

RainbowD: 一种异构云环境下高效的 Docker 镜像分发系统

张 浩^{1),2)} 孙毓忠^{1),2)} 肖 立^{1),2)} 唐 勇³⁾ 胡满满^{1),2)}
杜沁园⁴⁾ 蔡志彬⁵⁾ 冯百明³⁾

¹⁾(中国科学院计算技术研究所 北京 100190)

²⁾(中国科学院大学 北京 100049)

³⁾(西北师范大学计算机科学与工程学院 兰州 730070)

⁴⁾(中国科学技术大学软件学院 合肥 230000)

⁵⁾(北京工商大学计算机与信息工程学院 北京 100048)

摘 要 近年来,互联网技术飞速发展,互联网用户的数量呈指数级增长;随之而来的是用户需求的服务数量越来越大、服务种类也越来越多. 不仅如此,虚拟化技术也在推陈出新,Docker 以其轻量性、可移植性好等优良特性备受开发者青睐,Docker 在虚拟化技术中扮演越来越重要的角色;为了提升服务的部署效率,越来越多的人选择使用 Docker 在云环境中部署服务. 在集群中部署服务时需要将具有对应服务的镜像分发到节点上,如何高效地分发 Docker 镜像成了研究的热点问题. 然而,在当代互联网发展的趋势下,集群呈现规模大、异构性强的特点;尤其是异构性比较明显的集群,节点的类型多种多样,节点的性能更是参差不齐,低性能节点的不恰当使用会严重制约 Docker 镜像分发的效率. 为此,在本文中,我们基于实际应用场景的真实特性,提出了一种异构云环境下高效的 Docker 镜像分发方案,并实现成 Docker 镜像分发系统,记为 RainbowD. RainbowD 采用了一种新型的高并发机制处理镜像分发的任务;首先,发挥集群中所有节点的作用来减轻仓库节点的负载;其次,使用一种性能优先级排序的方法动态的选择性能高的节点,在有限的资源下提升镜像分发效率;最后,由于对节点进行优先级排序,集群中每个节点不需要再向其余所有节点发送消息,并首次使用 SSH 协议分发镜像,解决了 P2P 网络中存在的消息延迟、消息重复和冗余等问题,减少了带宽资源的浪费. 特别地,由于 BitTorrent 协议的安全性的弊端备受学者诟病,考虑到云环境下集群的安全性问题,利用 SSH 协议加密特性和安全性验证,提升了云环境的安全性. 在本文中,我们设置了两个独立实验,结果如下:首先,RainbowD 和其他镜像分发方案对比验证其镜像分发效率,在实验的异构云环境下,RainbowD 与使用 Docker 的方案相比,镜像分发效率显著提升,分发效率提高了 30% 以上,分发耗时最多缩短了 80%;和 Dragonfly 相比,RainbowD 在 70% 的情况下效率近似或优于 Dragonfly;在集群规模为 100 个节点、分发 200M 的镜像耗时可以达到 150s;其次,RainbowD 可以有效的抵御常见的网络攻击,如 DNS 欺骗和会话劫持攻击. 从实验结果可以看出,RainbowD 在镜像分发的任务中具有高效性和安全性.

关键词 Docker 技术;异构云环境;镜像分发;高并发;SSH 协议

中图法分类号 TP18 **DOI 号** 10.11897/SP.J.1016.2020.02067

RainbowD: A Heterogeneous Cloud-Oriented Efficient Docker Image Distribution System

ZHANG Hao^{1),2)} SUN Yu-Zhong^{1),2)} XIAO Li^{1),2)} TANG Yong³⁾ HU Man-Man^{1),2)}
DU Qin-Yuan⁴⁾ CAI Zhi-Bin⁵⁾ FENG Bai-Ming³⁾

¹⁾(Institute of Computing Technology Chinese Academy of Sciences, Beijing 100190)

²⁾(University of Chinese Academy of Sciences, Beijing 100049)

³⁾(College of Computer Science & Engineering, Northwest Normal University, Lanzhou 730079)

⁴⁾(School of Software Engineering, University of Science and Technology of China, Hefei 230000)

⁵⁾(School of Computer and Information Engineering, Beijing Technology and Business University, Beijing 100048)

收稿日期:2019-06-07;在线发布日期:2019-12-20. 本课题得到“面向云计算的网络化操作系统”(2016YFB1000505)、国家自然科学基金委员会(NSFC)-广东省人民政府联合基金超级计算科学应用研究专项计划(第二期)(U1611261)资助. 张 浩,硕士研究生,主要研究兴趣为云计算和机器学习. E-mail: zhanghao17s@ict.ac.cn. 孙毓忠(通信作者),博士,研究员,中国计算机学会(CCF)会员,主要研究领域为大数据智能(机器学习)分析与计算. E-mail: yuzhongsun@ict.ac.cn. 肖 立(通信作者),博士,副研究员,中国计算机学会(CCF)会员,主要研究方向为人工智能、医学影像、计算生物. E-mail: xiaoli@ict.ac.cn. 唐 勇,硕士研究生,主要研究兴趣为云计算和数据挖掘. 胡满满,硕士研究生,主要研究兴趣为机器学习和数据挖掘. 杜沁园,硕士研究生,主要研究兴趣为软件工程以及云原生数据库. 蔡志彬,主要研究兴趣为云计算. 冯百明,博士,教授,主要研究兴趣为分布式与并行计算.

Abstract In recent years, the rapid development of Internet technology has led to an exponential increase in the number of Internet users. As a result, the number of services required by users is increasing, and the types of services are increasing. Not only that, with the virtualization technology developing, Docker is favored by developers for its lightweight nature and good portability. Docker is playing an increasingly important role in virtualization technology. In order to improve the efficiency of service deployment, more and more people use Docker to deploy services in cloud environments. When deploying services in a cluster, you need to distribute images to nodes. Therefore, how to efficiently distribute Docker images has become a hot topic of research. However, under the development trend of the contemporary Internet, clusters are characterized by large scale and strong heterogeneity. Especially for clusters with relatively obvious heterogeneity, there are many types of nodes, and the performance of the nodes is uneven. So, the inappropriate use of low-performance nodes will seriously restrict the efficiency of Docker image distribution. To this end, in this paper, we propose an efficient Docker image distribution scheme in a heterogeneous cloud environment based on the real characteristics of actual application scenarios, and implement it as a Docker image distribution system, which is called as RainbowD. RainbowD adopts a new high-concurrency mechanism to handle the task of image distribution. First of all, it uses all the nodes in the cluster to release the load of the registry. Secondly, a performance-prioritized method is used to dynamically select nodes with high performance for improving the efficiency of Docker image distribution with limited resources in a heterogeneous cloud environment. Finally, because the nodes are prioritized, each node in the cluster does not need to send messages to all the other nodes, and RainbowD use the SSH protocol to distribute the images for the first time, which solves the disadvantages of message delay, message duplication and redundancy in the P2P network and reduces the waste of network bandwidth. Specially, the drawbacks of the security of the BitTorrent protocol have been criticized by scholars. Considering the security of clusters in the cloud environment, the SSH protocol encryption feature and security verification are used to improve the security of the cloud environment. In this paper, we set up two independent experiments, and the results are as follows: First, RainbowD is compared with other image distribution schemes to verify its efficiency. In the experimental heterogeneous cloud environment, RainbowD has significantly improved the efficiency of Docker image distribution compared to Docket. For example, compared with Docket, RainbowD has improved by more than 30 % and even up to 80 % in the efficiency of Docker image distribution. In 70 % of cases, the efficiency of RainbowD is similar or better than Dragonfly. In the cluster size is 100 nodes, distributing the 200 M images can take up to 150 seconds. Second, RainbowD can effectively resist common network attacks such as DNS spoofing and session hijacking attacks. It can be seen from the experimental results that RainbowD is highly efficient and safe in the task of Docker image distribution.

Keywords Docker technology; heterogeneous cloud environment; images distribution; high concurrency; SSH protocol

1 引 言

随着云计算和大数据时代的到来,虚拟化技术的应用越来越广泛,很多企业都大力发展和应用虚拟机技术;据阿里云公开报道^①,阿里倾尽公司工程师的心血发展虚拟化技术,部署百万甚至千万量级

的业务,为消费者提供安全、稳定、流畅的服务;腾讯发展腾讯云平台^②,为用户提供高效的部署应用、资源调度等一系列完整的功能,为全球的用户提供云计算、大数据、人工智能等服务,这些知名互联网厂商的业务扩展都离不开虚拟化技术。

① <http://dockone.io/article/8190>
② <https://cloud.tencent.com/document>

虚拟化技术由产生至今经过了长期的发展,从最早出现的为解决第三代架构缺点的虚拟机系统到 IBM 重新对虚拟机的定义^[1-3],再到成熟的虚拟机产品,如 VMware、Virtualbox 和 KVM(Kernel-based Virtual Machine)等^[4-6],直到近几年 Docker 容器技术的出现,每一次虚拟化技术的进步都见证了来自全世界学者的努力. Docker 的出现具有划时代的意义,它一经问世便打破了虚拟机的垄断,不仅如此, Docker 以其轻量性、隔离性好、操作简单等独有的特点迅速占据大量的市场,备受用户青睐^[7]. 其实,容器技术在 Docker 出现之前并非空穴来风,从 chroot 系统调用开始就已经出现了雏形,经过数十年的研究,出现了 LXC(Linux Containers)技术,实现了 Linux 容器管理方案, LXC 采用沙箱机制来隔离资源以及进程, Docker 沙箱机制实现的思想也来源于此^[8-9]. 尽管 Docker 的轻量化特性提升了部署服务的效率,但随着集群规模越来越大、集群异构特性越发明显,使用 Docker 技术提升在大规模集群中的服务部署效率出现了瓶颈,如何开发一种高效的 Docker 镜像分发机制成了一个研究的热点问题.

Docker 镜像分发的方法有很多,如使用 Docker 官方提供的镜像站——Docker Hub、国内镜像仓库、本地镜像仓库、借鉴虚拟机镜像分发思想的树状分发机制^[10]、基于 P2P(Peer-to-Peer)思想的镜像分发机制等^[11];然而,在实际的应用环境下,这些方法都有着不同程度的缺陷. 比如,使用 Docker Hub 分发镜像时,镜像分发要经过很长的网络路由路径,对网络状态要求高,而且存在着传统 C/S(Client-Server)架构的弊端^[12-13]. Docker Hub 要响应来自全世界的镜像分发请求,其自身的性能瓶颈限制了请求响应的速度,镜像分发效率低下;使用国内镜像仓库虽然在一定程度上缩短了网络路由路径,但治标不治本,存在着和 Docker Hub 相似的弊端;使用树状网络拓扑结构解决了 C/S 架构的弊端,在 n 个节点的集群中分发镜像时,理想情况下分发时间缩短为原来的 $\log(n)$;其中,基于 P2P 思想的镜像分发机制是一种更优的方法,它可以彻底释放仓库节点的压力,充分发挥集群中所有节点的作用,每一个拥有镜像的节点都有资格为其他节点分发镜像. 然而,现实中集群的异构特性越发明显,从性能上看不同节点之间千差万别,镜像分发的效率不仅和选择分发镜像的方案有关,还受集群中节点性能影响,节点的带宽、磁盘 IO 等参数越高,分发耗时越短,所以不合理地使用低性能节点会限制整个镜像分发任务的效率,这给提升集群服务部署效率带来了新的挑战. 此

外, P2P 网络存在很多的缺陷,首先 P2P 网络存在消息延迟、重复和冗余等问题,通常 P2P 网络消息以一种“洪泛”的现象发送,每一个节点要给其余节点发送消息,有些消息不能够直接到达目标节点,而是通过多个节点过渡转发的形式,尽管通过“随机游走”等算法可以在一定程度上降低消息延迟的影响,但该现象仍然很普遍,而且消息之间存在着大量的重复,极大的浪费了带宽资源^[14-20]. P2P 网络的安全性问题备受学者诟病,传统的 P2P 网络一般基于 BitTorrent 协议实现,存在着很多法律上的争议和安全的隐患, P2P 网络经常受到不怀好意的人攻击,由于 BitTorrent 协议自身缺乏安全性验证的特性,非法攻击很容易奏效;常见的被攻击方式有网络监听、中毒攻击(非法用户提供的内容与描述不符)、拒绝服务攻击、背叛攻击(非法用户只获取资源却不贡献,这也是一种最常见的安全隐患)、在数据中插入病毒、过滤攻击(一些网络运营商试图禁止来自 P2P 网络上的数据)等^[21-25];在云环境下集群中的集群节点或者镜像数据如果被攻击,会造成不可估量的损失.

针对以上的问题,本文设计了异构云集群环境下高效的镜像分发系统——RainbowD. RainbowD 通过使用一种高并发机制处理镜像分发任务,发挥集群中所有节点的作用来分担镜像仓库的任务,解决 C/S 架构中仓库节点响应请求压力过大的问题;考虑到异构云环境的真实特性,异构云环境中节点的性能参差不齐, RainbowD 通过动态的选择高性能的优质节点,避免了低性能节点使用过多而降低分发效率,在有限的资源下提升系统吞吐率; RainbowD 基于对节点优先级排序的结果并首次使用 SSH 协议进行镜像分发,避免了使用 BitTorrent 协议存在的消息延迟、消息重复和冗余现象;此外, SSH 协议在传输数据时,会进行安全性的身份认证和数据加密,保证集群能够抵御常见的网络攻击,提升云环境和数据的安全性.

本文之后将在第 2 节介绍相关工作的部分,比如 Docker 介绍、Docker 镜像优化、常见的 Docker 镜像获取方式介绍和 SSH 协议;第 3 节介绍 RainbowD 功能和核心结构;第 4 节是实验结果与分析;第 5 节进行总结.

2 相关工作

2.1 Docker 介绍

Docker 是一个开源的应用容器引擎,是实现虚

拟化的一个重要方式. Docker 采用 client-server 架构, server 端成为 Docker 的守护进程, 当守护进程启动后就会一直运行在宿主机的后端, 时刻准备着接收 Docker 客户端的消息; Docker 的客户端主要和用户交互, 然后将从用户接收到的命令传递给 Docker 守护进程, 最终守护进程将命令执行的结果发送给 Docker 客户端, 由 Docker 客户端展示给用户^[26-27]. 架构如图 1 所示.

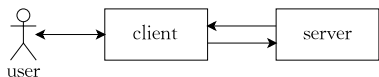


图 1 Docker C/S 架构

Docker 容器完全采用沙箱机制, 不同的容器之间互不依赖、相互隔离; 沙箱机制是一种主要应用于计算机安全方面的虚拟化技术, 它提供的环境相对于不同的程序之间都是相互独立的, 其原理主要是运用重定向技术, 把对文件的操作定向到自身独立的环境中, Docker 容器之间互相隔离的特性也是基于这个思想实现的^[8-9, 28]. Docker 和传统的虚拟机相比有很多优势, 在使用传统的虚拟机安装应用或者部署服务时, 需要先在宿主机上虚拟出一套硬件设备, 接着虚拟出一个操作系统, 最后才能安装应用^[29]; 而 Docker 只需要在宿主机上创建 Docker 引擎就可以利用宿主机的操作系统调用硬件资源, Docker 引擎可以直接与宿主机的操作系统进行通信, 为容器分配资源, 由于容器没有复杂的虚拟化硬件设备, 所以 Docker 容器相对虚拟机来说更加节省宿主机的磁盘空间和其他系统资源, 由于 Docker 比较轻量化, 启动 Docker 的时间只需要 1 秒钟, 这些特性保障了服务能够高效的部署^[30-32]. Docker 优秀的隔离性和轻量性成了 Docker 技术异常火热的主要原因, 从 2013 年以来, 不管是 Github 上代码的活跃度还是 Docker Hub 上镜像的种类都能看的出来. 据不完全统计, 在 2014 年, Docker Hub 上已经有超过 14 000 个 Docker 化应用存储在共有仓库中; 2015 年, Docker 化的应用已经超过 150 000 个, 仅一年的增长就有十倍之多, 除了公有仓库中应用数量的增加, 很多用户或者企业建立了满足自身需求的私有仓库, 由于受众广泛, 不管是企业还是用户都会首选 Docker 部署服务^①.

Docker 主要包含以下五个部分, 分别是 Docker 客户端、Docker daemon、Docker 镜像、Docker 容器、Docker 镜像仓库. Docker 客户端是用户和 Docker daemon 交流的工具, 用户可以远程使用 Restful

API(Application Programming Interface)或者命令行工具, 客户端封装了一些常用的命令, 用户根据自身不同的需求选择合适的命令. Docker daemon 在宿主机的后台运行, 用来处理客户端的请求, 常见的功能有运行或者停止容器、从镜像仓库获取镜像或者将镜像存储到镜像仓库等. Docker 镜像是用来新建容器的完整功能的静态文件, 是容器的基础, Docker 镜像按层组织, 不同的镜像层通常代表不同的功能, 镜像构建时, 会一层层构建, 前一层是后一层的基础, 镜像层可读不可写, 即构建完毕后不会再次改变; Docker 容器是镜像的动态运行过程, 多个容器在同一个宿主机上运行时相互隔离; Docker 镜像仓库是一个镜像的资源库, 存储着镜像需要的各种各样的资源, Docker Hub 是 Docker 官方提供的镜像仓库^[33-37].

2.2 Docker 镜像优化

在镜像分发中, 为了提升镜像分发的效率, 在满足需求功能的情况下尽量构建更小的镜像是可行的手段之一. 常见的方式有以下几种:

(1) 选择更小的基础镜像. 在构建功能复杂的 Docker 镜像, 需要在基础镜像添加相应的功能, 在满足需求的情况下, 选择更小的基础镜像^[38].

(2) 将 Dockerfile^[39] 指令串联. Dockerfile 是用来实现构建、定制镜像的脚本, 它是一个文本文件, 每一条指令可以构建一层, 将多条指令串联在一起, 减少构建镜像的层数, 镜像层越少, 层模板文件和配置文件等也就越少, 可以有效的减小镜像的大小.

(3) 使用镜像删减策略减小镜像大小. 周毅等人提出了一种使用概率模型的方法来预测要删除的镜像内容, 这个方法的关键是对于导出镜像文件信息的收集和分析; 针对一些常见的镜像, 为了确保功能的完备, 构建镜像的时候会将所有可能有联系但是几乎不可能用到的文件添加进去, 在做镜像文件导出时, 统计不同目录被访问到的概率, 给定一个阈值, 如果概率大于阈值, 则全部导出, 否则, 则有选择地导出该目录下的子文件^[40].

2.3 Docker 镜像分发方式

在 Docker 镜像优化的基础上, 可以再配合使用 Docker 镜像分发机制完成镜像分发的任务. Docker 镜像分发的机制有很多, 常见的方法是从镜像仓库获取; Docker 官方提供了一个镜像站——Docker Hub. Docker Hub 为用户提供了获取镜像和共享镜

① <http://www.lupaworld.com/article-254520-1.html>

像的服务的工具,它支持获取官方存储的公有镜像和用户构建存储室存储镜像两种模式。公有镜像是 Docker 提供的一些常用的、高质量的镜像,用户可以直接使用 pull 命令拉取;对于存储室,一般用户构建的存储室有两种状态:公有态和私有态,构建者对存储室有管理权限,其他用户可以直接获取公有态存储室的镜像,而对于私有态的存储室,其他用户需要获得管理者的授权后才可以获得存储室里的镜像,这是用户共享镜像的一种方式^[35-36,41]。从 Docker Hub 上获取镜像是一种最普遍和通用的方式,该方式的架构如图 2 所示。

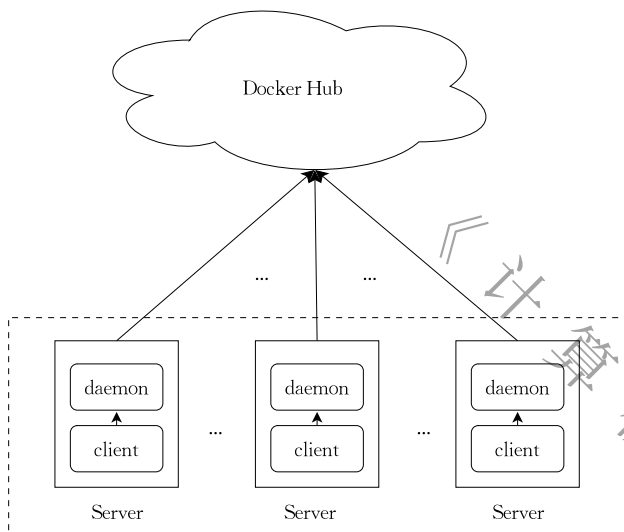


图 2 从 Docker Hub 上获取镜像

在国内使用 Docker Hub 获取镜像要经过漫长的网路路由路径,为了缩短这个路径长度,很多企业建立了国内镜像站点,如 DaoCloud 镜像市场^①、阿里云镜像仓库^②、网易云^③、微镜像——希云 cSphere^④、镜像广场——时速云^⑤、灵雀云^⑥等,这些产品的出现在一定程度上提升了利用 Docker 部署服务的效率。此外,很多企业或者用户开始在集群中搭建本地私有仓库,所谓本地仓库就是选择集群内的一个或多个节点存储镜像资源,供集群内部使用,通常不对外开放。

在 Docker 出现之前,虚拟机同样也有镜像分发的挑战,学者们针对虚拟机镜像分发做了很多研究,后来将这些研究成果应用在 Docker 镜像分发任务中。2009 年, Schmidt 等人为了降低镜像分发时网络阻塞和镜像仓库的压力,提出使用一种二叉树的结构分发镜像,如果分发一个镜像的时间为 t ,在理想状态下, n 个节点的镜像分发时间缩短为 $t \times \log(n)$ ^[10]; 2010 年, Schmidt 等人对自己的成果做了优化,提出

使用 P2P(Peer-to-Peer)网络分发镜像机制,改方法通过 BitTorrent 协议分发镜像,当服务集群内的大量节点同时需要部署特定的服务或者安装应用时,镜像分发的任务不再单独由仓库节点完成,而是由仓库节点和已经传输完成的节点一起完成,集群中拥有了特定的镜像资源的节点可以加入镜像分发的任务,为其他未拥有资源的节点执行镜像分发操作,仓库节点的压力被转移,从根源上解决了传统 C/S 架构的瓶颈^[11]。借鉴 P2P 思想的企业级镜像分发系统接二连三的出现,如 VMware 的 Harbor^⑦、Quay^⑧、Docket^⑨、美团的 HULK^⑩、腾讯的 GaiaStack^⑪、阿里的 Dragonfly^⑫、华为云容器镜像服务 SWR^⑬等;此外,在学术界,2016 年, Liang 等人为数据中心提出了一种高效混合的 Docker 镜像分发系统,将 P2P 技术和使用 Docker Hub 获取镜像的方式相结合,根据需求有选择地使用 P2P 网络或 Docker Hub 获取镜像或镜像层^[42]; 2018 年, Wang 等人提出了一种高效的镜像分发系统——FID,基于 BitTorrent 协议分层分发 Docker 镜像^[43]; Zheng 等人提出了一个镜像共享的系统——Wharf,它使用分布式文件系统在集群中共享镜像,减少镜像通过网络传输的数量^[44]。以下内容分别是对 Docket、Harbor、Quay、Dragonfly、FID 和 Wharf 进行了详细的介绍。

Docket 提供了一个自定义的 P2P 镜像仓库,该仓库可以通过 BitTorrent 分发镜像。在大规模集群中部署镜像时,这是一个相对高效和轻量级的解决方案。Docket 的设计初衷是为了缓解镜像仓库节点的压力,当所有节点都从一个镜像仓库节点获取镜像时,镜像仓库的压力和集群的规模呈线性关系,镜像仓库的性能的瓶颈限制了镜像仓库的效率; Docket 采用了一种 P2P 的思想,用集群中其他节点来分担镜像仓库的压力。Docket 为仓库中的镜像生成 torrent 文件,集群中所有的节点都可以通过 torrent 文件下载镜像,这些节点拥有了镜像资源后可以供别的节点下载,最后将所有节点的镜像加载

① DaoCloud. <https://www.daocloud.io>

② aliyun. <https://www.aliyun.com>

③ 163yun. <https://www.163yun.co>

④ cSphere. <http://csphere.com>

⑤ tenxcloud. <http://docs.tenxcloud.com/guide/registr>

⑥ alauda. <http://www.alauda.cn>

⑦ Harbor VMWare. <https://github.com/vmware/harbor>

⑧ quayctl CoreOS. <https://github.com/coreos/quayctl>

⑨ docket. <https://github.com/netvarun/docket>

⑩ hulk. <https://tech.meituan.com/2017/05/12/hulk-scheduler-introduction.html>

⑪ Tencent GaiaStack. <http://dockone.io/article/1555>

⑫ Dragonfly. <https://github.com/dragonflyoss/Dragonfly>

⑬ SWR. <https://support.huaweicloud.com/swr/index.html>

到 Docker 引擎。

Harbor 是 VMware 公司研发的一个镜像仓库服务器,用来存储和分发 Docker 镜像;Harbor 基于传统的 C/S 架构,所有的用户(clients)都可以通过这个仓库拉取镜像,后来,VMware 研发了一个 P2P 版本的 Harbor 原型系统——DID. DID 使用 BitTorrent 协议实现镜像分发,初始状态时,控制器节点首先从 Docker 镜像仓库获取一个镜像,以此为基础为整个集群做镜像分发,分发时以一整个镜像为单位,当一个节点拥有了镜像后,就可以作为“seeder”节点为其他节点分发镜像(“seeder”节点表示该节点已经拥有了对应的镜像文件,其他节点可以从该节点获取资源).为了验证高效性,VMware 做了一个实验,结果显示:相对传统的直接从镜像仓库拉取镜像的方式,在小规模的集群下拉取比较小的镜像文件时,传统的方式时间更短,但是随着集群规模和镜像大小的增大,DID 的效率会渐渐超过传统的拉取镜像的策略,在大规模集群中,使用 DID 的策略相对传统的方式在时间上缩短了 80%.在集群规模逐渐增大的过程中,存在一个两种策略时间上相等的临界点.在 100 个节点的集群规模中,DID 效率更高,在当今大数据时代,DID 相对传统的方式有着更好的应用前景^①.

Quay^② 是一个提供镜像仓库服务的商用产品,是最先进的可视化仓库,用户可以通过图像化界面看到关于镜像最大化的信息,比如镜像历史、标签等,用户可以轻松的在云端编辑 Dockerfile;Quay 的 P2P 实现通过 BitTorrent 获取镜像文件,支持 rkt 格式的镜像^[45],工作模式如下:当很多获取镜像的请求来到时,一个节点获取完整的镜像文件后,接着就可以作为“seeder”节点,充分利用了除仓库节点之外集群中其他节点的资源,虽然没有达到 P2P 的最理想状态,但是也大幅提升了时间效率,和 DID 类似.

Dragonfly——“蜻蜓”是阿里巴巴自主研发的开源系统,起初用来做文件分发,是阿里基础运维平台的重要组成部分,后来做了升级,支持 Docker 容器技术,可以做镜像的分发.每个月 Dragonfly 可以为阿里巴巴分发 3.4PB 的数据,可靠性达到 99.999% 以上.在镜像分发中,Dragonfly 采用了 P2P 的思想,从根本上解放了镜像仓库的压力,克服了传统 Docker 镜像分发的瓶颈. Dragonfly 的镜像分发架构如图 3 所示^③.

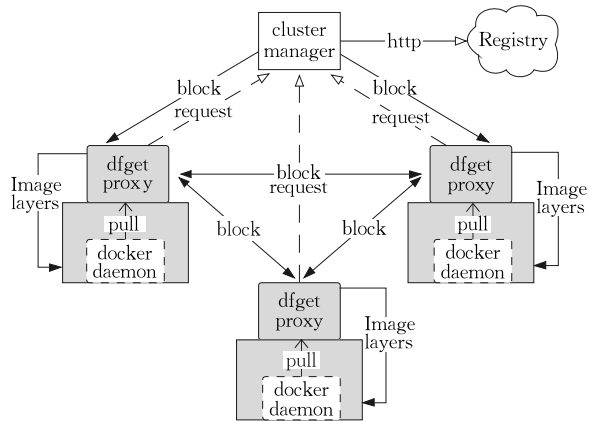


图 3 Dragonfly 架构图

在镜像分发中,Registry——镜像仓库相当于文件服务器;dfget 是 P2P 的客户端,称为“peer”,主要用来下载或共享文件块,dfget proxy 被称为 dfdaemon,是一个守护程序,用来监听来自用户的 docker pull 或 docker push 的 http 请求,监控到这个请求之后再决定利用 dfget 是做拉取镜像的操作或推送镜像的操作,在这里重点使用 docker pull 操作;cluster manager 称为超级节点,用来管理每一个客户端节点,并决定哪一个客户端的文件共享到另一个客户端.当集群内的大量“peer”节点同时发起获取镜像的请求,通过 cluster manager 节点进行处理,若集群内别的节点暂时都还没有镜像资源,响应快的节点会先从 Registry 获取镜像文件,当集群内已经有一个或一些节点有了请求的镜像文件,就可以通过 P2P 网络分发给剩余的节点,直至发起请求的“peers”都拥有了该镜像资源.

FID 提出了一种用 P2P 网络加速镜像分发的思想,构建一个 P2P 镜像仓库.仓库中存放的是能够组成镜像的静态 Blob 文件,镜像仓库为每一个静态 Blob 制作 Torrent 文件,客户端通过 BT Tracker 服务端获得对应 Blob 的存储地址,做分发操作,不同的镜像有些 Blob 文件是可以共享的,在一个节点上公用的层文件只传输一次,在大规模的镜像传输中总量上可以减少一些.FID 代理有两种工作模式,分别为 load 模式和 proxy 模式;load 模式是通过“load”命令将镜像文件加载到 Docker daemon,proxy 模式中 FID 代理以 Docker 引擎的 http 代理服务运行,捕获下载 Blob 的请求,通过 P2P 方式

① 用 P2P 方法快速分发 Docker 镜像. <http://dockone.io/article/1162>

② Quay. <https://docs.quay.io/>

③ Dragonfly. <https://d7y.io/en-us/docs>

下载对应的 Blob^[43].

Wharf 采用了一种分布式文件系统(Distribute File System)的方式来共享镜像. 分布式文件系统是指文件系统所管理的资源不一定直接存储在本地节点上,而是存在多个节点上,这些节点之间通过网络相连,用户不用关心文件存储在哪个节点上,在使用时可以通过网络从文件所在的节点获取,经常用来解决在大文件存储中单个节点存储空间不够的问题,也常应用在多节点共享使用文件的场景;在镜像分发领域,可以将镜像资源通过分布式文件系统存储,以一种资源共享的方式允许多个节点同时使用镜像资源. 在 Docker 中,镜像由多个层文件(layer)组成,根据这些层文件(layer)的作用将其分类为两种状态:局部状态和全局状态,局部状态的 layer 代表某些需要部署的服务独有的层,全局状态的 layer 表示节点之间可以共享. Wharf 在集群中选择多个节点构建分布式文件系统,将全局状态的 layer 存储到分布式文件系统中,最后根据用户的需求将不同的 layer 组装成完整的镜像. Wharf 利用分布式文件系统使多个节点共享镜像的全局层,提高镜像分发的效率^[44].

表 1 是常见镜像分发方案的总结.

表 1 镜像分发方式汇总

镜像分发方式	实现方式	是否开源
从 Docker Hub 获取镜像	Client/Server	—
DID	BitTorrent	是
Quay	BitTorrent	是
Docket	BitTorrent	是
HULK	BitTorrent	否
FID	BitTorrent	否
Dragonfly	BitTorrent	是
Wharf	DFS	否

2.4 SSH 协议

互联网现在成了最普遍的沟通工具,客户端与客户端、客服端与服务器之间的操作很频繁,这些操作都可以通过网络完成,如远程连接操作、字符传输操作、文件流传输操作等;但是刚开始时网络的传输没有提供安全机制,所有的数据在传输过程中都是明文发送的,黑客就有了可乘之机,企业和个人深受其害,网络攻击的常见方式有网络监听、连接/会话劫持、路由欺骗、DNS 欺骗、拒绝服务型攻击等;当企业的服务器集群上的数据遭受到网络的入侵,防火墙和一些权限控制能起到一定程度上的安全防护效果,但是并不能从根本上解决问题. SSH 是一种

比较可靠、能够保证远程登录会话和一些别的网络服务安全性的协议,它工作在传输层的顶端,使用二进制数据包的形式,传输二进制的数​​据流,这些数据包在在加密传输之前可以经过压缩算法进行数据压缩,比如 gzip 等,提升了数据传输的效率^[46]. SSH 协议由三部分组成,分别是传输层协议——保证了服务器的正确认证、保密性、完整性和提供了压缩功能,用户认证协议——用于向服务器提供客户端用户鉴别功能和连接协议——提供交互式会话登录、远程执行命令等,不同的协议保证了 SSH 不同方面的安全性;SSH 的架构是 C/S 模式,即客户端和服务端,客户端一般包含了 ssh 程序和其他的一些应用程序,如常用的 scp 等,服务端是一个守护进程,随时准备响应客户端的请求,每一次的验证和加密都保证了 SSH 协议安全性的特点^[47-51].

使用 SSH 协议从建立连接到传输文件主要分为以下几个步骤:

- (1) 客户端向服务端发出建立连接的请求.
- (2) 服务端处理该请求,并将生成的公开的 RSA (非对称加密)的主机密钥和公有的 RSA 服务器密钥发送给客户端;客户端接收到主机密钥后会和自身数据库的密钥比较.
- (3) 客户端生成一个 256 位的随机码,客户端和服务端进行算法选择,选择一个双方都支持的算法,客户端将这个随机码加上接收到的主机密钥和服务器密钥进行加密,并将加密之后的密文发送给服务端.
- (4) 服务端将接收到的密文解密并恢复会话密钥. 服务端会给客户端发送一个验证,目的是为了证明服务端有能力正确的解密,拥有了合适的私钥.
- (5) 认证阶段. 客户端向服务端发送认证请求,请求中通常包含用户名、认证方法、密码和密钥;服务端会处理认证请求并对客户端进行认证,如果认证失败,则向客户端发送认证失败信息;客户端会继续使用支持的一种认证方法继续认证,直到认证次数达到上限或者认证成功为止.
- (6) 认证成功后进行会话请求. 此时客户端向服务端发起会话请求,服务器处理该请求,双方进入会话交互阶段.

3 RainbowD 功能和核心结构

RainbowD 是一个在云环境下批量管理节点和

利用 Docker 部署应用或服务的系统,它是对 Docker 在应用方面的开发,其功能包含 Docker 的批量安装和卸载、镜像的获取和删除、容器创建和销毁、Dockerfile 文件的生成和传输等;Dockerfile 是一个用来定制镜像的有效手段,在利用 Dockerfile 定制镜像的过程中,通常定制的镜像是基于一些基础镜像做的功能的拓展,在 RainbowD 中为了重复使用这些基础镜像或者镜像层文件以减少镜像分发总量,用户可以先将基础镜像分发到各个节点上,然后批量分发和执行 Dockerfile 文件,将更多的选择权交给使用者,使用者可以根据实际情况灵活使用; RainbowD 在做分发时可以支持压缩操作,这可以减少传输数据的总量,如分发的内容原来大小为 1M,压缩率是 40%,则真实传输的数据量为 0.4 M. RainbowD 的主要功能是在一个云环境集群中分发镜像,旨在提升镜像分发的效率. RainbowD 将一个镜像分发的任务尽可能的并行化,并充分发挥集群中所有节点的网络传输作用,解放仓库节点的网络传输压力;在异构的环境中, RainbowD 会对集群中的节点进行性能优先级排序,使用一种“贪心”的思想,每次根据当前时刻节点的占用情况做出最佳选择,实时的记录和更新集群中节点的状态,避免了 P2P 网络中消息延迟、消息重复等弊端,动态地分配镜像传输的任务,提升镜像分发的效率; RainbowD 首次使用 SSH 协议,相对 BitTorrent 协议不存在冗余现象并且减少制作 torrent 文件的过程. 此外, SSH 协议具有多项安全性验证,提升了云环境的安全性. RainbowD 分发镜像的架构如图 4 所示.

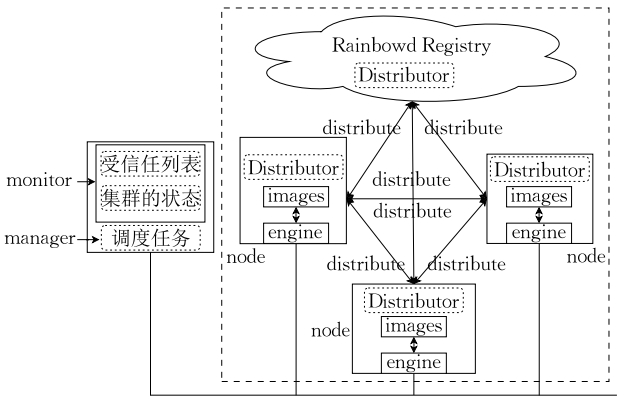


图 4 RainbowD 分发架构

3.1 RainbowD Registry

RainbowD Registry 是镜像仓库,仓库中存放完整的镜像. 在 RainbowD 中,仓库的信息用一个基于 Key/Value 的表记录,Key 对应的是镜像 ID,不同的镜像通过镜像 ID 索引,镜像 ID 由镜像名和版

本组成,Value 对应的是在文件系统中镜像存放的地址;根据集群中业务或者功能的需要,镜像仓库可以随时地增加或者删除镜像.

3.2 Monitor

RainbowD 的 Monitor 模块是一个集群监视器,用来监控集群中节点状态改变的信息. 在初始状态下,Monitor 存放了一个信任列表,列表中存放了集群中的节点信息,这些信息包括 IP 或者域名、用户名、密码等. 在初始状态下,集群中的节点刚被初始化,此时节点是安全的,在传输过程中,如果节点的信息被攻击或者被篡改,比如 IP 欺骗攻击,在每次进行数据传输前会进行身份验证,如果和信任列表中的记录不一致,则将该节点删除并提示用户开启安全监测;信任列表提升了整个集群可扩展性,修改信任列表的记录可以动态的扩展集群的规模. 信任列表的格式如表 2 所示.

表 2 信任列表

公有 IP	用户名	密码
192. 168. 1. 110	username1	password1
...	...	...
192. 168. 1. 180	username1	password1

Monitor 模块还记录了集群内节点(包括仓库节点)的状态改变的信息,这些信息包括节点拥有镜像资源的种类、是否在执行分发镜像的子任务而被占用、节点的性能等. 针对特定的镜像,如果节点拥有该镜像,则称为“seeder”,seeder 节点的数量是动态变化的,在分发过程中,seeder 节点可以为镜像仓库分担传输压力. seeder 节点的性能的判断是根据上一次该节点完成一次镜像分发时长来决定,时间越长,性能越低,反之性能则越高.

3.3 Manager

Manager 模块具有管理、调度任务等作用,根据 Monitor 模块记录的信息,Manager 模块做镜像分发的任务调度. 在某一个时刻,如果多个节点同时有获取某一特定镜像的请求,这些节点称为 demander, Manager 模块首先通过 Monitor 模块得到集群所有节点的状态信息;接下来,准备发起多个线程为 demander 节点并行的分发镜像,每一个 demander 节点获取镜像的过程互不影响;分发之前,将当前时刻未被占用的 seeder 节点根据性能排序,seeder 节点排序的准则是执行 Distributor 操作完成的时间,时间越短,则认为性能优先级越高. 节点执行分发操作时间示例如图 5 所示.

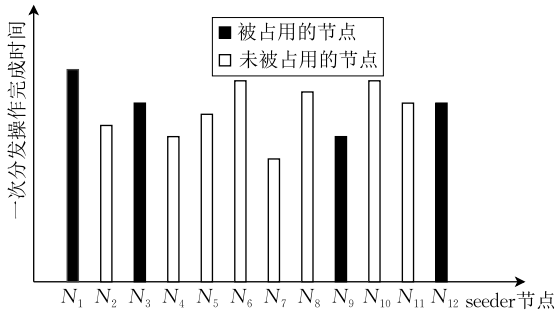


图5 节点执行一次分发操作的时间

在执行分发操作之前, Manager 模块会先根据分发操作完成的时间选择出性能优先级最高的节点 N_7 , 得到当前时刻的一个局部最优解, demanders 各个节点同时竞争 N_7 , 其中一个 demander 节点竞争成功后, 两个节点建立连接, 执行镜像分发操作; 每一个 demander 节点都会根据对 seeder 节点的性能从高到低依次竞争 seeder 节点, 如果 demander 节点的个数多于可用的 seeder 节点个数, 这时所有的 seeder 节点全部会被占用, 处于加锁状态, 没有竞争成功的 demander 节点将处于等待状态, 直到有 $\{seeder, demander\}$ 的 Distributor 操作执行结束, 此时对应的 seeder 节点的锁会被释放, seeder 节点会转变为未被占用的状态, 而且由于 demander 已经拥有镜像, 所以将其更改为 seeder, 分发成功的镜像存放在节点的文件系统中, 该节点在准备为处于等待状态的 demander 分发镜像的同时并开始动态地将镜像加载到 Docker 引擎中, 镜像加载的过程是本地完成的, 不占用网络的带宽; 接下来, 剩余的 demander 节点继续竞争 seeder 节点, 直到 demander 集合变为空, 所有的节点都有了该镜像. 整个过程如图 6 所示.

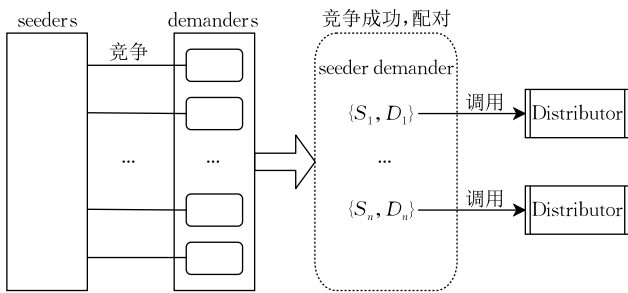


图6 并发分发镜像的过程

RainbowD 镜像分发算法如下:

算法 1. RainbowD 分发镜像.

1. LOOP
2. $requestNode = getRequestNodeFromCluster()$
3. $seederNodeQueue = getSeederFromCluster()$

4. $sortByPerformance(seederNodeQueue)$
5. WHILE $requestNode$ IS NOT EMPTY;
6. $seederId = seederNodeQueue.take()$
7. $demandId = requestNode.take()$
8. LOCK($seederId, demandId$)
9. NEW THREAD($send \langle seederId, demandId \rangle$ to Distributor of $seederId$, execute Distributor until distributing image succeed)
10. UNLOCK ($seeder, demand$) and change $demand$ to $seeder$
11. $seeder$: load image to docker engine
12. $seederNodeQueue = getSeederFromCluster()$
13. $sortByPerformance(seederNodeQueue)$
- END LOOP

在 RainbowD 中, 每一次执行镜像分发操作的 seeder 和 demander 都是经过合理地调度得到的当前时间点最合适的匹配, 在消息传递时只需要在对应的 seeder 和 demander 之间进行, 解决了 P2P 网络中消息延迟和消息重复的问题; 所谓消息延迟指的是某个节点在进行消息转发时, 不直接转发给集群中所有的节点, 而是转发给部分节点, 这些接收到消息的节点再次进行转发, 产生了消息需要进行多次转发之后所有节点才能完全收到的延迟现象; 所谓消息重复指的是每一个节点都会向集群中其他节点转发消息, 所以一个节点可能会收到多个相同的消息, 消息重复的比例很大, 容易造成网络资源的浪费.

RainbowD 在做镜像分发时根据节点性能的性能排序采用的是一种“贪心”的思想, 每次都选择当前时间点最合适的节点, 是一种基于 seeder 节点性能高低而进行的短作业优先调度, 即让分发最快的节点先完成, 接着这些节点被释放继续完成镜像分发的任务; 由于每次都做当前时刻最优的选择, 没有以全局的视角关注整个镜像分发任务, 所以 RainbowD 的调度结果不一定总能得到一个理想状态下的最优调度顺序. 在真实的实验环境中, 镜像分发的速度受很多因素的影响, 如机器的性能、不同时刻的网络波动、CPU 占有率、新加入的 demander 节点的 I/O 速度、甚至是镜像分发任务的不确定性 (所有节点发起请求的时间不可能完全同步) 等, 这些客观因素都导致了镜像分发任务的不确定性. 根据以上分析, 在进行重复性实验时, 实验结果会在一定范围内波动.

3.4 Distributor

Distributor 是一个镜像分发的组件. 当 demander

节点竞争成功 seeder 节点后,manager 将镜像 ID、镜像存放路径、demonder 节点的用户名、密码等信息传递给 seeder 节点上的 Distributor,Distributor 模块使用 SSH 协议执行分发操作,直到 Distributor 执行结束 manager 更改节点的状态信息,由于 SSH 协议的命令是交互式的,本文对 SSH 协议的传输操作做了非交互式封装. RainbowD 采用增量传输的方式实现断点续传,当网络发生暂时性阻塞时,已经传输成功的部分不再重复传输,当网络连通后重新建立连接继续增量的传输未成功的部分,避免在不稳定的网络中出现重传的现象.

3.5 容错处理和一致性分析

在系统运行的过程中,可能会出现一些无法预料的异常情况发生,针对可能出现的错误,RainbowD 做了一些异常处理. 针对 monitor、manager 原节点可能出现宕机等异常的情况,采用一个部署相同功能但不启动相关服务的节点作为备份节点,原节点和备份节点互相监控对方的状态,如果备份节点监测到原节点宕机,则备份节点启动,并发出警告,提醒相关人员修复原节点,如果原节点监测到备份节点发生宕机等异常情况,发出警告并提醒相关人员修复该节点,用互相监控的方式处理宕机等异常情况;仓库节点 RainbowD Registry 同样如此,采用备份仓库节点和原仓库节点形成互相监控. 针对集群中的节点,在镜像分发的任务中,如果是 seeder 节点宕机或者其他原因不能正常工作,则将该节点做标注并将其作为 seeder 节点的属性删除;如果是 demonder 节点出现了异常,则将和其配对的 seeder 节点从占用状态释放,并将异常的节点做标注. 为了保证在镜像分发过程中的一致性和完整性,在镜像分发的子任务中为 seeder 节点和 demonder 节点加锁,以防止同一个 seeder 节点同时为多个 demonder 节点分发,降低镜像分发的效率,并且防止镜像重复分发,造成资源和时间的浪费.

为了验证容错机制,在系统正式运行前做宕机模拟处理,主动地将相关节点关闭,检验宕机容错机制是否响应. 通过观测节点的状态参数验证分发的一致性,节点的状态如果改变,一定是从请求资源的状态转为获得资源的状态;如果产生了从获得资源状态到获得资源状态的重写、从获得资源状态变为请求资源状态或者请求资源状态一直没有改变,则系统产生了一致性方面错误,出现了重复分发或者镜像分发失败的现象.

4 实验结果与分析

4.1 实验环境

集群中的节点可以分为三类,分别是仓库节点、物理节点和虚拟节点.

仓库节点有两个,其中一个是另一个的备份,当发生宕机状况时会被启用. 仓库节点配置如表 3 所示.

表 3 仓库节点配置

节点	CPU/ 个	CPU 核数/ (核/个)	CPU 频率/ GHz	带宽/ Gbps	内存/ GB	硬盘/ TB
仓库节点	4	8	2.0	1	500	4.5

物理节点有 25 个,简要配置信息如表 4 所示.

表 4 物理节点配置

节点	CPU/ 个	CPU 核数/ (核/个)	CPU 频率/ GHz	带宽/ Gbps	内存/ GB	硬盘
N1	2	6	2.0	1	3	75 G
N2	2	6	2.0	1	7	800 G
N3~N12	2	6	2.0	1	31	2 T~ 10 T 之间
N13~N17	2	6	2.0	1	46	3 T~ 8.5 T 之间
N18~N25	2	6	2.0	1	15	2 T

虚拟节点有 75 个,简要配置如表 5 所示.

表 5 虚拟节点配置

节点	CPU/ 个	CPU 核数/ (核/个)	CPU 频率/ GHz	带宽/ Mbps	内存/ GB	硬盘/ G
N26~N100	1	1	2.0	128~512 之间	1	10

4.2 对比实验设计和结果分析

4.2.1 对比实验设计

针对镜像分发的任务,本文选择了以下 2 个开源的镜像分发的方案和 RainbowD 做对比,验证 RainbowD 在镜像分发上的高效性.

方案 1. 利用 Docket 分发镜像. 当多个节点需要同时分发镜像,这些节点并发的获取 torrent 文件,并下载需求的镜像;镜像分发完成的时间为所有节点并发下载镜像的时间和将镜像加载到 Docker 引擎的时间.

$$T_{\text{docket}} = T_{\text{download}} + T_{\text{load}}$$

(1)

方案 2. 利用 Dragonfly 分发镜像. Dragonfly 使用 P2P 网络分发镜像,镜像分发完成的时间为请求节点从 docker daemon 发出请求开始到所有请求

的节点拥有完整镜像的时间。

$$T_{\text{dragonfly}} = T(n_1, n_2, \dots, n_m) \tag{2}$$

方案 3. 利用 RainbowD 分发镜像. 镜像分发完成的时间为从节点发起请求开始到节点状态为拥有镜像的时间和。

$$T_{\text{RainbowD}} = T(n_1, n_2, \dots, n_m) \tag{3}$$

以下的实验设计是为了比较以上三种方案在不同情况下的效率; 本文设置了两个变量, 分别是集群的规模和分发镜像的大小, 为了减小除了自变量之外其他不确定因素对结果的影响, 将 Docket、Dragonfly 和 RainbowD 都部署在同一个异构云集群中. 根据分发镜像大小的不同, 本文选择了以下多种镜像, 镜像大小保留一位小数, 如表 6 所示。

表 6 镜像种类和大小	
镜像名称	镜像大小/MB
registry	33.3
memcached	62.1
ubuntu	86.2
debian	100.6
rabbitmq	124.8
neo4j	201.0
postgres	310.8
mongo	382.7
wordpress	408.2
tomcat	462.7
java	643.1
jenkins	695.7
golang	774.3
sonarqube	830.7
python	922.5

表 6 中的镜像是受用户欢迎度比较高的镜像, 颇受用户青睐, 具有很好的代表性^①; 在本文中 Docker 镜像的大小是变量之一, 表 6 中的镜像大小呈递增趋势, 可以有效的验证镜像大小和分发效率的关系。

4. 2. 2 实验结果和分析

影响镜像分发的时间的两个重要因素分别是集群的规模和镜像的大小, 根据这两个变量, 本文将实验分为两组, 每组包含三个实验, 以此验证在不同的条件下 RainbowD 的性能. 以下是实验的结果, 进行了多次重复实验, 取均值后再取整。

实验 1. 1. 选择 registry 容器镜像, 集群的规模从 10 个节点变化到 100 个节点, 统计不同的镜像分发方案的时间. 统计结果如图 7 所示。

实验 1. 2. 选择 debian 容器镜像, 集群的规模从 10 个节点变化到 100 个节点, 统计不同的镜像分发方案的时间. 统计结果如图 8 所示。

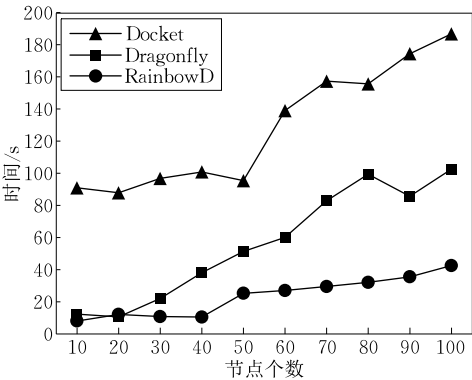


图 7 分发 registry 镜像的时间折线图

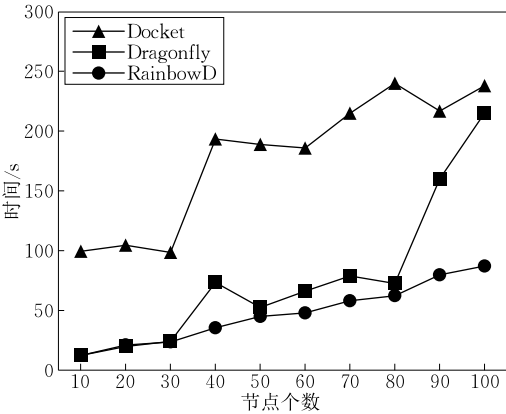


图 8 分发 debian 镜像的时间折线图

实验 1. 3. 选择 sonarqube 容器镜像, 集群的规模从 10 个节点变化到 100 个节点, 统计不同的镜像分发方案的时间. 统计结果如图 9 所示。

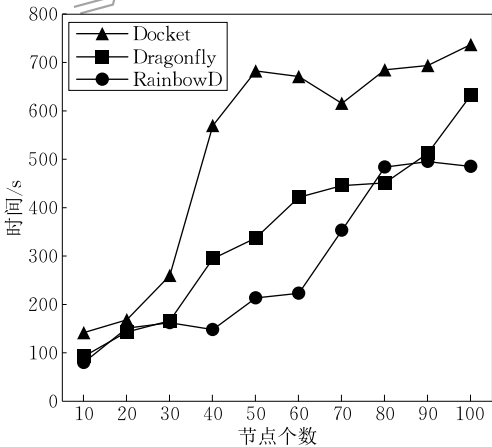


图 9 分发 sonarqube 镜像的时间折线图

从第一组的实验结果折线图 7~图 9 可以看出, 在分发相同镜像的前提下, 随着集群规模的增大, 三种方案的整体耗时都有一个上升的趋势. Docket 和本文提出的 RainbowD 相比, 整体性能劣

① <https://hub.docker.com/search?type=image>

于 RainbowD, Docket 使用 BitTorrent 协议分发镜像,首先需要制作 torrent 文件,而且随着节点规模的增大,消息延迟和消息重复的现象越发明显,这些都降低了镜像分发的效率;此外, Docket 将镜像加载到 Docker 引擎的过程分离,所以当所有节点的文件系统都拥有了镜像时,再进行动态加载镜像.从折线图 8、图 9 中可以看到,在集群规模从 30 个节点增加到 40 个节点时, Docket 和 Dragonfly 的效率有一个骤变的趋势,从表 4、表 5 可以知道编号 N26 以后的节点都是虚拟节点,性能相对较低,当集群从 30 个节点继续增加时,加入了大量的低性能的节点,这些低性能节点本身的分发时间对整体耗时影响较大,所以完成镜像分发任务的耗时会突然变长;但使用 RainbowD 方案很少出现这个现象,这是因为 RainbowD 在分发的过程中,开始每一个镜像传输子任务之前会优先选择可用的性能优先级最高的 seeder 节点,尽量降低低性能节点的影响, Docket 和 Dragonfly 会随机的选择可用节点,所以 RainbowD 整体耗时平稳的增长,在图 7 中该现象不太明显的原因主要是 registry 镜像很小,只有 33 M 左右,所以变化不明显;特别的,在 RainbowD 中新加入的节点会公平的竞争 seeder 节点,在集群增大的过程中,随着虚拟节点的不断增多,随机选择到高性能的 demander 节点的概率越来越小,所以 RainbowD 完成分发任务的耗时也会快速增加.从数据来看,使用 RainbowD 分发镜像比使用 Docker Hub 获取镜像效率提升了 90% 以上; RainbowD 的效率相对 Docket 最多提升 90% 左右,并且在 70% 以上的情况下优于或等于 Dragonfly.

实验 2.1. 选择集群的规模为 20 个节点,镜像的大小变化如表 6 所示,统计不同的镜像分发方案的时间.统计结果如图 10 所示.

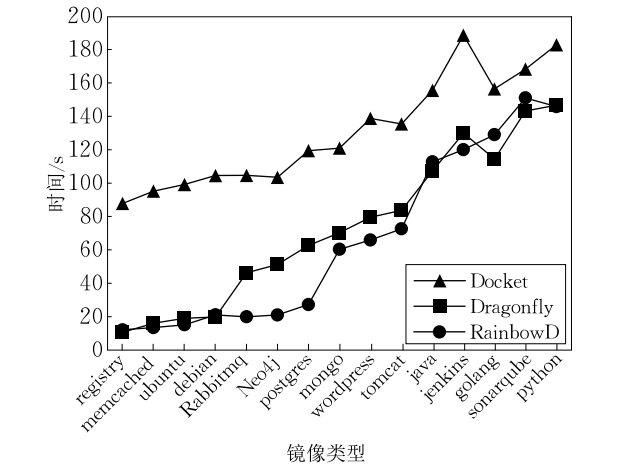


图 10 在 20 个节点上分发镜像的时间折线图

实验 2.2. 选择集群的规模为 50 个节点,镜像的大小变化如表 6 所示,统计不同的镜像分发方案的时间.统计结果如图 11 所示.

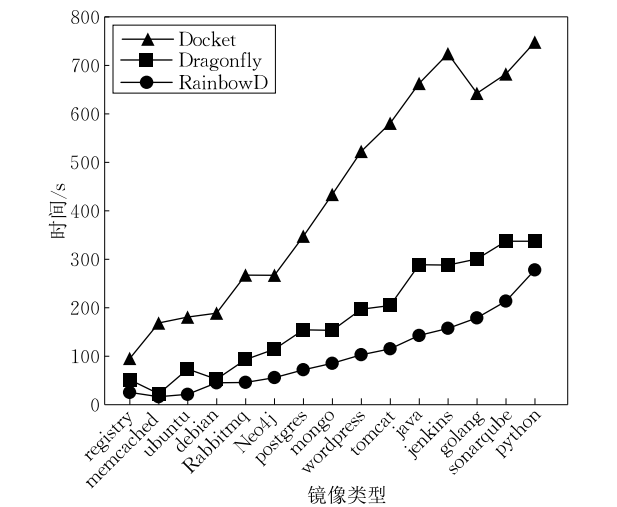


图 11 在 50 个节点上分发镜像的时间折线图

实验 2.3. 选择集群的规模为 100 个节点,镜像的大小变化如表 6 所示,统计不同的镜像分发方案的时间.统计结果如图 12 所示.

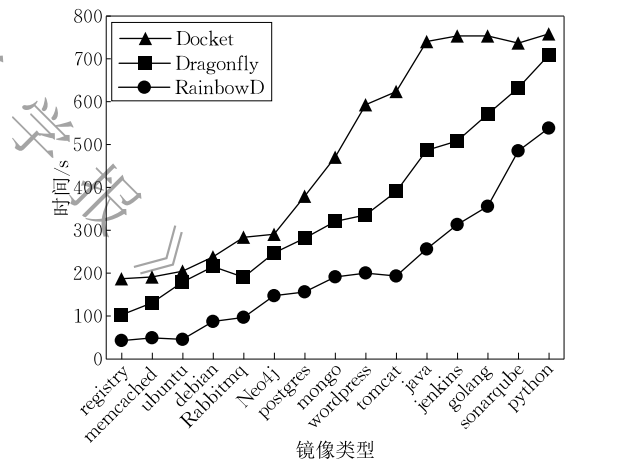


图 12 在 100 个节点上分发镜像的时间折线图

从折线图 10~图 12 可以看出,使用 Docket、Dragonfly 和 RainbowD 三种方案分发镜像时,随着镜像大小的增加,整体耗时呈现一个上升的趋势,但由于其他外界不稳定条件的影响,结果在一定误差范围内波动.究其原因,在集群规模一定的情况下,随着镜像大小的增加, Docket 在完成一次镜像分发操作的时间变长,集群规模一定的情况下整体耗时会呈现增长的趋势, Dragonfly 也是如此; RainbowD 虽然会对 seeder 节点进行选择,但是随着镜像大小的增加,分发镜像的时长也会增加,所以整体耗时同样呈现增长的趋势.从图 10~图 12 中数据可以看出,不管在集群规模为 20、50、100 个节点,在镜像分发的

任务上 RainbowD 的效率优于 Docket, RainbowD 有 80% 以上的结果近似或者优于 Dragonfly.

FID^[43] 这篇文章提出了一种使用 P2P 的思想分发镜像, 尽管没有公开源码, 但是可以根据文章中的实验结果做对比. 由于本文的实验环境和 FID 这篇文章中的实验环境有差别, FID 文章中使用的集群实验环境整体性能更优; 所以, 本文采取了一种间接对比的方式, 在各自的实验环境下, 分别比较 FID 和 Docket 的对比结果与 RainbowD 和 Docket 对比的结果更合理. 对比使用 FID 相对 Docket 提升的效率和本文中 RainbowD 相对 Docket 提升的效率, 对比结果如表 7、表 8 所示.

表 7 在 40 个节点上 RainbowD 和 FID 的对比

镜像大小	RainbowD 相对 Docket 的提升 / %	FID 相对 Docket 的提升 / %
30 M	90	93
200 M	79	82
500 M	82	55
1 G	53	50

表 8 在 80 个节点上 RainbowD 和 FID 的对比

镜像大小	RainbowD 相对 Docket 的提升 / %	FID 相对 Docket 的提升 / %
30 M	82	93
200 M	66	80
500 M	48	55
1 G	40	40

从表 7、表 8 可以看出, RainbowD 相对 Docket 的提升效率和 FID 相对 Docket 的提升效率相比有 62.5% 的结果是近似的, 相差结果在 5 个百分点左右; 在 40 个节点上分发 500 M 和 1 G 左右的镜像, RainbowD 的提升效果优于 FID; 以上可以看出 RainbowD 在分发镜像的任务上也具有高效的特性.

从上面对比实验的结果可以看出 RainbowD 在实验环境下取得了良好的效果. 在日常的生产生活中, 集群的规模越来越大, 云环境越来越复杂, 不同节点之间的异构特性越发明显, RainbowD 可以并发的处理大量节点同时获取镜像的任务, 而且在分发镜像时可以有选择地优先发挥性能优先级高的节点的作用, 所以在面对更大规模的集群时, RainbowD 依旧能够体现出其优良特性.

4.3 安全性实验

为了验证 RainbowD 的安全性, 本文选择了两种常见的网络攻击方式, 分别是 DNS 欺骗攻击和会话劫持攻击.

4.3.1 DNS 欺骗实验设计

将非法主机伪装成 Docket 的镜像仓库给集群

的节点分发非法的镜像. 实验主要有以下几个步骤:

过程 1. DNS 欺骗的过程.

- Step1. 在非法主机上构建非法镜像仓库.
- Step2. 将非法主机伪装成镜像仓库.
- Step3. IF: 集群内有节点发起获取镜像的请求;
THEN: 尝试将非法镜像分发到相应节点上.

以 hello-world 镜像为例, 将 hello-world 镜像做了非法修改, 实验结果如表 9 所示.

表 9 DNS 欺骗攻击结果

Docket	RainbowD
\$. /client -h "http://mini0" pull hello-world;latest docker run hello-world \$.you are attacked!	WARNING: POSSIBLE DNS SPOOFING DETECTED!

从表 9 可以看出, 使用 DNS 欺骗时, Docket 的仓库节点被非法更换, 集群中的节点获得了非法的镜像; 而 RainbowD 可以有效的防御 DNS 欺骗攻击, 保障了集群的安全性.

4.3.2 会话劫持攻击

利用会话劫持技术, 截取集群内节点的请求信息, 镜像重定向到恶意的镜像仓库地址实验主要有以下几个步骤:

过程 2. 会话劫持的过程.

- Step1. 攻击者劫持集群内节点发出的请求.
- Step2. 篡改返回报文.
- Step3. IF: 截取成功;
THEN: 尝试将非法镜像分发到相应节点上.

以 hello-world 镜像为例, 将 hello-world 镜像做了非法修改, 实验结果如表 10 所示.

表 10 会话劫持攻击结果

Docket	RainbowD
\$. /client -h "http://mini0" pull hello-world;latest \$docker run hello-world you are attacked!	SSH 协议采用了密钥进行加密, 无法进行简单的数据篡改.

从表 10 可以看出, 使用会话劫持攻击时, 使用 Docket 分发镜像时, 集群的请求返回报文被修改, 集群节点上得到的是非法的镜像, 而 RainbowD 使用 SSH 的加密特性, 可以有效的防御会话劫持.

云计算时代的到来导致数据的传输量呈井喷式增长, 安全问题也随之而来, 云环境随时随地都有被攻击的风险, 从上面的实验结果可以看出, 在抵御常见的网络攻击上, RainbowD 具有良好的效果, 可以降低云环境下集群被攻击的风险.

5 总 结

本文提出了一种面向异构云环境下的 Docker 镜像分发策略,提升了 Docker 镜像分发任务的效率,同时,在实验上分别和 Docket、Dragonfly 做直接对比,验证了本文提出的方案——RainbowD 在异构云环境中的高效性和安全性。

在大规模集群中使用容器部署服务或者应用时,容器镜像分发耗时太长的问题一直制约着开发者的效率,从 Docker Hub 上获取镜像的瓶颈主要有两个:一方面,由于网络路由路径过长,Docker Hub 将容器的镜像分发到目标节点要经过很长的网络传输才能达到,浪费了 Docker 镜像分发的时间;另一方面,Docker Hub 存在着 C/S 架构的弊端,镜像仓库本身的性能限制了镜像分发效率;国内镜像站点虽然能够缓解路由路径的问题,但是不能满足商业需求;Docket、Quay 和 DID 都使用了 P2P 网络的思想,为 P2P 网络在镜像分发中的应用奠定了基础,它们虽然解决了 C/S 架构的弊端,一定程度上缩短了镜像分发的时间,但是制作 torrent 文件、消息延迟、消息重复、数据冗余、加载镜像等都延长了镜像分发任务时间,效率还有待提升;Dragonfly 和 FID 继续做了优化,相对 Docket 等方式进一步提升了效率,但它们的效率受到了实际应用环境的制约,在异构云环境中不能达到理想状态,并且在分发过程中存在着消息延迟和重复等问题.针对以上的问题,本文提出了 RainbowD 镜像分发系统,为了缓解镜像仓库的压力,集群中其他拥有资源的节点也可以承担起镜像分发的任务;为了进一步提升效率,本文首先从异构云的自身特点入手,对节点进行优先级排序,在每一次镜像分发操作中选择最优的节点,从局部最优解尽可能的得到整个镜像分发任务的比较高效的调度次序;其次,不再像 P2P 网络一样每一个节点都向其他节点进行消息传递,并且采用 SSH 协议进行镜像分发,在解决 P2P 网络存在的消息延迟、消息重复和冗余等问题的基础上进一步提升了安全性,实验证明 RainbowD 可以预防大部分的网络监听、路由欺骗、DNS 欺骗、会话劫持等.在验证镜像分发效率的对比试验上,本文进行了直接实验对比和间接实验对比两种方式,实验结果表明:RainbowD 在异构的实验环境上取得了优秀的效果,比 Docket 的效率最少提升了 30%,在 80% 的实验结果上优于或近似于 Dragonfly 的效率,在相对 Docket 的提升效果上,62.5% 的情况下和 FID 方案

近似,在部分情况下优于 FID 方案。

从实验结果来看,RainbowD 在异构的实验集群环境下取得了很好的效果,但是随着云计算和大数据的发展,集群的规模只会愈来愈大,如何在大规模或超大规模集群上获得更好的效率将会是接下来研究的一个问题,而且当集群变成大规模或者超大规模时,节点由于宕机删除或者新节点加入的情况时有发生,针对新节点请求分发全部镜像的情况,RainbowD 将全部镜像分发的任务根据节点性能由高到低有选择的分配到多个节点上,提升该任务的并发程度,虽然不能达到理论上最理想的状态的 P2P 网络,但也能在一定程度上保证高效性,这个应用案例将是 RainbowD 未来性能优化的方向;不仅如此,随着云计算业务的复杂,开发者开发的独立镜像大小也会越来越大,镜像内层与层之间的联系越来越多、镜像之间重复的层文件也会越来越多,人工的分析的工作量将会越来越大,如何更高效、合理、智能的利用 Docker 的这些特性将是接下来要解决的问题,还有如何将 G、10G 等为基础单位的超大规模镜像高效的分发也将是接下来要攻克的难关,以此来满足行业飞速发展的需求。

致 谢 感谢科技部项目《智能化数据中心管理、编程规范与应用生态》(课题编号:2016YFB1000505)与国家自然科学基金委员会项目《基于天河二号的生物医学健康大数据应用支撑平台》(项目基金号:U1611261)对本工作支持,感谢各位老师的耐心指导,感谢《计算机学报》编辑部老师们的辛勤工作!

参 考 文 献

- [1] Xiu Chang-Hong, Liang Jian-Kun, Xin Yan. Overview of virtualization technology. *Network Security Technology & Application*, 2016, (5): 18-19(in Chinese)
(修长虹, 梁建坤, 辛艳. 虚拟化技术综述. *网络安全技术与应用*, 2016, (5): 18-19)
- [2] Denning P J. Third generation computer systems. *ACM Computing Surveys*, 1971, 3(4): 175-216
- [3] Olbert A G. Extended control program support: VM/370: A hardware assist for the IBM virtual machine facility/370. *ACM SIGMICRO Newsletter*, 1978, 9(3): 8-25
- [4] Goldberg R P. Survey of virtual machine research. *Computer*, 1974, 7(6): 34-45
- [5] Li Peng. Selecting and using virtualization solutions: Our experiences with VMware and VirtualBox. *Journal of Computing Sciences in Colleges*, 2010, 25(3): 11-17
- [6] Raho M, Spyridakis A, Paolino M, et al. KVM, Xen and Docker: A performance analysis for ARM based NFV and cloud computing//*Proceedings of the 2015 IEEE 3rd Workshop on*

- Advances in Information, Electronic and Electrical Engineering (AIEEE). Riga, Latvia, 2015: 1-8
- [7] Boettiger C. An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Operating Systems Review*, 2014, 49(1): 71-79
- [8] Bernstein D. Containers and cloud: From LXC to docker to kubernetes. *IEEE Cloud Computing*, 2015, 1(3): 81-84
- [9] Calarco G, Casoni M. On the effectiveness of Linux containers for network virtualization. *Simulation Modelling Practice & Theory*, 2013, 31(2): 169-185
- [10] Schmidt M, Fallenbeck N, Smith M, et al. Secure service-oriented grid computing with public virtual worker nodes// *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications*. Patras, Greece, 2009: 555-562
- [11] Schmidt M, Fallenbeck N, Smith M, et al. Efficient distribution of virtual machines for cloud computing// *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing*. Pisa, Italy, 2010: 567-574
- [12] Zhang Sui-Zheng. Client/Server architecture and technology development. *Railway Computer Application*, 1995, 4(4): 11-13(in Chinese)
(张遂征. Client/Server 体系结构及其技术发展. *铁路计算机应用*, 1995, 4(4): 11-13)
- [13] Tian Chen. Research on Internet Content Distribution Optimization [Ph. D. dissertation]. Huazhong University of Science and Technology, Wuhan, 2008(in Chinese)
(田臣. 互联网内容分发优化问题研究[博士学位论文]. 华中科技大学, 武汉, 2008)
- [14] Wang Jian-Yong, Gong Fu-Ting, Li Yu-Ling. Resource searching strategy of reducing redundancy in unstructured P2P network. *Computer Engineering and Applications*, 2010, 46(36): 122-125(in Chinese)
(王建勇, 龚伏廷, 李玉玲. 非结构化 P2P 网络中减少冗余的搜索策略. *计算机工程与应用*, 2010, 46(36): 122-125)
- [15] Donet J A D, Pérez-Solà C, Herrera-Joancomartí J. The Bitcoin P2P network// *Proceedings of the 1st Workshop on Bitcoin Research (in Association with Financial Cryptography 14)*. 2014: 1-7
- [16] Lin Huai-Qing, Li Zhi-Tang, Zhang Ye-Jiang. Improving flooding protocol for unstructured P2P network. *Application Research of Computers*, 2009, 26(10): 3874-3876
- [17] Xu G, Wang G, Liu J. A hybrid redundancy approach for data availability in structured P2P network systems// *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*. Melbourne, Australia, 2008: 195-199
- [18] Zheng Q, Lu X, Zhu P, et al. An efficient random walks based approach to reducing file locating delay in unstructured P2P network// *Proceedings of the IEEE Global Telecommunications Conference*. St. Louis, USA, 2005
- [19] Chen Shiping, Li Yuan, Rao Kaihua. Building a scalable P2P network with small routing delay// *Proceedings of the 10th Asia-Pacific Web Conference, Progress in WWW Research and Development (APWeb 2008)*. Shenyang, China, 2008: 456-467
- [20] Margariti S V, Dimakopoulos V V. A study on the redundancy of flooding in unstructured P2P networks. *Parallel Algorithms & Applications*, 2013, 28(3): 214-229
- [21] Li Jun-Qing, Pan Quan-Ke, Wang Wen-Hong, et al. Research on ant colony optimization in defending DDoS attacks of P2P network. *Application Research of Computers*, 2009, 26(1): 339-341(in Chinese)
(李俊青, 潘全科, 王文宏等. 蚁群优化在 P2P 网络防范 DDoS 攻击中的应用研究. *计算机应用研究*, 2009, 26(1): 339-341)
- [22] Zhu Yun-Peng, Lu Yu-Liang. Research on DoS attack to DHT of P2P. *Network Security Technology & Application*, 2007, (3): 45-47(in Chinese)
(朱云鹏, 陆余良. 针对 P2P 网络中 DHT 的 DoS 攻击技术研究. *网络安全技术与应用*, 2007, (3): 45-47)
- [23] Zhang Ji-Wen, Dong Yu. Analyses worm attack in large scale P2P network. *Control & Management*, 2006, 22(24): 245-246(in Chinese)
(张基温, 董瑜. 大规模 P2P 网络下蠕虫攻击的研究. *微计算机信息*, 2006, 22(24): 245-246)
- [24] Liu Yue, Li Qiang, Li Zhou-Jun. Survey of P2P network security and defense mechanism. *Computer Science*, 2013, 40(4): 9-13(in Chinese)
(刘悦, 李强, 李舟军. P2P 网络安全及防御技术研究综述. *计算机科学*, 2013, 40(4): 9-13)
- [25] Xu Xiao-Dong, Li Gang, Yang Yan. Study on defense of unstructured and uncentralized P2P network DDoS attacks. *Application Research of Computers*, 2012, 29(12): 4618-4621(in Chinese)
(许晓东, 李刚, 杨燕. 非结构去中心化的 P2P 网络 DDoS 攻击的防御研究. *计算机应用研究*, 2012, 29(12): 4618-4621)
- [26] Nickoloff J. Docker in Action. New York, USA: Manning Publication Company, 2016
- [27] Verma M, Dhawan M. Towards a more reliable and available Docker-based container cloud. *arXiv preprint arXiv:1708.08399*, 2017
- [28] Qian Wei, Cui Dong-Mei. Building a container sandbox server based on Docker. *Network Security and Information*, 2016, (9): 151-154(in Chinese)
(钱卫, 崔冬梅. 基于 Docker 构建容器式沙箱服务器. *网络安全和信息化*, 2016, (9): 151-154)
- [29] Fries R, Parthasarathy S, Hunt A G. Virtual machine image analysis. 2014
- [30] Zheng Ling, Jiang Meng. Research on Docker elastic scheduling architecture. *Computer Programming Skills and Maintenance*, 2016, (23): 15-17(in Chinese)
(郑玲, 江萌. 基于 Docker 弹性调度架构的研究. *电脑编程技巧与维护*, 2016, (23): 15-17)
- [31] Preeth E N, Mulerickal F J P, Paul B, et al. Evaluation of Docker containers based on hardware utilization// *Proceedings of the 2015 International Conference on Control, Communication and Computing India (ICCC 2015)*. Trivandrum, India, 2015: 697-700

[32] Anderson C. Docker. Software Engineering, 2015, 32(3): 102-c3

[33] Merkel D. Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014: 239-243

[34] Giannakopoulos I, Papazafeiropoulos K, Doka K, et al. Isolation in Docker through layer encryption//Proceedings of the IEEE 37th International Conference on Distributed Computing Systems. Atlanta, USA, 2017: 2529-2532

[35] Shu R, Gu X, Enck W. A study of security vulnerabilities on Docker Hub//Proceedings of the 7th ACM on Conference on Data and Application Security and Privacy. Scottsdale, USA, 2017: 269-280

[36] Cook J. Docker Hub. Docker for Data Science. Berkeley, USA: Apress, 2017

[37] Huang X W, Hsieh C Y, Wu C H, et al. A token-based user authentication mechanism for data exchange in RESTful API//Proceedings of the 18th International Conference on Network-Based Information Systems. Taipei, China, 2015: 601-606

[38] Lee H, Fox G C. Efficient software defined systems using common core components//Proceedings of the IEEE 10th International Conference on Cloud Computing. Honolulu, USA, 2017: 407-414

[39] Zhang Y, Yin G, Wang T, et al. An insight into the impact of Dockerfile evolutionary trajectories on quality and latency //Proceedings of the IEEE 42nd Annual Computer Software and Applications Conference. Tokyo, Japan, 2018: 138-143

[40] Zhou Yi, Deng Yu-Hui. Reduction strategy for Docker images based on probabilistic model. Journal of Chinese Computer Systems, 2018, 39(9): 1908-1913(in Chinese) (周毅, 邓玉辉. 一种概率模型的 Docker 镜像删减策略. 小型微型计算机系统, 2018, 39(9): 1908-1913)

[41] Nathan S, Ghosh R, Mukherjee T, et al. CoMICon: A co-operative management system for Docker container images //Proceedings of the IEEE International Conference on Cloud Engineering(IC2E). Vancouver, Canada, 2017: 116-126

[42] Liang M, Shen S, Li D, et al. HDID: An efficient hybrid Docker image distribution system for datacenters//Proceedings of the Software Engineering and Methodology for Emerging Domains. Singapore, 2016: 179-194

[43] Wang Kangjin, Yang Yong, Li Ying, et al. FID: A faster image distribution system for Docker platform//Proceedings of the 2017 IEEE 2nd International Workshops on Foundations and Applications of Self * Systems (FAS*W). Tucson, USA, 2017: 191-198

[44] Zheng C, Rupprecht L, Tarasov V, et al. Wharf: Sharing Docker images in a distributed file system//Proceedings of the ACM Symposium on Cloud Computing. California, USA, 2018: 174-185

[45] Xie X L, Wang P, Wang Q. The performance comparison of native and containers for the cloud//Proceedings of the 2018 International Conference on Smart Grid and Electrical Automation (ICSGEA). Xiangtan, China, 2018: 378-381

[46] Ylonen T. SSH: Secure login connections over the Internet//Proceedings of the Conference on USENIX Security Symposium. USENIX Association, Berkeley, USA, 1996, 6: 4_4-4_9

[47] Iyappan P, Arvind K S, Geetha N, et al. Pluggable encryption algorithm in secure Shell(SSH) protocol//Proceedings of the International Conference on Emerging Trends in Engineering & Technology. Nagpur, India, 2009: 808-813

[48] Subramanyam, Udayakiran. Study and Sample Implementation of the Secure Shell Protocol (SSH) [M. S. dissertation]. University of North Texas, Denton, Texas, USA, 2003

[49] Dusi M, Gringoli F, Salgarelli L. A model for the study of privacy issues in secure shell connections//Proceedings of the 4th International Conference on Information Assurance and Security. Washington, USA, 2008: 311-317

[50] Huang Xiaoqiao, Guo Hongbin. Construct the secure shell transport model and analyze its security performance. Procedia Environmental Sciences, 2011, 11(part-PA): 0-188

[51] Mokhtarnameh R, Muthuvelu N, Ho S B, et al. A comparison study on key exchange-authentication protocol. International Journal of Computer Applications, 2010, 7(5): 5-11



ZHANG Hao, M. S. candidate. His current research interests include cloud computing and machine learning.

SUN Yu-Zhong, Ph. D. , professor, His main research interests include big data intelligence analysis (machine learning) and calculation.

XIAO Li, Ph. D. , associate professor. His main research interests include artificial intelligence, medical image, biological

computation.

TANG Yong, M. S. candidate. His current research interests include cloud computing and data mining.

HU Man-Man, M. S. candidate. Her current research interests include machine learning and data mining.

DU Qin-Yuan, M. S. candidate. His current research interests include software engineering and cloud native database.

CAI Zhi-Bin, undergraduate. His current research interest is cloud computing.

FENG Bai-Ming, Ph. D. , professor. His current research interests include distributed and parallel computing.

Background

With the developing of the cloud computing era, Docker technology is becoming more and more widely used. How to efficiently distribute Docker images in a large cluster has been a hot issue. At the appearance of Docker, Docker provided users with the official image registry named Docker Hub. Users from all over the world are free to get images on Docker Hub. However, because the network routing path is too long and the Docker Hub need to respond to many requests at the same time, Docker Hub has a relatively high transmission pressure. The efficiency of obtaining images from the Docker Hub is very low.

Although domestic Docker registries shorten the routing path of the images transmission, the efficiency still cannot meet the demand. In order to reduce the pressure of the Docker registry, P2P technology has been applied to the tasks of Docker image distribution, which generally improved the efficiency of Docker image distribution by 80% or even 90%, such as Docket, Dragonfly, FID, etc. However, when faced with complex heterogeneous cloud environments, methods such as Docket have limitations in efficiency.

In order to solve the above problems, this paper proposes a new image distribution system named RainbowD in heterogeneous cloud environments. Firstly, it uses all the nodes in

the cluster to release the load of the registry. Secondly, considering the real characteristics of the heterogeneous cloud environment, a performance-prioritized method is used to dynamically select nodes with high performance for improving efficiency with limited resources. Finally, because the nodes are prioritized, each node in the cluster does not need to send messages to all the other nodes, and RainbowD use the SSH protocol to distribute the images for the first time, which solves some disadvantages in the P2P network. Such as message latency, message duplication, and redundancy. Specially, the drawbacks of the security of the BitTorrent protocol have been criticized by scholars. Considering the security of clusters in the cloud environment, the SSH protocol encryption feature and security verification are used to improve the security of the cloud environment.

We evaluate the proposed method on some cases. The result showed that our system performed better than other mainstream methods.

The proposed work is under the support of the Networked Operating System for Cloud Computing (Grant No. 2016YFB1000505) and the Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund (U1611261).