个物理平台虚拟出更多的虚拟平台,而其中的每一个虚拟平台则可以作为独立的终端加入云端的分布式系统。比起直接使用物理平台,虚拟化在资源的有效利用、动态调配和高可靠性方面有着巨大的优势。目前使用比较多的如 XEN,KVM 以及 VMware 都是一种平台虚拟化技术[1]。平台虚拟化是一种对计算机或操作系统的虚拟。其对用户隐藏了真实的计算机硬件,表现出另一个抽象计算平台[2]。虚拟机模拟一个足够强大的硬件使客户机操作系统独立运行。而基于容器的操作系统级虚拟化技术的关键思想在于操作系统之上的虚拟层按照每个虚拟机的要求为其生成一个运行在物理机器之上的操作系统副本,从而为每个虚拟机产生一个完好的操作系统,并且实现虚拟机及其物理机器的隔离[3]。

文中主要介绍一种基于容器的轻量级虚拟化技术

收稿日期:2014-09-26 **修回日期**:2014-12-30 **网络出版时间**:2015-07-21
基金项目:国家自然科学基金重点基金(612724420)
作者简介:汪 恺(1989-),男,硕士研究生,研究方向为大数据与云计算;张功萱,博士生导师,CCF 高级会员,研究方向为 Web 服务、可信计算等。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20150721.1448.056.html

LXC,分析 LXC 所依靠的两个机制,Cgroups 和 Namespace,并对其资源分配和隔离的性能进行测试,在此基础上介绍了基于 LXC 实现的开源项目 docker,总结并展望 LXC 的未来。

1　LXC 容器虚拟化以及实现机制

LXC 是 Linux Containers 的简称,是一种基于容器的操作系统层级的虚拟化技术。对于像 KVM、XEN 等平台虚拟化技术来说,LXC 采用完全不同的方法[4]。平台虚拟化是在仿真的硬件上引导单独的虚拟系统并通过半虚拟化技术等相关机制降低开销。LXC 不是在全隔离的基础改进效率,而是通过简易的机制实现隔离。从而实现的系统虚拟化机制可以像 chroot 一样扩展和移植,能够在同一个服务器同时支持数千台的仿真系统[4]。如图 1 所示,LXC 不需要 XEN 和 KVM 所必须的虚拟机监控层就可实现资源的隔离,共享库之上的应用程序 A 和 B 属于同一个容器,而 C 和 D 则属于另一个容器。不同的容器之间互不干扰,同一个容器内的进程可以正常通信。

图 1　容器虚拟化原理图

LXC 可以在操作系统层上为进程提供虚拟的执行环境,一个虚拟的执行环境就是一个容器[5]。可以为容器绑定特定的 CPU 和 memory 节点,分配特定比例的 CPU 时间、IO 时间,限制可以使用的内存大小,提供 device 访问控制,提供独立的 Namespace。

容器有效地将由单个操作系统管理的资源划分到孤立的组中,以更好地在孤立的组之间平衡有冲突的资源使用需求。与其他虚拟化相比,这样既不需要指令级模拟,也不需要即时编译[6]。容器可以在核心 CPU 本地运行指令,而不需要任何专门的解释机制。此外,也避免了准虚拟化和系统调用替换中的复杂性。通过提供一种创建和进入容器的方式,操作系统让应用程序就像在独立的机器上运行一样,但又能共享很多底层的资源。

1.1　Cgroups 机制

Cgroups 系统设计的初衷是为了实现管理以及控制系统资源,是由 Linux 内核提供的按照层级结构划分、聚集进程的一套机制,这些分组受内核的参数影响,通过改变这些参数影响分组中进程资源的分配[7]。Cgroups 加入了 Linux 内核之前已经存在的 cpuset、内存、设备各个子系统的功能,分配资源时按进程为最小单位。Cgroups 通过一个结构体 css_set 来存储指向进程所关联分组的资源信息,如果一个进程被加入分组之后,会关联一个 css_set 结构,在结构体之中有一个指针,存储着指向资源控制子系统的地址,这些子系统包括 CPU、Memory 等控制器[8]。

Cgroups 是一个具有层级结构的控制族群,每一个子节点继承了其父节点的控制属性。上面提到的每一个子系统必须附加到相应的层级上才能起作用,一旦子系统被附加在了一个层级上之后,该层级上的所有控制族群都受到这个子系统的控制[9]。一个进程可以加到某个控制族群,也可以从一个进程组迁移到另一个控制族群。一个进程组的进程可以使用以控制集群为单位分配的资源,同时也受到同一个控制族群的限制。控制族群树如图 2 所示。

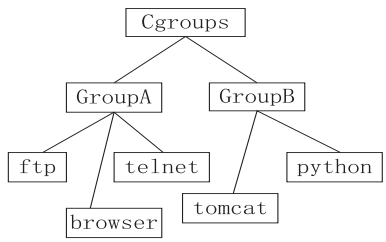

图 2　控制族群图

其中,GroupA 和 GroupB 是两个不同的进程组,每个进程组会被分配不同的控制属性,进程组内的所有进程享有上一层级所有的控制属性。

1.2　Namespace 机制

Namespace 机制是实现轻量级操作系统级虚拟化的基础,它为各个进程分组提供不同的命名空间[10]。在之前的 Linux 操作系统中,PID 以及网络设备之类的资源是在全局的范围内共享的,在内核中会维持一些关于全局资源的列表。在引入命名空间之前,每一个进程只有唯一一个 PID,通过进程的 PID 可以得到进程的相关信息。如今,有了命名空间,进程可以在一个新的 PID 命名空间中拥有一个新的 PID,这个 PID 只在相应的命名空间中有效。Namespace 机制将系统中的资源分成若干部分,在同一个命名空间的进程共享该命名空间内的资源,可以把一个命名空间看成一个容器,每个容器里面的进程互不干扰[2]。

每一个进程其所包含的命名空间都被抽象成一个 nsproxy 指针,共享同一个命名空间的进程指向同一个指针,指针的结构通过引用计数来确定使用者数目。当一个进程其所处的用户空间发生变化的时候就发生分裂。通过复制一份老的命名空间数据结构,然后做一些简单的修改,接着赋值给相应的进程。命名空间本身只是一个框架,需要其他实行虚拟化的子系统实现自己的命名空间。这些子系统的对象就不再是全局维护的一份结构了,而是和进程的用户空间数目一致,

每一个命名空间都会有对象的一个具体实例。目前 Linux 系统实现的命名空间子系统主要有 UTS、IPC、MNT、PID 以及 NET 网络子模块。下面介绍该机制如何通过这些子模块达到进程隔离的效果。

(1)PID 空间为进程提供了一个独立的 PID 环境。fork,clone 等系统调用产生的进程都有独立的进程 ID。新创建的第一个进程在该命名空间内的 PID 从 1 开始,就像独立系统里的 init 进程一样。命名空间内的孤儿进程都将以该进程为父进程,当该进程被结束时,所有的进程都会被结束。PID 空间是层次性的,新创建的空间将会是创建该空间进程所属空间的子空间。子空间中的进程对于父空间是可见的,一个进程将拥有不止一个 PID,而是在所在的空间以及所有直系祖先空间中都将有一个 PID。系统启动时,内核将创建一个默认的 PID 空间,该空间是所有以后创建的空间的祖先,系统所有的进程在该空间都是可见的。

(2)IPC 空间由一组 System V IPC objects 标识符构成,这标识符由 IPC 相关的系统调用创建。在一个 IPC 空间里面创建的 IPC Object 对该空间内的所有进程可见,但是对其他的空间不可见,这样就使得不同空间之间的进程不能直接通信,就像是在不同的系统里一样。当一个 IPC 空间被销毁后,该空间内的所有 IPC Object 会被内核自动销毁。PID 和 IPC 命名空间可以组合起来一起使用,只需在调用 clone 时,同时指定相应的 FLAG,CLONE_NEWPID 和 CLONE_NEWIPC,新创建的命名空间既是一个独立的 PID 空间,又是一个独立的 IPC 空间。不同命名空间的进程彼此不可见,也不能互相通信,这就实现了进程间的隔离。

(3)MNT 空间为进程提供了一个文件层次视图。如果不设定这个 FLAG,子进程和父进程将共享一个 mount 空间,其后子进程调用 mount 或 umount 将会影响到空间里所有的进程。如果子进程在一个独立的 mount 空间里面,就可以调用 mount 或 umount 建立一份新的文件层次视图。该 FLAG 配合 pivot_root 系统调用,可以为进程创建一个独立的目录空间。

(4)Net 空间为进程提供了一个完全独立的网络协议栈视图。包括网络设备接口,IPv4 和 IPv6 协议栈,IP 路由表,防火墙规则,Sockets,等。一个 Net 空间提供了一份独立网络环境,就跟一个独立的系统一样。

(5)UTS 空间就是一组被 uname 返回的标识符。新的 UTS 空间中的标识符通过复制调用进程所属的空间的标识符来初始化。clone 出来的进程可以通过相关系统调用改变这些标识符。NET 和 UTS 命名空间可以组合起来一起使用,在 clone 的时候设定相应的 FLAG,CLONE_NEWUTS 和 CLONE_NEWNET,可以虚拟出一个有独立主机名和网络空间的环境,就跟网络上一台独立的主机一样。

1.3 LXC 启动过程

LXC 的启动首先设置进程的 uid,gid 和 euid,并开启相应的功能,功能初始化之后解析命令行参数和配置文件,获得要创建容器的一些信息,接着调用 lxc_start 来启动容器,在 lxc_start 中关闭所有打开文件的句柄,然后调用函数 lxc_init 执行初始化操作[11]。

lxc_init 通过调用 lxc_cmd_init 函数新创建一个 socket,并对其进行监听,用于接收外部的命令;然后将容器的状态改为 starting,并写入另一个通道文件;最后通过配置文件为容器设置环境变量,以及给容器分配和创建 tty 设备和 console 设备。

初始化的工作完成之后,接着 LXC 会调用 lxc_spawn 函数来创建子进程,这一步是整个过程的核心。父进程通过 clone 系统调用去创建与 PID、IPC、文件系统相关联命名空间的进程,在执行完 lxc_clone 之后,子进程则执行回调函数 do_start,发送消息给父进程,通知父进程可以开始配置,父进程则开始创建 Cgroups 分组并将子进程加入到该分组中,并根据配置文件对 Cgroups 做相应的配置,最后在新的命名空间里面让新的进程去执行 sbin/init,这样一个容器就被启动[12]。容器启动之后程序会调用 lxc_poll 函数,该函数会通过 epoll 机制来监听容器的状态,宿主机会根据容器内的状态触发相应的动作。整个 LXC 的启动过程见图 3。

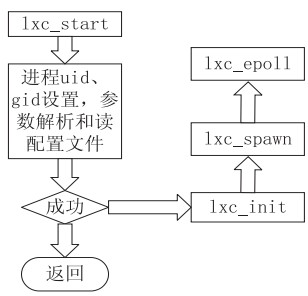

图 3 容器启动过程

依赖于 Cgroups 和 Namespace 的帮助,LXC 才得以实现资源的分配和隔离,每一个容器就是一个虚拟的运行环境,不同的容器之间是透明的。下文将针对 LXC 的分配和隔离的特性进行实验。

2 实 验

通过简单的实验来验证 LXC 的资源管理和隔离的性能,验证 Cgroups 技术对于 CPU 资源的分配和占用的情况,验证一些理论上的想法,如表 1 所示。

表 1 实验环境和工具

物理机	测试工具
4 核 Intel(R) Core(TM) i5-2430M CPU @ 2.40 GHz Ubuntu12.04.1 LTS	apache、压力测试工具 ab、死循环程序

LXC 通过 Cgroups 实现资源分配功能。在 CPU 资源角度,主要有两个变量 cpu. shares 和 cpuset. cpus。简单来说 cpu. shares 是一个使用 CPU 的份额,按照比例计算。例如,两个容器的 cpu. shares 都是 1 024,那么这两个容器使用 CPU 的理论比例就是 50% :50% 。cpuset. cpus 是分配给这个容器使用的具体 CPU。举例来说,一台物理机有 4 个 CPU,分别是 CPU_0、CPU_1、CPU_2、CPU_3,那么可以给容器分配编号为 0-3 的 CPU。

2.1　测试方法

(1)在一个容器内起 apache 服务器,同时在另一个物理节点上使用 ab 进行压力测试,由此监测单个容器的资源隔离和分配。

(2)在两个容器内起死循环程序,调整两个容器资源参数,由此监测、对比多个容器的资源隔离和分配,通过以下 4 种方案进行测试。

方案 1:两个容器共用 1 个 CPU,各自分配 50% ,一个跑死循环、一个闲置;

方案 2:两个容器共用 1 个 CPU,各自分配 50% ,2 个都跑死循环;

方案 3:两个容器共用 1 个 CPU,一个占 80% ,一个占 20% ,2 个都跑死循环;

方案 4:两个容器共用 2 个 CPU,各自分配 50% ,2 个都跑死循环。

(3)通过 top 命令监测各个 CPU 利用率及进程占用 CPU 的利用率。

2.2　实验结果

ab 压力测试的结果如表 2 所示。通过 Cgroups 的资源分配,在 CPU 个数增加的情况下,apache 的相应次数也在增加。死循环的实验如表 3 所示。资源的分配额度是可控的,同时可以看出 LXC 支持资源的抢占,在共享 CPU 的情况下,忙碌的容器会抢占闲置容器的资源。

表 2　ab 压力测试请求数

CPU 个数	响应请求数/sec
1	543.24
2	722.21
3	841.45
4	878.12

表 3　两个容器 CPU 占有率　　%

方案号	容器 1CPU 占有率	容器 2CPU 占有率
1	98	1
2	48	49
3	77	19
4	97	98

3　结束语

文中介绍了一种轻量级的虚拟化技术 LXC,介绍了实现资源分配和隔离的两种机制 Cgroups 和 Namespace,而后对其资源的控制做了测试。如今越来越受欢迎的 Docker 正是 LXC 的扩展,它是一个开源的应用容器引擎,让开发者可以打包他们的应用以及依赖包到一个可移植的容器中,然后发布到任何流行的 Linux 机器上[7]。容器完全使用沙箱机制,相互之间不会有任何接口。几乎没有性能开销,可以很容易地在机器和数据中心上运行。正是这种轻量级,才使 openstack 支持了 Docker 的部署,目前的 LXC 安全性能还有待提高,随着云计算的发展,这种基于容器的虚拟化技术也会慢慢被普及。

参考文献:

[1] 薛海峰,卿斯汉,张焕国. XEN 虚拟机分析[J]. 系统仿真学报,2007,19(23):5556-5558.

[2] 吴　革,李　健,赖英旭. 基于操作系统容器虚拟化技术的 JBS 模型的研究[J]. 网络安全技术与应用,2010(4):39-41.

[3] 吴义鹏. 基于容器的虚拟机调度算法优化及实现[D]. 北京:北京邮电大学,2011.

[4] 金　海,廖小飞. 面向计算系统的虚拟化技术[J]. 中国基础科学,2008,10(6):12-18.

[5] 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 21st Euromicro international conference on parallel, distributed and network-based processing. [s. l.]:IEEE,2013:233-240.

[6] 李安伦. 基于 Xen 隔离的嵌入式 Linux 系统安全增强技术[D]. 南京:南京理工大学,2013.

[7] Soltesz S, Pötzl H, Fiuczynski M E, 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.

[8] Vaughan-Nichols S J. New approach to virtualization is a lightweight[J]. Computer,2006,39(11):12-14.

[9] Deshane T, Shepherd Z, Matthews J, et al. Quantitative comparison of Xen and KVM[M]. Boston, MA, USA: Xen Summit,2008.

[10] Zheng Y, Nicol D M. A virtual time system for openvz-based network emulations[C]//Proc of IEEE workshop on principles of advanced and distributed simulation. [s. l.]: IEEE, 2011.

[11] 丘诗雅. 基于应用虚拟化技术的安全移动办公解决方案[J]. 移动通信,2011,35(17):66-68.

[12] Biederman E, Networx L. Multiple instances of the global Linux namespaces[C]//Proceedings of the Linux symposium. [s. l.]:[s. n.],2006.