Journal of Computer Applications
计算机应用,2019,39(12):3628-3632
ISSN 1001-9081
CODEN JYIIDU
2019-12-10
http://www.joca.cn

文章编号:1001-9081(2019)12-3628-05
DOI:10.11772/j.issn.1001-9081.2019040765

基于流量控制的 Docker 容器网络带宽控制机制

王志伟[1],杨　超[2*]
(湖北大学 计算机与信息工程学院,武汉 430062)
(*通信作者电子邮箱 stevenyc@hubu.edu.cn)

摘　要:针对 Docker 容器缺乏对网络带宽资源进行限制的能力的问题,提出了一种基于流量控制(TC)的 Docker 容器网络带宽控制机制。首先,基于 CGroups 文件系统的实时监测机制,利用 Linux 内核的虚拟文件系统(VFS)作为媒介,将 Docker 容器创建时设置的网络控制参数传递给 Linux 内核流量控制器 TC;然后,通过引入 IFB 模块实现上下行带宽控制,并使用 rate、ceil 及 prio 参数进行空闲带宽共享及容器优先级控制;最后,控制 TC 执行具体的网络限制,以实现容器之间灵活的网络资源控制。实验结果表明,该机制在容器独占带宽场景下可有效地将实际容器带宽限制在2%的波动范围内,而在共享空闲带宽场景下可在平均误差0.5%的范围内精准限制容器带宽,同时该机制能够基于优先级弹性地管理资源。该机制具有提供更为原生的接口且无需额外工具配合的优势,可为基于 Docker 的云平台的细粒度弹性网络资源控制提供便捷有效的解决思路。

关键词:Docker 容器;资源控制;网络带宽;CGroups 机制;流量控制
中图分类号:TP393　　**文献标志码**:A

Bandwidth control mechanism for Docker container network based on traffic control

WANG Zhiwei[1], YANG Chao[2*]
(*School of Computer Science and Information Engineering, Hubei University, Wuhan Hubei* 430062, *China*)

Abstract: As Docker container lacks the ability of limiting network bandwidth resources, a bandwidth control mechanism was proposed for Docker container network based on Traffic Control (TC). Firstly, based on the real-time monitoring mechanism of CGroups file system, Virtual File System (VFS) of Linux kernel was used as a medium to pass the network control parameters set when Docker container was created to the Linux kernel controller TC. Then, the Intermediate Functional Block device (IFB) module was introduced to archive uplink and downlink bandwidth control, and the parameters (rate, ceil and prio) were used to achieve idle bandwidth sharing and container priority control. Finally, the specific network limitations were conducted by controlling the TC, and flexible network resource control between containers was realized. The experimental results show that the proposed mechanism can effectively limit the actual container bandwidth within 2% fluctuation range in the container exclusive bandwidth scenario, and can precisely limit the network bandwidth of the container with average 0.5% error range in the shared idle bandwidth scenario. Meanwhile, the mechanism can flexibly manage resources based on priorities. With the advantage of providing a more native interface for Docker and requiring no additional tools, this mechanism can provide a convenient and effective solution for fine-grained elastic network resource control on Docker-based cloud platform.

Key words: Docker container; resource control; network bandwidth; CGroups mechanism; Traffic Control (TC)

0　引言

Linux 容器技术是一种轻量级操作系统层的虚拟化技术。相对于传统的虚拟机,Linux 容器具有快速部署、轻量化、低成本、低资源消耗等显著优势。其中,Docker 容器是目前最成功的开源容器产品[1-3]。Docker 具备持续集成、版本控制、可移植性、隔离性和安全性等重要优势,众多企业和个人使用 Docker 来部署、测试、开发应用,同时基于 Docker 来部署云平台已经是主流趋势[4-6]。Docker 容器技术被应用在越来越多的领域[7-10]。

目前,Docker 仍缺乏对容器网络带宽进行限制的能力,增强 Docker 对于网络带宽资源进行控制的能力,并且采用一种弹性和基于优先级的网络带宽资源管理方式对 Docker 云平台的资源管理至关重要[11]。Zheng 等[12]指出 Docker 默认的本地访问控制是非常弱的,默认的安装策略包含对网卡设备和文件系统的全部访问权限;但是相较于在单个容器级别限制对网卡的使用而言,对整个 Docker 限制网卡设备的使用是低效并且不够灵活的。Khalid 等[13]的研究表明如果不限制 Docker 容器对网络的使用,攻击者可以通过大量使用网络资源来影响统一主机下其他用户的使用。在此基础上,Khalid

收稿日期:2019-05-06;**修回日期**:2019-07-12;**录用日期**:2019-07-19。
基金项目:国家自然科学基金资助项目(61170306);智能信息处理与实时工业系统湖北省重点实验室开放基金资助项目(znxx2018MS05)。
作者简介:王志伟(1982—),男,北京人,硕士研究生,主要研究方向:信息安全;　杨超(1982—),男,湖北武汉人,副教授,博士,CCF 会员,主要研究方向:信息安全。

等[13]又提出一种基于硬件的策略来改变 Docker 对网络使用计时的方案,但是由于引入了特殊的硬件,所以对宿主机提出了新的要求。

本文在 Linux 内核的层面设计一种精准的 Docker 容器网络带宽(以 Kb/s 计算)控制系统,解决 Docker 对容器网络带宽资源缺少限制的问题,进而有效地实现容器之间网络资源的均衡,其优势在于:1)粒度细。限制是作用于容器实力级别而不是整个 Docker 系统。2)控制精准。能够以 Kb/s 的精准程度控制容器的上行和下行速度。3)对系统要求低。能够在不借助额外硬件的帮助下完成,更适合现行硬件条件的宿主机环境。该系统具备以下功能:

1)支持容器的网络带宽上行、下行带宽在容器启动时被独立设定。

2)保证能够利用到被分配到的 IO 带宽额度。

3)具备弹性资源控制能力,在网络带宽有空闲资源时,可以按照主机额度来均分其他 CGroups 的空闲带宽,不会浪费 IO 带宽资源。

4)支持为不同服务的容器设定优先级,确保对时延要求较高的服务的服务质量。

1　相关工作

1.1　Docker 现有资源限制的情况

在 Docker 中,默认情况下容器的资源只受到 host 端内核资源调度的限制,但是可以在使用 Docker run 命令启动容器时添加一些资源控制标志来控制容器的 Memory、CPU 以及 Disk IO 资源的分配。Docker 借助了 Linux 内核的 namespace 对资源进行隔离,例如 CPU 资源、磁盘资源、网络资源,然后又借助 Linux 内核的 CGroups 对隔离的资源施加限制。由于 Linux 内核虽然支持对网络资源进行隔离,却并没有支持对网络资源的使用进行限制,所以现有的方案一般都采用 Docker 搭配其他网络资源限制的工具使用。

本文基于 Linux 内核流量控制(Traffic Control, TC)[14]为 Docker 建立 TC 队列,并为 Docker 容器建立分类和过滤器,使得每个容器的数据包都通过过滤器来确定分类,并最终达到限制容器网络带宽的目的。该方案能较为精准地控制容器的上行和下行带宽,并且支持共享限制带宽和为容器设置优先级。同时该方案能为 Docker 提供更为原生的使用接口,直接通过 Docker 参数指定而无需额外工具的配合。

1.2　Linux 内核流量控制器 TC

Linux 操作系统中的 TC 用于 Linux 内核的流量控制,它利用队列规定建立处理数据包的队列,并定义队列中的数据包被发送的方式,从而实现对流量的控制。TC 模块实现流量控制功能使用的队列规定分为两类:一类是无类队列规定,另一类是分类队列规定。无类队列规定相对简单,而分类队列规定则引出了分类和过滤器等概念,使其流量控制功能增强。无类队列规定是对进入网络设备(网卡)的数据流不加区分统一对待的队列规定。使用无类队列规定形成的队列能够接收数据包以及重新编排、延迟或丢弃数据包。这类队列规定形成的队列可以对整个网络设备(网卡)的流量进行整形,但不能细分各种情况。常用的无类队列规定主要有先进先出(First Input First Output, FIFO)、令牌桶过滤器(Token Bucket Filter, TBF)、随机公平队列(Stochastic Fairness Queue, SFQ)等。这类队列规定使用的流量整形手段主要是排序、限速和丢包。

分类队列规定是对进入网络设备的数据包根据不同的需求以分类的方式区分对待的队列规定。数据包进入一个分类的队列后,它就需要被送到某一个类中,也就是说需要对数据包作分类处理。对数据包进行分类的工具是过滤器,过滤器会返回一个决定,队列规定就根据这个决定把数据包送入相应的类进行排队。每个子类都可以再次使用它们的过滤器进行进一步的分类,直到不需要进一步分类时,数据包才进入该类包含的队列排队。除了能够包含其他队列规定之外,绝大多数分类的队列规定还能够对流量进行整形。这对于需要同时进行调度(如使用 SFQ)和流量控制的场合非常有用。Linux 流量控制主要分为建立队列、建立分类和建立过滤器三个方面,基本实现步骤如下:

1)针对网络物理设备(如以太网卡 eth0)绑定一个队列 qdisc,在该队列上建立分类 class;

2)为每一分类建立一个基于路由的过滤器 filter;

3)最后与过滤器相配合,建立特定的路由表。

2　网络带宽控制系统实现

2.1　系统总体架构

本文方案主要是基于 TC 来实现对 Docker 的网络带宽进行限制,基于 Linux 的内核模块可以使依赖降到最低,同时通过在 Docker 内集成 TC 能够更加准确地收集容器信息从而精准地控制网络带宽,另外可以向 Docker 提供和原生类似的一致接口。

Linux 内核从 2.2 版本开始已经支持使用 TC 命令实现了流量控制的功能,TC 功能十分强大,利用 TC 完全能够实现对网络带宽的控制。但是为了使用户态工具 TC 和 Docker 进行协调配合,针对网络带宽设计了一种基于 CGroups 文件系统实时监测机制,利用内核 CGroups 系统实现的虚拟文件系统(Virtual File System, VFS)作为媒介,将 Docker 容器创建时设置的网络控制参数传递给 TC。所设计系统的技术流图如图 1 所示。

图 2 展示了网络带宽控制系统的总体架构。在用户空间,通过定制 Docker 源码,实现对混合 CGroups 的支持,同时利用 CGroups 文件系统为媒介,用户态网络带宽控制模块能够感知 Docker 容器设置的变化,进行相应网络带宽控制。各模块功能如下:

1)网络带宽 Controller:作为用户态 Linux 守护进程,负责与 Docker daemon 配合进行网络带宽的控制。

2)Net CGroups detector:监测 CGroups 文件系统内 net_ctls 下 Docker 容器网络控制的变化,能够读取网络控制参数,将参数配置传给 TC controller 模块。

3)TC controller:接收从 Net CGroups detector 模块传递的网络控制指令和具体参数,控制 TC 执行具体的网络限制。

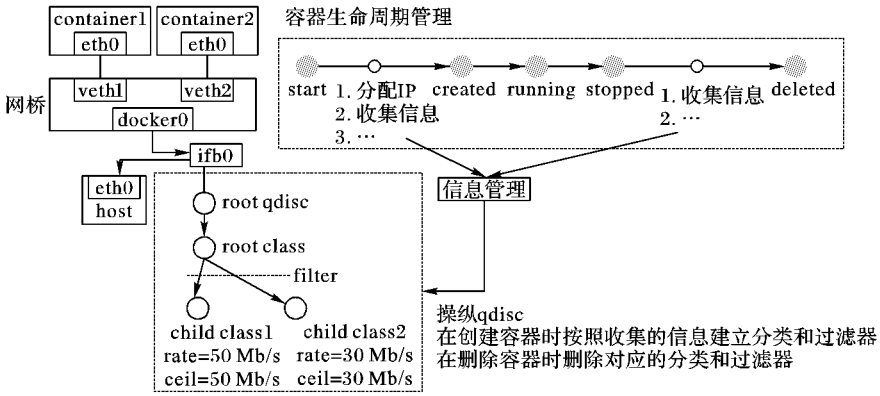

图 1 技术流图

Fig. 1 Technical flow diagram

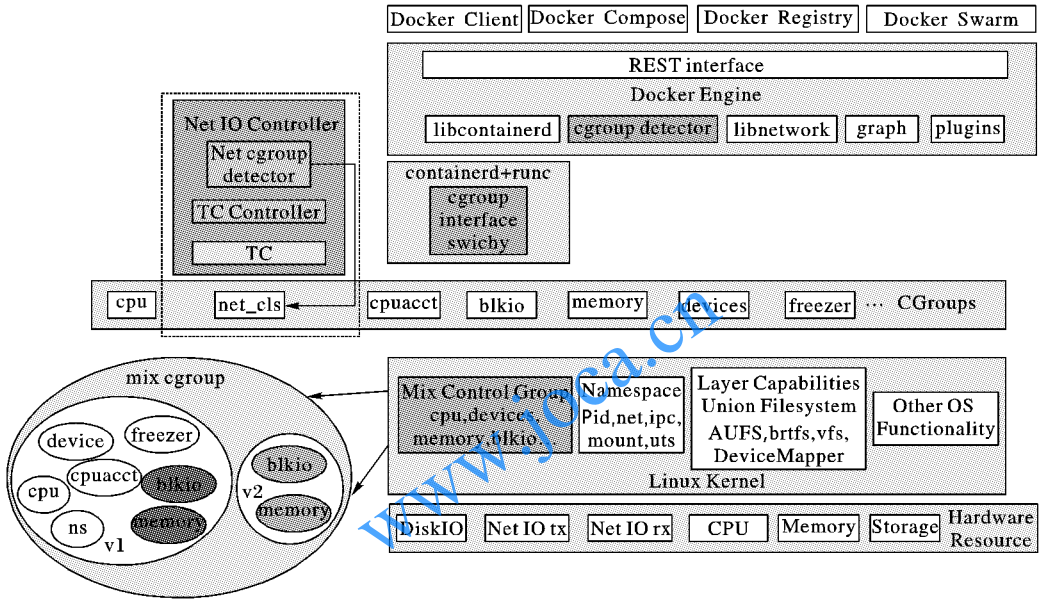

图 2 网络带宽控制系统总体设计架构

Fig. 2 Overall design architecture of network bandwidth control system

2.2 限制容器的上限带宽

Docker 启动容器时默认的网络模式是 bridge，该模式也是业界应用最广泛的一种网络模式，该网络模式的拓扑如图 3 所示。Docker 在主机中建立了一个叫作 docker0 的网桥，而后容器与主机之间的流量都是通过这个网桥进行转发的。由于 TC 工具只能对网卡的发送速率进行限制，因此容器的上下行带宽设置方法有略微的不同，这里分开讨论。

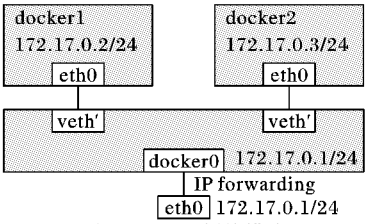

图 3 Docker 桥接模式

Fig. 3 Bridge mode of Docker

对于容器的下行流量，数据包是从 eth0 发送到 docker0，而后再通过 docker0 发送到 docker1(172.17.0.2)上。因此只要限制了 docker0 发往 docker1(172.17.0.2)的速率，就可以限制容器 docker1 的下行带宽了。具体要做的操作有：

1)使用 TC 工具为 docker0 网桥添加一个 qdisc，具体使用的是分层令牌桶(Hierarchical Token Bucket, HTB)[15]。

2)在这个 qdisc 上建立一个根分类(例如 2:1)，可以选择设置 docker0 网桥的总可用带宽。

3)为容器建立子分类(例如 2:2)，同时设置该分类的带宽。

4)利用 TC filter 将目的地址为容器的 IP 的流量都归类为 2:2。

对于容器的上行流量，使用了 Linux 内核的 IFB(Intermediate Functional Block device)模块，手动加载 IFB 模块后，系统的网络中将多出一个 ifb0 的设备，需要将 docker0 网卡从容器端接受到的所有流量都转发到 ifb0 网卡，然后再用类似的方法限制带宽。具体要做的操作有：1)加载 IFB 模块；2)启动 ifb0 设备；3)将 docker0 的流量转发到 ifb0 设备上；4)为 ifb0 添加了 qdisc，本文使用的 HTB；5)在这个 qdisc 上建立根分类(3:1)，可以选择设置所有容器总可用上行带宽；6)为容器建立子分类(3:2)，同时设置该分类的带宽；7)利用 TC filter 将源地址为容器的 IP 的流量都归类为 3:2。

2.3 支持容器共享空闲带宽

在实际的生产环境中，有部分容器带宽使用率很低，而有

些容器的带宽利用率很高,这就造成了资源的浪费,因此需要支持容器共享空闲带宽。在使用 TC 工具为容器建立子分类时,可以限制容器的带宽,限制带宽有 rate 和 ceil 两种选项:rate 选项设置的带宽是容器能够被保证的带宽,而 ceil 选项设置的带宽是在父分类有空闲带宽时,容器能够得到的最大的带宽。因此,如果需要设置容器可以共享空闲带宽,只要调整这个 ceil 值,让它大于 rate 值;如果不想让容器共享空闲带宽,那么只要设置这个 ceil 值与 rate 值相等即可。

2.4　支持为每个容器设定优先级

Docker 提倡一个容器只负责一种服务,不同的服务对网络带宽和时延的要求也不同,因此需要支持为容器设立优先级,在网络出现拥堵的情况下,具有高优先级容器的数据包将被优先处理,确保较低的延时,保证容器的服务质量。TC 工具支持为分类设置 prio 参数,该参数即可实现我们的需求,参数值越小优先级越高,优先级高的分类数据包将被优先发送。

3　实验结果与分析

3.1　实验环境

为了验证本文系统在限制容器上限网络带宽、按优先级分配网络带宽和弹性分配带宽等方面的效果,在实际的物理机上进行了实验验证。实验结果表明,本文系统能够精确限制容器网络带宽,并按比例将空闲网络带宽分配给各个容器。

实验首先测试对容器带宽的上限进行限制。在未限制带宽的基准测试中,容器全力下载则会耗尽所有带宽,所以下载速率非常高,通过限制上限之后,即使容器全力下载其下载速率也不能突破施加的限制,从而说明施加的上限是有效的。

在验证共享带宽的实验中,主机中同时启动两个容器,并且其对两者施加的上限之和小于主机的带宽上限。在这种资源闲置的情况下,通过设置不同的参数希望能够使得容器突破上限使用闲置资源,所以容器的实际带宽可能略大于上限,并且两个容器实际使用的带宽之和接近主机的带宽上限,从而说明容器能够有效利用空闲带宽。

在验证容器优先级的实验中,在上一个实验环境的基础上进行。对上一实验中的两个容器设置不同的优先级,原本两个容器都能共享主机中空闲的带宽,现在优先级高的容器将优先使用这些空闲带宽,在高优先级容器结束后低优先级容器能继续使用空闲带宽,从而说明对容器的优先级设定是有效的。

3.2　限制容器的上限带宽

为了测试系统对容器带宽的限制作用,在 Docker 中启动一个容器后,不对它进行任何的限制,在 10 s 的时间间隔内,每 0.5 s 打印该时段内传输字节量以及平均的传输速率,作为基准线。重新启动容器,并将 rate 设置为 50 Mb/s,其余 ceil、prio 参数默认为无效,以同样的时间间隔打印测试结果,结果如图 4 所示。

从图 4 中可以观察到,刚启动容器时,传输速率较大,但会迅速下落。当不作任何带宽限制时,实际传输速率波动幅度较大,总体保持在较高水平。当限制带宽为 50 Mb/s,启动容器后实际传输速率能迅速回落至设定值,且波动幅度极小,不超过平均传输速率的 2%,限制效果较为精准。

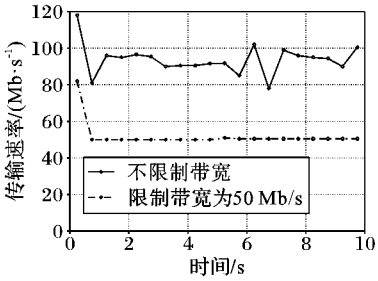

图 4　容器网络带宽限制

Fig. 4　Bandwidth limitation in container networks

3.3　容器共享空闲带宽

前面提到,参数 rate 设置了容器的上限带宽,若同时设置参数 ceil(大于等于 rate 的一个值),则在父分类有空闲带宽时,会实际达到不超过 ceil 值的带宽利用。为测试容器共享空闲带宽的情况,启动一个容器后,并建立两个子分类,其中子分类 1 的参数设置为 rate = 20 Mb/s,ceil = 20 Mb/s,即严格限制带宽为 20 Mb/s;子分类 2 的参数设置为 rate = 50 Mb/s,ceil = 50 Mb/s。同样在 10 s 的时间间隔内,每 0.5 s 打印该时段内传输字节量以及平均的传输速率,作为参考基准。

重新启动一个容器,建立两个子分类,其中:子分类 1 的参数设置为 rate = 20 Mb/s,ceil = 120 Mb/s(大于容器的总带宽);子分类 2 的参数设置为 rate = 50 Mb/s,ceil = 120 Mb/s。两次实验结果如图 5 所示。

从图 5 中可知,在设置 ceil 为一个较大参数的情况下,容器中的子分类会利用空闲带宽,并且在默认优先级的情况下,几乎等额分配空闲带宽。图中 rate = ceil = 20 Mb/s 的数据波动不可忽视,原因在于:每个 0.5 s 的时间段内打印的传输数据并不准确是这个时间段内传输的数据,存在上个时间段部分传输的数据记录到这个时间段,而这个时间段内部分传输的数据被记录到下一个时间段内的情况,因此数据并不准确维持在 20 Mb/s 而略有波动,但总体还是在 20 Mb/s 附近。实验 10 s 的总时间段内平均传输速率为 20.1 Mb/s,误差约为 0.5%,与设置数据精准符合,故上述波动认为是可接受的。

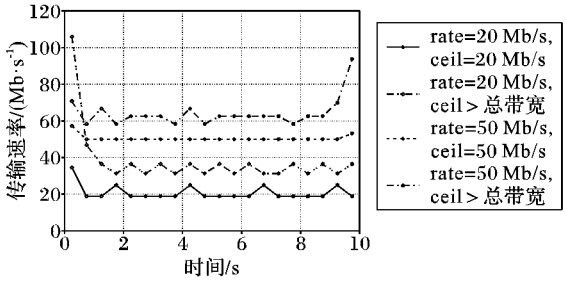

图 5　空闲网络带宽分配

Fig. 5　Idle network bandwidth allocation

当利用空闲带宽时,测试数据波动幅度增大,符合预期。当 rate = 50 Mb/s,ceil 大于总带宽,在测试的最后时段数据有明显较大的增幅,原因在于:启动容器并建立子分类的过程中,由于是手动设置参数并开始进程,因此两个子分类有可观的时间先后差。在该数据的最后两个测试时间点,rate = 20 Mb/s的子分类已结束进程,释放了带宽空间,容器中有了更多的空闲带宽,因此该子分类的传输速率迅速增大。

3.4　容器优先级设定

为了测试优先级(即参数 prio)对数据传输速率的影响,

对容器中的两个子分类配置如下参数:子分类 1 中 rate = 20 Mb/s,prio = 1;子分类 2 中 rate = 50 Mb/s,prio = 3。其中 ceil 设置为 120 Mb/s(大于容器总带宽),同样在 10 s 的时间间隔内,每 0.5 s 打印该时段内传输字节量以及平均的传输速率。将该数据与 3.2 节中得到的参考基准数据对比,结果如图 6 所示。

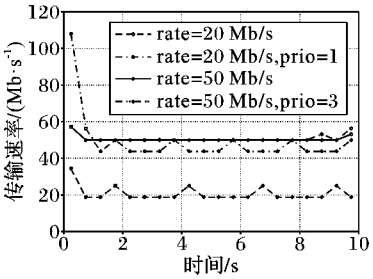

图 6 不同优先级的容器网络带宽

Fig. 6 Network bandwidth of containers with different priorities

参数值越小优先级越高,优先级高的分类数据包将被优先发送。在图 6 中,即 rate = 20 Mb/s、prio = 1 的子分类优先级更高,在容器有空闲带宽的情况下,占满了空闲带宽,而优先级较低(prio = 3)的子分类仅使用了保证的 50 Mb/s 带宽。

在测试的最后时段,优先级更高的子分类先结束进程(由于手动设置而导致两个子分类开始进程的时刻存在时间差),优先级低的子分类开始使用容器内的空闲带宽,因此子分类 2(rate = 50 Mb/s、prio = 3)的传输速率在最后略有提升。

4 结语

Docker 并没有给出网络限制的方案,因为目前网络是通过插件来实现的,和容器本身的功能相对独立,不容易实现,扩展性也很差。Docker 社区已经有很多呼声并且有很多关于添加网络带宽限制的提议,腾讯的 Docker 云平台 GaiaStack 也探索了一些对网络资源进行限制的方法。

本文系统基于 CGroups 文件系统实时监测机制,利用内核 CGroups 系统实现的 VFS 文件系统作为媒介,将 Docker 容器创建时设置的网络控制参数传递给 TC,进而控制 TC 实现具体的网络控制。经测试可知,本文的系统对于网络带宽的限制可以达到精准的控制,还可以对容器任务定义优先级,在保证其他容器带宽的前提下,优先分配空闲带宽给优先级高的容器,保证优先级高的分类数据包被优先发送。但 Docker 中其他资源的限制都是在内核层面实现的,因此本文的系统没有从内核的本质上解决网络限制的问题,未来将在内核层面作更深入的研究。

参考文献 (References)

[1] BOETTIGER C. An introduction to docker for reproducible research, with examples from the renvironment [J]. ACM SIGOPS Operating Systems Review, 2015, 49(1): 71 - 79.

[2] 王亚玲,李春阳,崔蔚,等. 基于 Docker 的 PaaS 平台建设[J]. 计算机系统应用,2016,25(3):72 - 77. (WANG Y L, LI C Y, CUI W, et al. Construction of Docker-based PaaS [J]. Computer Systems & Applications, 2016, 25(3): 72 - 77.)

[3] BERNSTEIN D. Containers and cloud: from LXC to Docker to Kubernetes [J]. IEEE Cloud Computing, 2014, 1(3): 81 - 84.

[4] LIU D, ZHAO L. The research and implementation of cloud computing platform based on Docker [C]// Proceedings of the 11th International Computer Conference on Wavelet Active Media Technology and Information Processing. Piscataway: IEEE, 2014: 475 - 478.

[5] KAN C. DoCloud: an elastic cloud platform for Web applications based on Docker [C]// Proceedings of the 18th International Conference on Advanced Communication Technology. Piscataway: IEEE, 2016: 478 - 483.

[6] ISMAIL B I, GOORTANI E M, AB KARIM M B, et al. Evaluation of Docker as edge computing platform [C]// Proceedings of the 2015 IEEE Conference on Open Systems. Piscataway: IEEE, 2015: 130 - 135.

[7] ABDELBAKY M, DIAZ-MONTES J, PARASHAR M, et al. Docker containers across multiple clouds and data centers [C]// Proceedings of the 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing. Piscataway: IEEE, 2015: 368 - 371.

[8] BELLAVISTA P, ZANNI A. Feasibility of fog computing deployment based on Docker containerization over RaspberryPi [C]// Proceedings of the 18th International Conference on Distributed Computing and Networking. New York: ACM, 2017: Article No. 16.

[9] CHUNG M T, QUANG-HUNG N, NGUYEN M T, et al. Using Docker in high performance computing applications [C]// Proceedings of the 2016 IEEE 6th International Conference on Communications and Electronics. Piscataway: IEEE, 2016: 52 - 57.

[10] LIU Q, ZHENG W, ZHANG M, et al. Docker-based automatic deployment for nuclear fusion experimental data archive cluster [J]. IEEE Transactions on Plasma Science, 2018, 46(5): 1281 - 1284.

[11] XIE B, SUN G, MA G. Docker based overlay network performance evaluation in large scale streaming system [C]// Proceedings of the 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference. Piscataway: IEEE, 2016: 366 - 369.

[12] ZHENG Z, WU X, ZHANG Y, et al. QoS ranking prediction for cloud services [J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(6): 1213 - 1222.

[13] KHALID J, ROZNER E, FELTER W, et al. Iron: isolating network-based CPU in container environments [C]// Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2018: 313 - 328.

[14] 龙恒. Linux 流控工具 TC 的原理及实用案例分析[J]. 计算机与现代化,2010(11):80 - 84, 88. (LONG H. TC principle of Linux traffic control tool and it's practical case analysis [J]. Computer and Modernization, 2010(11): 80 - 84, 88.)

[15] 李晓利,郭宇春. QoS 技术中令牌桶算法实现方式比较[J]. 中兴通讯技术,2007,13(3):56 - 60. (LI X L, GUO Y C. Comparison between token bucket algorithms in QoS technology [J]. ZTE Communications, 2007, 13(3): 56 - 60.)

This work is partially supported by the National Natural Science Foundation of China (61170306), the Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System (znxx2018MS05).

WANG Zhiwei, born in 1982, M. S. candidate. His research interests include information security.

YANG Chao, born in 1982, Ph. D., associate professor. His research interests include information security.