分　数：	
评卷人：	

華中科技大學

研究生（数据中心技术）课程论文（报告）

题　目：**Cntr: Lightweight OS Containers** 阅读报告

学　　号　**M201873290**

姓　　名　向程

专　　业　计算机技术

课程指导教师　施展　曾令仿

院（系、所）　计算机学院

2018 年　**11** 月　**27**　日

一、论文背景

本篇文章题名为：《CNTR: Lightweight OS Containers》，由爱丁堡大学的 Jörg Thalheim and Pramod Bhatotia、华盛顿大学的 Pedro Fonseca 以及密歇根大学的 Baris Kasikci 协作完成。发表在 2018 年的 USENIX ATC 会议上。

由于容器相比传统虚拟化技术的种种优势，以及 docker 的出现，容器技术得到了越来越广泛地应用。

容器可以在提供安全的隔离性的同时，拥有比传统虚拟化技术更好的性能表现，同时容器还具有部署速度快，迁移成本低，简化交付流程，拥有更好的伸缩性等等优势，拥有这么多可以降低生产成本的特性，容器的流行也是在所难免的。

然而作者发现容器还可以进一步轻量化，并且提出了一个基于用户空间文件系统（FUSE）切实可行的解决方案。目前已有的一些容器轻量化解决方案，如下：

1. 用更小的 Linux 发行版来当容器的基础，比如 Alpine 发行版，但这种发行版往往功能不够完善，同时不能避免集成过多的工具问题。
2. 将容器构建在联合文件系统（Union filesystem）上，如 UnionFS。用这种技术可以在一种共同的基础镜像上创建多个容器，但是这样的话，应用场景又比较窄。
3. 还有一种方案是全内核（Unikernel），是一种单地址空间的库文件系统，可以理解为进程都运行在内核态，但是他依然会有工具包的额外开销。

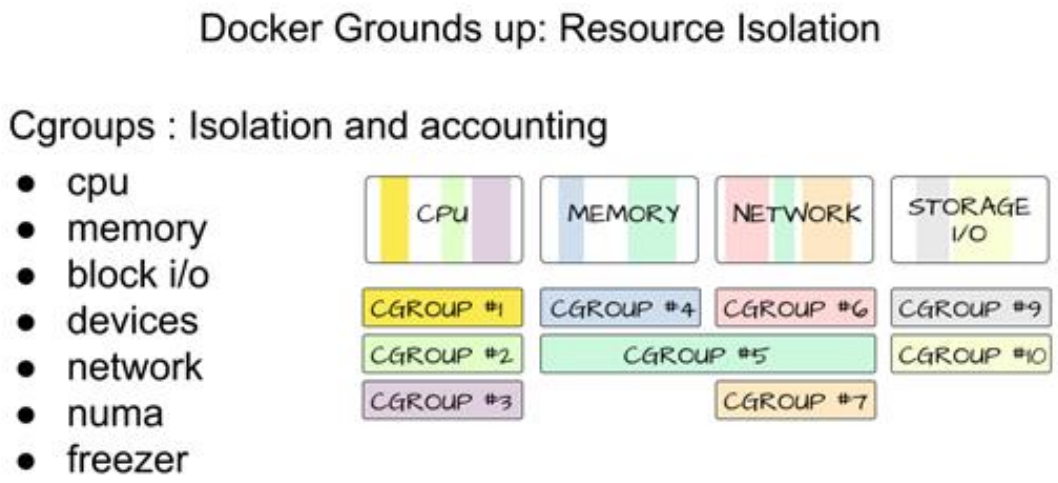
二、容器的底层技术

在阅读这篇论文之前，我们应该先了解下容器的底部到底建立在何种机制之上的，虽然计算机界有句话叫做“计算机的任何问题都可以靠加一个抽象的中间层来解决”，但是我们要想改动这个中间层，就不得不深究下抽象层之下是怎样狰狞的真实了。容器亦是如此。由于容器大多在 Linux 上部署运行，Windows 也并不开源，所以只能了解下基于 Linux 的容器内核机制了。而且这篇论文的具体实现也用到了很多内核机制，所以不了解底层机制也很难读懂这篇文章。

给容器提供底层支持的机制有很多，但最重要的还是 cgroups 以及 namespace，这两个机制给容器提供了最关键的隔离以及资源分配提供了可能。

1. Cgroups

Cgroups 即 Control Group 的缩写，很早就被写进 Linux 内核了，这种机制可以让开发者对某个进程或者进程组进行某些资源的限制，比如 CPU 资源，内存资源，devices 限制等等。在多租户环境，很多不同的容器跑在同一台机器上时，可以很轻松地调节不能进程能使用地系统资源上限，既避免了系统资源的浪费，也使得某些程序无法恶意抢占资源。



Cgroups 对不同资源的限制都是由不同的子系统实现的，每种子系统都需要和不同的模块配合才能实现功能，比如进程调度模块等等。而且由于 Cgroups 是层级结构，Cgroup 之间可以理解为树形架构，所以可以进行十分复杂精细的资源限制。

2. Namespace

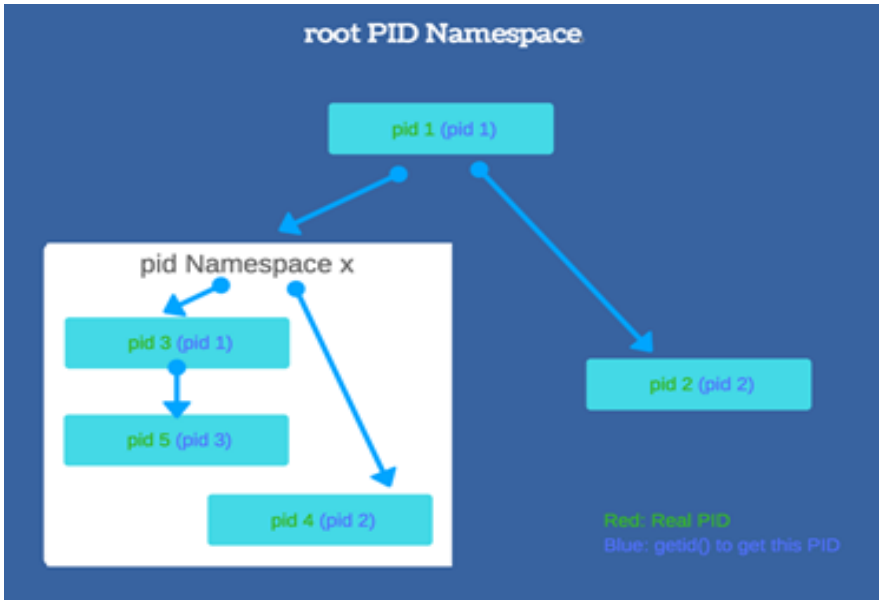
Namespace 即命名空间，它是容器可以在同一个内核上通过容器引擎达到隔离效果的关键技术。Namespace 也是很久以前就并入内核的机制。

目前 Namespace 有 Mount UTS IPC PID Network User Cgroup 七种，分别对应不同的隔离。

类型	功能说明
Mount Namespace	提供磁盘挂载点和文件系统的隔离能力
IPC Namespace	提供进程间通信的隔离能力
Network Namespace	提供网络隔离能力
UTS Namespace	提供主机名隔离能力
PID Namespace	提供进程隔离能力
User Namespace	提供用户隔离能力
Cgroup Namespace	提供控制组隔离能力（Linux4.6 才引入的新特性）

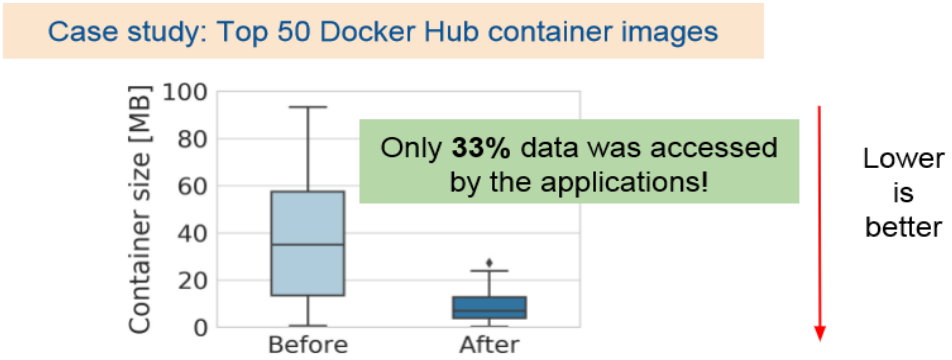
在 Linux 中，大多数东西都是树形层级结构，比如进程，进程号为 1 的为所有其他进程的祖先进程，它们共享一棵进程树，而如果从某个进程下新开一个命名

空间，那么该命名空间内的 1 号进程就会成为这个子树的祖先进程，该命名空间中的进程也无法知晓其他分支进程的存在，由此就达到了进程树之间的隔离，挂载命令空间也是类似，通过对挂载命名空间的隔离，处于子命名空间的会觉得自己独占一个文件系统。



三、现有容器技术的一个问题

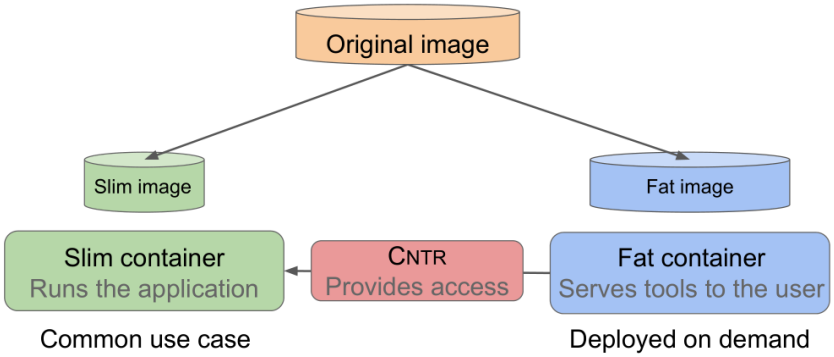
虽然容器相比物理机或者虚拟机式的虚拟化技术已经很轻量级了，但是广泛部署的容器中，依然有大量运行时不会被用上的东西打包进了容器，比如 shell、编辑器和包管理工具等等，核心功能的运行并不需要这些工具，但是在管理、查看容器状态、Debug 等等场景下，这些工具就能排上用场了。



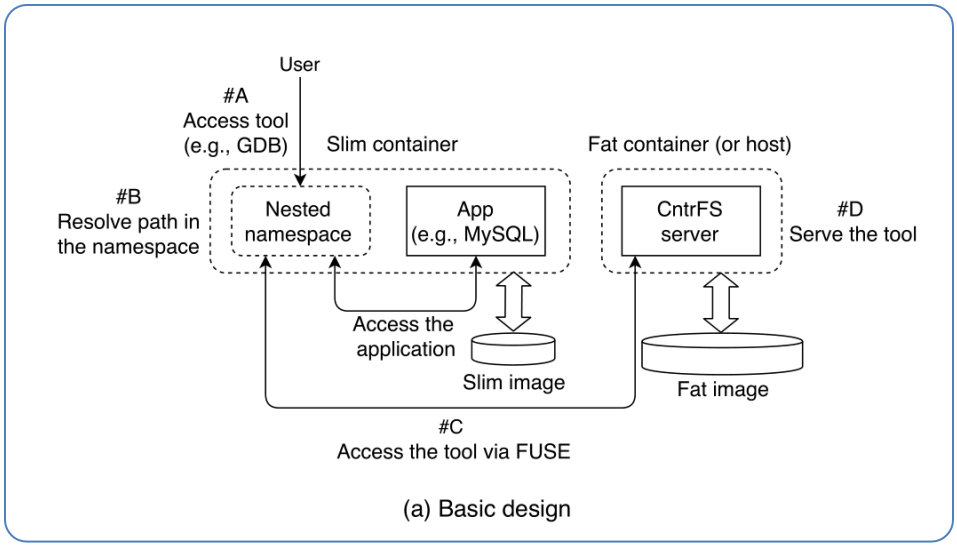
在实践上，这些工具大大增加了容器的大小，使得容器的部署变慢，而且会有很多 CPU、内存以及磁盘的额外开销。考察下 DockerHub 上排名最高的容器，平均来算，大概有 66.6%的空间都被这些运行时无用的工具占据了，所以不是一个可以忽视的问题。

四、解决方案及实现细节

作者提出的解决思路是将一个需要工具包的容器分为两部分，胖容器与瘦容器：胖容器存储 shell，编辑器，Debug 工具等等运行时不必要的工具；而瘦容器只存储运行时需要的应用以及相关依赖。设计用到的关键机制是用户空间文件系统（FUSE），FUSE 实现了兼容虚拟文件系统（VFS）的文件操作接口，然后可以将文件系统在用户态运行。



在绝大部分时间，我只需要运行瘦容器就可以实现功能，少数需要管理容器的时候，就可以在瘦容器里连接胖容器，使用各种工具，所以一个胖容器可以同时供多个瘦容器使用。



具体的实现细节：首先，CntrFS Server 连接胖容器，然后在瘦容器里 fork 一个子进程，用私有挂载连接 CntrFS Server，这里的连接使用用户空间文件系统（FUSE)实现的，在用户通过命令行操作容器时，可以自动进行路径解析，可以同时操作瘦容器或者胖容器，但由于是私有挂载，又能保证隔离性。

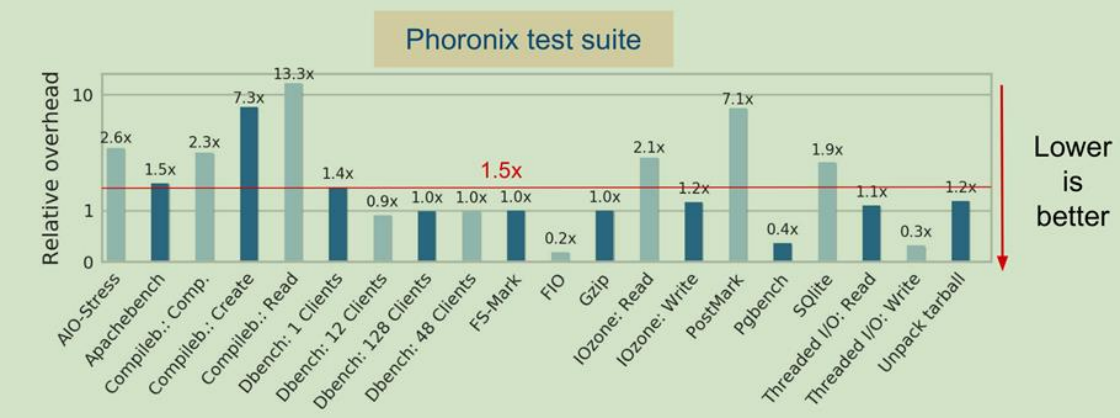
五、评估

但是 Cntr 能否保证完整性、透明性以及高效率呢？首先，Cntr 是透明的，如果一种容器技术如 Docker 或者 LXC 想使用 Cntr，它们无须做任何改变。

作者还用 xfstest filesystem regression test suite 对 Cntr 进行了回归测试，通过了 95.74%的测试用例，而未通过的 Docker 也未能通过，表明 Cntr 是完整的，并不会出现使用了 Cntr 就影响原有的某些功能。

Tests	Supported tests
94	90 (95.74%)

关于性能测试，是在 Amazon EC2 平台租用了 m4.xlarge 虚拟机，该虚拟机拥有双核 Intel Xeon E5-2686 CPU 和 16GB RAM. Linux 内核版本 4.14.13. 100GB EBS GP2（SSD）和 ext4 文件系统。经过对各种参数进行了基准测试，结果如下：



最后可以看出，在大多数情况下，额外开销是可以接受的，但是在 Compileb.: Read 等测试中，额外开销比较高，这是由于在使用内嵌私有挂载命令空间后，进行路径名查找会比较慢，这一项影响了性能，但由于我们只会在极少数情况下才会使用胖容器里的工具，也基本不会做高吞吐量的操作，所以额外开销还是可以忍受的。

六、总结

在这篇文章中，作者是从很小的点来着力的：即怎么进一步降低容器大小而又不影响功能。作者提出的思路即运行时，只在容器里放必要的应用和依赖，而将工具包另放在胖容器中备用，达到降低容器大小而又不影响功能的效果。

经过测试，Cntr 确实实现了承诺的功能，不过由于没有在高负载的生产环境下使用，所以我觉得 Cntr 的性能和稳定性等等方面还需要进一步考察。