基于 LKM 的 Docker 资源信息隔离方法①

陈莉君, 张义飞

(西安邮电大学 计算机学院, 西安 710121)

摘 要: 针对 Docker 容器目前存在的内存资源信息尚未隔离的问题, 设计了一种基于 LKM 技术的资源信息隔离方法. 该方法通过 LKM 的形式利用系统调用劫持来修改读取到的 procfs 文件内容, 从而实现了 Docker 中的容器资源信息隔离的功能, 使得在其上运行的容器可以不用作任何修改就能达到资源信息隔离的目的. 最后通过实验证明资源信息隔离的功能是可用的.

关键词: LKM; Docker; procfs; 容器; 隔离

Docker Resource Information Isolation Method Based on LKM

CHEN Li-Jun, ZHANG Yi-Fei

(School of Computer Science, Xi'an University of Posts and Telecommunications, Xi'an 710121, China)

Abstract: In view of the problem that the memory resource information in Docker container is not isolated, we design a resource information isolation method based on LKM technology. The method in the form of LKM uses system to call hijacking to modify the reading of the procfs file content, so as to realize the function of the Docker container resources information isolation, on which the containers run without any modification can achieve the purpose of resource information isolation. The experiments prove that the resource information isolation function is available.

Key words: LKM; Docker; procfs; container; isolation

随着云计算的普及, 人们对应用的多样化需求促使 PaaS[1]技术的快速发展, 其中云应用的隔离性和资源控制显得尤为重要. 传统云应用的隔离性和资源控制是通过使用虚拟机来实现的, 但是部署在虚拟机中的应用会因为需要经过传统虚拟化技术提供的一个抽象层从而导致应用性能的下降, 在云环境下, 这将会导致基础设施层有效率损失[2-4]. 基于容器的虚拟化技术可以简化应用的部署, 并通过 Linux 内核提供的特性来实现隔离和资源控制, 不需要经过额外的抽象层, 所以在效率上要高于传统的虚拟化技术[2]. 通过隔离可以在单一操作系统上运行多个实例, 实例之间彼此互不影响, 通过资源控制则可以将实例限制在一个资源集合中运行. Docker 充分利用了 Linux 内核提供的 Namespace 和 Cgroups 特性. 但是目前 Linux 内核提供的 Namespace 特性还不够完善[5], 对于 procfs[6,7]文件系统的隔离还没有完全实现, 这导致系统资源信息没有隔离. 容器中的进程通过查看 procfs 文件系统内容或者使用 free 等基于 procfs 文件系统的命令查看系统资源信息的时候获取到的信息都是宿主机的资源信息, 并不是容器本身的资源信息. 一方面对于容器的使用者来说会得到不正确的结果给用户造成困扰, 另一方面对于一些需要获取系统资源信息的软件来说会导致其获取不到正确的资源信息, 例如监控软件的 agent 端.

目前对于 Docker 的资源信息尚未隔离的问题, 业界也有一些解决方案, 但是都必须通过修改程序获取资源信息的接口来解决. 例如 lxcfs[8], Docker API. 对于一些依赖于 procfs 文件系统来获取资源信息的软件来说, 必须修改其资源信息获取接口才能在容器中正常运行以获得正确的资源信息. 没有办法在不修改原有代码的前提下将其移植到容器中了, 此外这些解决方案都是用户态实现, 每次信息请求都会导致多次上

① 收稿时间:2016-04-11;收到修改稿时间:2016-05-23 [doi: 10.15888/j.cnki.csa.005536]

下文的切换和系统调用开销效率低下. 因此本文提出了一种基于 LKM 技术的 Docker 资源信息隔离方法.

1 相关支撑技术

1.1 LKM 技术

LKM[9]是动态扩充内核功能的一项技术. 它使得 Linux 操作系统内核可以在运行状态就能对功能进行扩充. 当编写完一个 LKM 程序, 用编译器将其编译为目标文件,然后就可以根据需要动态的进行加载, 在不需要其所提供的功能时卸载它. 在 LKM 程序编写和编译的过程中无须对内核进行重新编译.

使用 LKM 技术的另一个优点在于对于所有的 Docker 容器都是同效的, 不需要单独为每一个容器进行设置, 因为 Docker 本身就是一种轻量级的虚拟化技术, 所有的容器共用同一个内核, 因此一个内核模块会对所有的容器产生作用.

1.2 cgroups 资源控制机制

Cgroups(Control groups)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的机制, 是由 google 工程师提出, 其初衷是为了管理和控制系统资源. Cgroups 也是 LXC[8]为实现虚拟化所使用的资源管理手段[10]. Docker 也是利用了 Cgroups 这一机制实现了容器的资源管理, Cgroups 包含了九个子系统, 每一个子系统负责管理一种资源.

Linux 内核为了表示子系统, 给每一个子系统都设计一个结构体, 例如内存子系统, 在内核中就有一个对应的结构体叫做 mem_cgroup, 这个结构体负责管理这个子系统对应的 cgroup 的内存资源使用信息. 因此如果要获取一个进程所使用的内存资源信息必须要获得该进程对应的 mem_cgroup. 内核中提供了大量的接口, 可以很方便的通过进程的 task_struct 结构体得到这个进程所对应的 mem_cgroup 结构体.

2 资源信息隔离的设计

2.1 设计概述

本文提出的资源信息隔离方法主要由系统调用截获层、资源信息获取层、procfs 内容覆写层, 三层组成, 其架构模型如图 1 所示.

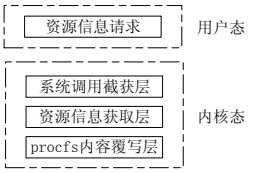

图 1 资源信息隔离架构模型图

该设计框图中, 系统调用截获层屏蔽了上层的多种资源信息 获取方式所带来的差异, 资源信息获取层通过 Linux Kernel 提供的资源信息接口, 得到发起资源信息请求的进程所对应的控制组的资源信息. procfs 内容覆写层通过修改 procfs 内容输出的接口从而将正确的资源信息输出.

2.2 分层设计

系统调用截获层: 该层分为两个部分, 第一个部分是判断发起的请求是否是一个资源获取的请求, 用户可能只是访问一个普通的文件或者一个设备而已, 因此需要根据请求的路径是否是/proc/meminfo 来判定是否是资源获取请求, 第二个部分是通过修改系统调用表 sys_call_table 将 sys_read 替换为本文的 docker_read, 无论上层应用是通过系统命令还是函数接口的方式来获取资源信息, 其最终都会去调用内核中的 sys_read 来获取资源数据. 系统调用截获层通过 LKM 技术将 sys_read 进行了截获, 并替换为本文实现的 docker_read, 从而成功将资源信息获取的逻辑陷入到本文提出的资源信息隔离逻辑中.

资源信息获取层: 该层主要包含两个部分的功能, 第一个部分是判断发起资源请求的进程是否使用了 Cgroups 限制其资源, 第二个部分则是通过 Cgroups 在内核中提供的资源信息获取接口, 得到发起资源请求的进程所在控制组的资源信息结构体, 该结构体中包含了该进程所在控制组中的资源信息.

procfs 内容覆写层: procfs 本身是一种内存文件系统, Linux 内核通过 seq_file 机制[11]给 procfs 中的文件设置了 callback 函数, 当开始读取 procfs 文件中的内容时会自动触发 callback 函数填充文件内容. procfs 内容覆写层通过资源信息获取层得到的控制组资源信息重新实现了 callback 函数, 并进行了注册.

3 资源信息隔离的实现

3.1 资源信息隔离实现

为了验证本文提出的资源信息隔离方法的正确性,

根据给出的架构模型设计并实现了一个详细的资源信息隔离系统, 如图 2 所示.

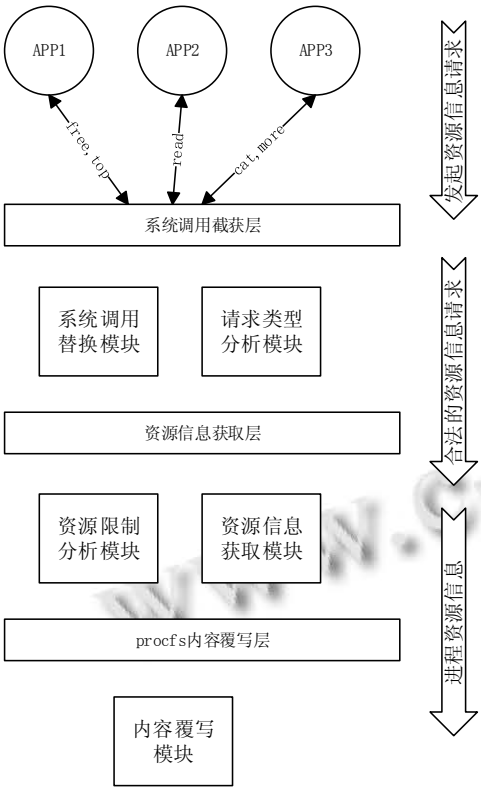

图 2　资源信息隔离系统架构图

该系统架构图中, 系统调用截获层通过对上层应用最终调用的 sys_read 进行劫持, 进而可以通过请求类型分析模块, 根据请求的文件路径是否是/proc/meminfo 来判定此次请求是否是一个资源信息请求, 在判定是一个资源信息请求后就交由资源信息获取层来处理, 在这层中通过资源限制分析模块判定当前发起资源信息请求的进程是否使用了 Cgroups 限制其资源信息, 对于没有使用 Cgroups 进程是不做处理的, 接着会使用资源信息获取模块来获取当前进程所在控制组的资源信息. 最后将这个进程的资源信息交给 procfs 内容覆写层, 在这层中会通过内容覆写模块将进程的资源信息输出到/proc/meminfo 文件中.

3.2 处理流程

本文所实现的资源信息隔离方法的请求处理流程如图 3 所示.

图 3 所示的资源信息隔离方法的执行流程描述如下:

(1) 应用程序发起资源信息获取的请求, 通过 glibc 库最终陷入内核, 调用 sys_read.

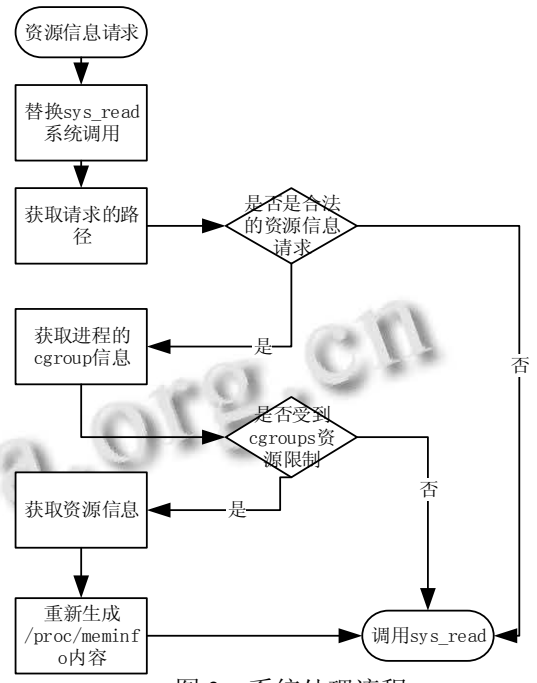

图 3　系统处理流程

(2) 系统调用截获层接受并处理资源信息请求.

a) 请求未到达前修改内核系统调用表 sys_call_table, 替换 sys_read.

b) 判定资源请求是否是对/proc/meninfo 的访问.

(3) 资源信息获取层接收合法的资源信息请求.

a) 获取当前进程的 cgroup 信息, 判断是否存在 cgroups 资源限制.

b) 获取当前进程的资源信息.

(4) procfs 内容覆写层接收当前进程所在控制组的资源信息.

a) 根据资源信息生成 meminfo 文件内容.

b) 输出到/proc/meminfo 文件.

4　实验

4.1 实验环境

为了验证本文所提的基于 LKM 的 Docker 资源信息隔离方法, 设置了如下的实验环境.

表 1　实验环境和工具

物理机	测试工具
4G 内存, Centos7.0, 内核版本 3.10 docker 版本 1.8.2	cat, free, vmstat, top 等命令 使用 read 系统调用编写的文件读写工具

4.2 试验方法

为了测试该方法的正确性和可用性, 笔者首先和

业界现有解决方案在功能上和性能上做了对比, 在功能上主要对比以下三个功能点:

功能点 1: 是否可以获取容器的真实内存信息.

功能点 2: 是否改动软件的资源信息获取接口.

功能点 3: 是否需要单独对每个容器进行设置.

在性能上主要通过对比各个解决方案在 1W, 10W, 100W 次的资源信息请求下所耗费的时间.

最后通过下面的方案来测试本文所提出的方法的正确性. 启动一个 nginx 容器, 并限制这个容器可以使用的最大内存为 512M. 通过以下两种方案进行测试.

方案 1: 使用 cat, free, vmstat, top 等 Linux 标准命令查看容器内存资源信息.

方案 2: 使用笔者编写的文件读取工具查看容器内存资源信息.

方案 3: 不使用本文所采用的方法, 通过 procfs 文件系统来查看容器内存资源信息.

最后通过对比查看到的内存资源信息和通过 cgroups 限制的内存资源信息是否相同.

4.3 实验结果

分别对 lxcfs, Docker API, 以及 DRISO(Docker Resource Information Isolation)即本文所提出的方法进行功能上的对比, 对比结果如表 2 所示.

表 2 功能对比

方法	功能点 1	功能点 2	功能点 3
Lxcfs	是	是	是
Docker API	是	是	否
DRISO	是	否	否

通过上面的结果可知, 本文所提出的 DRISO 在功能上要比现有的解决方案更加完善.

为了证明本文所提出的方法在实际生产环境下可用, 对上述几个解决方案做了性能上的对比, 测试结果如表 3 所示.

表 3 性能测试

方法	1W/μs	10W/μs	100W/μs
Lxcfs	102194	1179267	9698397
Docker API	8086037	76345256	687652451
DRISO	11801	93412	919369
未使用任何方法	7601	67674	648503

通过上面的结果可知, 在不是用任何方法的情况下性能是最好的, 本文所使用的 DRISO 方法性能次之, Docker API 的方式性能最差, 因为该方式每次资源信息请求都是一次完整的 HTTP 请求和响应, 这其中涉及了大量的操作, 而本文的 DRISO 方法, 却只有仅仅几次系统调用和数据从内核空间拷贝到用户空间的开销, lxcfs 性能较好, 因为 lxcfs 是基于文件接口的形式来获取资源信息, 和 procfs 一样也是一个文件系统, 但是因为其是基于 FUSE 的用户态文件系统, 这导致大量的数据拷贝操作, 和上下文切换的开销, 因此性能差于本文所提出的 DRISO 方法.

通过加载本文实现的 LKM 模块使得容器可以通过 Linux 标准命令来获取限制的内存资源信息, 如图 4 所示.

图 4 资源信息隔离前后对比图

上图中首先使用 docker 运行了一个 nginx 容器, 并通过-m 选项限制这个容器的最大可用内存为 512M, 然后运行 free 命令查看这个容器的可用内存信息, 通过结果可以得知查看到的内存资源信息是宿主机的资源信息, 接着通过插入本文所编写的资源隔离内核模块后, 再次运行相同的 nginx 容器并使用 free 命令查看容器内存资源信息, 发现内存资源信息有效的进行了隔离.

为了验证本文提出的基于 LKM 的 Docker 资源信息隔离方法在不同的资源获取方式下都可以正确工作, 分别通过 Linux 标准命令和笔者利用 read 系统调用编写的文件读取工具进行了测试, 方案 1 和方案 2, 3 的测试结果如表 4 所示.

表 4 测试结果

方案号	free 等命令查看到的内存	cgroups 限制的可用内存
1	512M	512M
2	512M	512M
3	4G	512M

综合上面所有的测试结果可知, 本文提出的 DRISO 方法是正确的, 并且是实际生产环境下可用的.

5 结语

本文主要介绍了 Linux 中的 Cgroups 机制, 并在此基础上基于 LKM 的技术实现了 Docker 的资源信息隔

离. 通过实验证明了其正确性. 随着 Docker 容器技术的快速发展, 容器的安全性[12]和隔离性[13]越来越重要, 这对如何在 Linux 操作系统层面加强对容器安全性和隔离性的支持提出了更高的要求.

参考文献

1 Pahl C. Containerization and the PaaS cloud. IEEE Cloud Computing, 2015, (3): 24–31.

2 Felter W, Ferreira A, Rajamony R, et al. An updated performance comparison of virtual machines and linux containers. 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE. 2015. 171–172.

3 Joy AM. Performance comparison between Linux containers and virtual machines. 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA). IEEE. 2015. 342–346.

4 Scheepers MJ. Virtualization and containerization of application infrastructure: A comparison. 21st Twente Student Conference on IT. 2014. 1–7.

5 Menage P. Linux Kernel Documentation/cgroups/cgroups.txt. 2011.

6 Bowden T, Bauer B, Nerin J, et al. The/proc filesystem. Linux Kernel Documentation, 2000.

7 赵付强,李允俊,宫彦磊. Proc 文件系统的研究与应用.计算机系统应用,2013,22(1):87–90.

8 Graber S, Hallyn S. LXC Linux Containers. https://linuxcontainers.org. [2013-05-15].

9 徐伟,贾春福.扩充 Linux 系统功能的 LKM 技术.计算机应用研究,2003,4:100–102.

10 汪恺,张功萱,周秀敏.基于容器虚拟化技术研究.计算机技术与发展,2015,8:138–141.

11 郭松,谢维波.Linux 下 Proc 文件系统的编程剖析.华侨大学学报(自然科学版),2010,5:515–520.

12 Bui T. Analysis of docker security. arXiv preprint arXiv:1501.02967, 2015.

13 刘思尧,李强,李斌.基于 Docker 技术的容器隔离性研究.软件,2015,4:110.