

# 一种面向多租户的 Linux 容器集群组网方法

朱瑜坚<sup>1,2</sup> 马俊明<sup>1</sup> 安博<sup>1,2</sup> 曹东刚<sup>1,2</sup>

(高可信软件技术教育部重点实验室(北京大学), 北京 100871)<sup>1</sup>

(北京大学信息科学技术学院, 北京 100871)<sup>2</sup>

**摘要** 目前越来越多的云平台开始采用容器组成的集群为云服务提供运行环境, 而其中如何在多租户环境下为用户的容器集群提供高效可用的网络成为了一个重要的技术问题。对此, 以 Linux 容器为例, 提出了一种面向多租户的 Linux 容器集群组网方法。这种方法参考了 Kubernetes 的组网方法, 并在其基础上简化了网络结构, 并引入了网络隔离, 使得构建而成的网络能够满足多用户场景下的需求。论文描述了此种组网方法在小规模和大规模应用场景下的设计和它在真实的容器云平台 Docklet 中的实现, 实现的代码是开源的, 并且, 还进行了实验与评估。实验证明, 这一组网方法所构成的虚拟网络与原生网络的性能相当的接近, 其 TCP 出口下行带宽与原生网络相差 0.5% 以内, 而 TCP 内部通信带宽大约只损失 3.5%, 且对批处理型应用和长服务型应用都有良好的支撑。

**关键词** 云计算, Linux 容器, 容器组网, 软件定义网络, 多租户

中图法分类号 TP393 文献标识码 A DOI (投稿时不提供 DOI 号)

## A Linux Containers Networking Approach for Multiple Tenants

ZHU Yu-jian<sup>1,2</sup> MA Jun-ming<sup>1</sup> AN Bo<sup>1,2</sup> CAO Dong-gang<sup>1,2</sup>

(Key Lab of High Confidence Software Technologies, Peking University, Beijing 100871, China)<sup>1</sup>

(School of Electronic Engineering and Computer Science, Peking University, Beijing 100871, China)<sup>2</sup>

**Abstract** At present, more and more cloud platforms begin to use Linux container cluster to provide runtime environment for cloud services. But how to build a stable and high-performance network for a user's Linux container cluster under multi-tenant circumstance is an important technical problem. A networking approach of Linux container cluster for multiple tenants is proposed in this paper. Compared to that of Kubernetes, our approach simplifies the network architecture and introduces network isolation. Our network can meet the needs under multi-tenant circumstance. This paper describes the design of the approach with a small and large scale of clusters and users and explains the implementation of it in a real cloud containers platform Docklet. The source codes are open source on GitHub. Besides, evaluation results show that the container network performance of our approach is close to the original network. The TCP export downlink bandwidth is different with the original one within 0.5% and the TCP internal bandwidth gets about 3.5% loss. The batch job and long service applications are also well supported by the approach.

**Keywords** Cloud Computing, Linux Containers, Containers Networking, Software Defined Network, Multiple Tenants

### 1 引言

如今, 伴随着数据处理规模的增长和对管理

大规模应用系统的需求, 相比虚拟机, 具有良好

到稿日期: 返修日期:

本文受国家重点研发计划基金(2016YFB1000105), 国家高技术研究发展计划 863 基金(2015AA01A202), 国家自然科学基金(61690201, 61421091)资助。

朱瑜坚(1993-), 男, 硕士生, 主要研究方向为云计算和系统软件; 马俊明(1994-), 男, 博士生, 主要研究方向为云计算和系统软件; 安博(1992-), 男, 博士生, 主要研究方向为云计算, 系统软件, 分布式计算; 曹东刚(1975-), 男, 博士, 副教授, CCF 高级会员, 主要研究方向系统软件, 并行计算与分布式计算, E-mail: caodg@pku.edu.cn。

性能的、更轻量的 Linux 容器 (Linux Container, 简称 LXC)<sup>[1][2]</sup>得到了越来越多的应用。一些知名的 Web 服务提供商开始采用 Linux 容器或其他容器(如 Docker)包装、管理他们的 Web 服务和应用, 如在 Google 的大规模集群管理项目 Borg 中, 所有的作业和应用都是运行在 Linux 容器内的<sup>[3]</sup>。再比如国内的著名问答网站“知乎”也采用了 Docker 容器搭建他们的 Web 服务<sup>[4]</sup>。此外, 一些云计算服务提供商也开始将容器包装成“虚拟机”提供给用户使用, 如 Amazon, 阿里云等。

在这些各式各样的应用场景中, 不可避免的是容器集群的连网、组网需求。在这些不同的应用场景中, 组网的需求也不尽相同。一些常用的开源容器管理软件, 如 Kubernetes, 给出了比较通用的解决方案, 能够基本满足容器集群的连网需求。但在某些场景下(比如容器集群作为服务提供给用户), 容器集群是由多个用户共享或是由多个用户容器集群所构成的, 这时容器集群组网不仅需要满足网络的可用性和性能方面的需求, 还需要满足用户网络隔离性的需求, 也即要求不同用户的容器集群是在不同的局域网内, 或者说是互相二层不可达的。而 Kubernetes 等一些常用的开源容器管理软件是为单个用户服务的, 也因此难以满足这一场景下的组网需求。

为了满足多租户或多用户场景下容器集群的组网需求, 本文提出了一种面向多租户的 Linux 容器集群组网方法。这一组网方法参考 Kubernetes 的组网方法, 采用了 VETH Pair, Open vSwitch (开放虚拟交换标准, 简称 OVS), GRE (Generic Routing Encapsulation, 通用路由封装)隧道等工具和技术, 并引入了 Kubernetes 中没有的用户网络隔离, 为每个用户的容器集群提供了可用的、独立的、性能较好的虚拟网络。尽管这种方法是基于 Linux 容器集群的, 但这种方法可扩展到不同的容器上, 具有较好的通用性。

针对不同规模的应用场景, 这一组网方法有不同的设计。一种小规模应用场景下, 例如小型企业和科研机构的私有云中, 这时, 物理集群规模小且只有一个节点可以访问外网, 同时用户规模也较小。在这一场景下, 这一组网方法采用单出口、VLAN (Virtual Local Area Network, 虚拟局域网)隔离的设计来构建网络。这种设计有较低的管理负载, 并且结构简单、清晰。而在较大规模应用场景下, 例如中大型企业的私有云和一些云服务商的云计算服务器集群中, 这时, 物理集群规模较大且有多个节点可以访问外网, 同时用户规模也较大。在这一场景下, 这一组网方法采用多出口、用户独享式的设计来构建虚拟网络。这种设计可以服务更多的用户, 同时在多用户的使用场景下有更好的带宽性能, 但也有较高的管理负载。

概括而言, 这一组网方法有如下特征和优势:

(1) 面向多租户。不同用户的容器集群间是在不同的局域网中, 也即互相二层不可达, 具有较好的网络隔离性, 能够为多个用户的容器集群提供网络服务。

(2) 较好的性能。实验证明, 这一组网方案所构成的虚拟网络与原生网络的性能相当的接近, 其 TCP 出口下行带宽与原生网络相差 0.5% 以内, 而 TCP 内部通信带宽大约只损失 3.5%。

Docklet<sup>[5]</sup>是北京大学软件工程研究所高可信软件实验室自主开发的一个基于 Linux 容器集群的开源轻量云平台, 并已在北京大学和北理工大数据系统软件国家工程实验室中安装提供服务。本文将这一组网方法(包括小规模和大规模应用场景下的设计)在 Docklet 项目中实现, 实现的代码在 GitHub 上开源维护, 并进行相关实验以评估。

本文之后的部分如下所述: 第二节叙述了相关工作, 第三节描述了小规模应用场景下的组网方法所形成的网络结构, 第四节则描述了在大规模应用场景下的组网方法, 第五节简单的介绍了

Docklet 和这一组网方法在其中的实现,第六节给出了一些实验评估,最后一节给出了一些结论,同时描述了未来工作。

## 2 相关工作

Kubernetes 是 Google 开源的容器集群管理系统<sup>[6][7]</sup>,其提供应用部署、维护、扩展机制等功能,利用 Kubernetes 能方便地管理跨机器运行容器化的应用。在 Kubernetes 中,相关的一个或多个容器构成一个 Pod,通常 Pod 里的容器运行相同的应用。Pod 包含的容器运行在同一个 Host (物理机)上,看作一个统一管理单元<sup>[8]</sup>。而 Kubernetes 的组网方法<sup>[9]</sup>可以如下描述:每个物理机上有许多 pod,每个 pod 通过 VETH Pair 与 Linux 网桥上的一个端口连接,而 Linux 网桥再与 OVS (Open vSwitch) 网桥连接。最后,每个物理机上的 OVS 网桥通过原生网络上架设 GRE/VxLAN 隧道进行连接。本文提出的组网方法参考了 Kubernetes 的组网方法,但相比 Kubernetes 的方法,本文的组网方法中去掉了 Linux 网桥,只采用 OVS 搭建网桥,简化了结构,并且引入了用户隔离,使得网络可以为多个用户服务。

Docker<sup>[10]</sup>是一个供开发者和系统管理员开发、迁移和运行应用的容器平台。Docker 能将应用以及应用所依赖的运行环境打包成 Docker 容器,使得其可以在任何 Linux 发行版上运行。对于单个 Docker 容器,Docker 提供了 bridge 网络,即桥接的方式,这一连网方法本质上是与本文提出的组网方法是相同的,都是利用 Linux 内核(对于单个容器而言)。而对于容器集群,Docker 提供了 overlay 网络,可以让容器跨机通信。这个网络本质是 Docker 自己利用 SDN (Software Defined Network, 软件定义网络)技术创建的一个跨机的虚拟网络,与用 OVS 网桥和 GRE 隧道搭建的网络相似<sup>[11]</sup>。可以看出,本文与 Docker 容器的组网方法是类似的,并且都有引入用户的隔离(Docker 中,处于不同 overlay 网的容器二层不可达),但是,Docker 采

用的是自己开发的 SDN 技术,而本文采用的是开源软件 Open vSwitch 搭建网络。而第 6 节的实验可证明,本文所提出的组网方法所形成的网络的带宽性能与 Docker 的网络相近,且在内部通信时延上略优于 Docker。

## 3 小规模应用场景下的设计

### 3.1 小规模应用场景下的设计的网络结构

在小规模应用场景下(本文的规模指租户规模,物理机节点规模和容器规模),这一面向多租户的 Linux 容器集群组网方法的网络结构(以两个用户为例)如下图 1 所示。

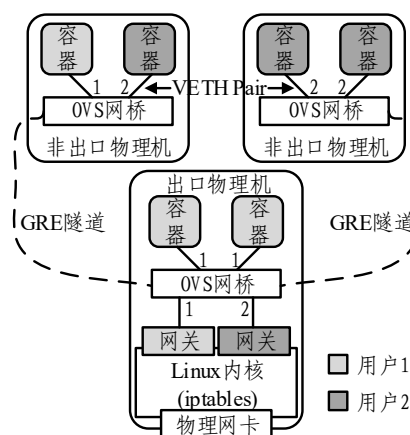


图 1 小规模应用场景下的组网方法的网络结构 (两个用户)

从图 1 中可以看出,在这两个用户的例子中,每个物理机上都有一个 OVS 网桥(或称为虚拟交换机),这些网桥通过 GRE 隧道进行连接,形成了一个广播域。每个网桥有若干端口,而 LXC (Linux 容器)则通过 VETH Pair 与 OVS 网桥上的其中一个端口连接(或者说 LXC 上的端口与 OVS 网桥的端口形成了一对 VETH Pair),所有端口原本共属一个广播域,但是给 OVS 网桥端口加上 tag (图 1 中网桥端口连接处的数字)之后会形成 VLAN,这是由于 OVS 中,网络包只会在 tag 相同的端口之间转发,因此相同 tag 的端口就属于一个广播域,从而形成一个隔离的局域网,这个 tag 也就可以看成 VLAN ID,每个用户唯一。

也即在这一例子中,用户 1 拥有 3 个容器,其网络以 tag 为 1 进行标志。用户 2 也拥有 3 个

容器，其网络以 `tag` 为 2 进行标志。他们的网关（`gw`）都在集群唯一的出口物理机上。且网关与 OVS 网桥连接的端口也以相应的 `tag` 进行标志。

而每个用户网关（`gw`）都好像宿主机上的一个网卡，这些网络包到达这些网关后，就好像从外部网络到达网卡一样，会进入 Linux 内核进行路由判断，最后决定是本地接收还是从相应的网卡转发出去。

此外，GRE 隧道只连接非出口物理机和出口物理机，因此，出口物理机成为整个网络的中心，整个网络呈现星型拓扑结构，从而避免二层环路。

概括起来，这一网络结构可以如下描述：每个用户的所有容器节点同属于一个局域网，他们都与宿主机上的 OVS 网桥的其中一个端口是 Veth Pair，且以相同的 `tag` 进行标志，而且这个局域网的网关都在集群唯一的出口物理机上。

可以看出，相比 Kubernetes 的网络，这一组网方法中只采用 OVS 来搭建网桥，而没有使用 Linux 网桥，一定程度上简化了网络结构。同时，利用 OVS 的 `tag` 分隔出了 VLAN，引入了用户网络隔离。

### 3.2 小规模应用场景下的设计的特点

**面向多租户：**该方法可以为多个租户服务

**共享式网络：**用户容器共享整个虚拟网络，通过 VLAN ID 划分虚拟局域网。每个物理机只有一个 OVS 网桥，非出口物理机和出口物理机之间只可能存在一个 GRE 隧道。

**集中式网关：**用户网关集中部署在集群唯一的出口物理机上。

### 3.3 小规模应用场景下的设计的优点和局限

概括起来，这一设计具有如下优点：

- (1) 结构清晰、简单。
- (2) 管理负载较小，用户网络的创建和删除响应速度快。

但同时，这一设计也具有如下局限：

- (1) 容器集群的网络出口易堵塞。由于集中

式网关，容器集群目的地址在外部的网络包都将由出口物理机的网卡出口发出，容易造成网络拥堵。

(2) 容器集群的内部通信易堵塞。出口物理机作为网络结构的中心，所有的内部网络包都通过它进行转发，容易造成网络拥堵。

(3) VLAN ID 耗尽。每个用户拥有一个唯一的 VLAN ID，但 VLAN ID 只有 4096 个，也即该网络最多只能为 4096 个用户服务，服务规模较小。

但是，在小规模应用下，带宽是足够的，并且在单出口的物理集群中，第一点局限是无法避免的。而对于小规模的用户数，4096 个 VLAN ID 也是足够的。因此，概括而言，在小规模应用下，这三点局限不是显著的。

## 4 大规模应用场景下的设计

### 4.1 大规模应用场景下的设计的网络结构

尽管在小规模应用场景下，上述局限不是那么显著，但在大规模应用场景下，尤其是租户规模较大时，上述局限就显得十分显著，并会带来十分不好的用户体验。

从小规模应用场景下的设计的网络结构可以看出，第一点和第二点局限都是由集中式网关造成的：网关即是网络出口，网关集中在一台物理机便会造成网络出口的易堵塞；网关也是每个容器集群网络的转发中心，网关集中在一台物理机便会造成容器网络内部通信的易堵塞。因此，针对这两点局限，一种优化方案便是分布式网关，即将用户网关分布式的部署在不同的物理机上。而第三点局限的由来是因为采用了 VLAN ID，因此为了解决这一局限的优化方案便是抛弃使用 VLAN ID，转而采用其他的用户网络隔离办法。

综上所述，为了解决小规模场景下的设计存在的局限和问题，并同时满足大规模多用户情况下的需求，本文结合了前述两种优化方案，在之前设计的基础上发展了一种适合大规模应用场景下的设计，其网络结构如下图 2 所示。

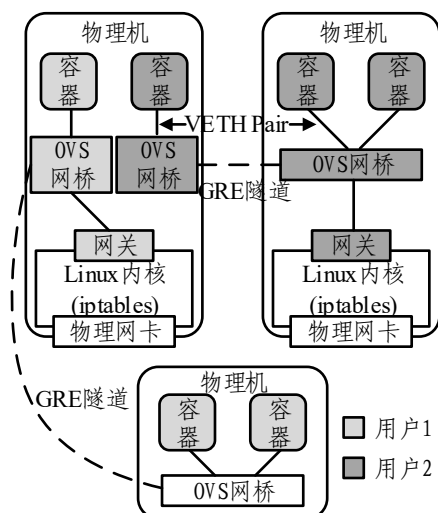


图 2 大规模应用场景下的组网方法的网络结构（两个用户）

从图 2 可以看出，用户 1 和用户 2 的网关不再集中分布在一个物理机上，而是分布在不同的物理机上。同时，OVS 网桥不再共享，而是每个用户拥有独享的、独立的虚拟网络。而在每个用户的虚拟网络中，GRE 隧道连接着用户虚拟网络中的非出口物理机和出口物理机，也形成星型拓扑。但是，由于用户网关可能分布在不同的物理机上，因此每个用户的出口物理机也可能不尽相同，也即每个虚拟网的中心节点也很可能不相同。

概括而言，这一设计基本沿用了之前设计中所使用的工具和技术，但对网络结构进行了一定的优化。相比之前的设计，网络结构主要有以下两点优化（同时也是特点）：

**分布式网关：**集群内部所有物理机平权（要求所有物理机都能访问外部网络），每个物理机都有可能作为容器集群的出口物理机，用户网关可分布在任意可连网物理机上。

**独享式网络：**去除 VLAN，每个用户都拥有独立的由 OVS 网桥构成的虚拟网络，不再与其他用户共享。每个物理机上可能存在多个 OVS 网桥（只要有容器就有网桥），同时，两个物理机之间可能存在多个 GRE 隧道（以不同的 key 值区分）。

#### 4.2 大规模应用场景下的设计的优势

相比小规模应用场景下的设计，理论分析和第 6 节的实验可证明，这一设计有如下优势：

(1) 多出口减少网络出口堵塞。这使得容器集群的网络出口负载均摊于整个物理集群上，而不是集中于唯一一个出口物理机，从而减少了网络出口堵塞，提高了网络访问性能，改善了用户体验。

(2) 内部网络通信性能较高。尽管每个用户的虚拟网仍是星型拓扑，但中心出口物理机很可能不相同，从而使得内部通信负载得到均摊，而不是集中于原来的唯一一个出口物理机，提高了内部网络通信性能。

(3) 避免了 VLAN ID 耗尽。因为每个用户是独享虚拟网，也就不需要 VLAN 就可以隔离用户间网络，从而避免了 VLAN ID 耗尽的问题。GRE 隧道的 Key 值类似于原来的 VLAN ID 的作用，但其由 32 位无符号整数表示，这使得这种组网方法可以服务更多的用户。

当然，与之前的设计一样，这种方法是面向多租户的，能够满足多用户情况下的需求。

#### 4.3 大规模应用场景下的设计的局限

另一方面，这种设计也存在如下局限：

(1) 要求所有物理机都能访问外部网。若是其中有一个无法访问外部网，那么会让在上面部署的用户网关无法向外收发网络包，从而导致用户无法访问自己的容器节点。

(2) 不同用户间的容器集群不一定能互相访问。如图 2 的结构图所示，网络包从网关出来后是进入 Linux 内核的，对于那些网关不在同机的容器集群节点，其路由路径本机也是无法知道的，若这些包的地址是其他用户网关，Linux 内核只能将其发送往默认网关。最后，由于这些地址是内部地址（若是公网地址，则可通过三层路由到达），默认网关一定会丢弃，于是这些包就无法抵达其他网关内的容器集群节点了。

(3) OVS 网桥数量和 GRE 隧道数上升，增加 OVS 管理负载。相比之前的结构，优化后的网

桥数量会大大上升（大约与物理机节点数乘以用户数同阶），同时 GRE 隧道数也会大大上升（与网桥数量同阶），这会增加 Open vSwitch 的管理负载。规模变大后，Open vSwitch 的管理有可能会变得低效，但根据本文第 6 节的实验，网络性能几乎不会有改变。

但是，这几点局限也不是那么显著。对于第一个局限，只要提供集中式网关和分布式网关自由切换的功能即可较好的解决。切换时需要进行网关在物理机间的迁移。

而对于第二个局限，在常见的使用场景中，用户间的通信需求是比较小的，即便有也是仅限小数据量的通信（交流）。用户更多的是需求外部与内部的大数据量通信（传输数据等）和集群内部的通信（分布式计算，如 MapReduce）。此外，用户间共享数据也可以通过分布式文件系统予以实现。

至于最后一个局限，可以提供网络动态创建的功能，只有容器处于活跃或运行状态时，才创建网络，而容器处于不活跃或停止状态时，就删除容器所需要的虚拟网络。用户使用容器大致有两种需求：长服务和批处理作业。长服务如 web 服务器，主要是为其他应用和用户提长久的服务。而批处理作业主要为数据处理和计算，典型的如 MapReduce 作业，这类需求通常需要较多计算节点，但使用的时间通常不太长，因此这类需求的容器生命周期通常不太长，网络动态创建和删除的功能可以大大减少这类需求的网络管理负载。

综上所述，小规模应用场景下的设计结构更加简单、清晰，同时有较低的管理负载，但在多用户使用下性能会比较差（第 6 节的实验也会证明这一点）。而大规模应用场景下的设计结构较为复杂，同时有较高的管理负载，但是在多用户使用下性能更好。

## 5 面向多租户的组网方法在 Docklet 平台中的实现

### 5.1 Docklet 简介

Docklet 是北京大学软件工程研究所的一个开源项目，遵从新 BSD 协议，项目代码在 GitHub 上进行维护，并已在北京大学和北理工大数据系统软件国家工程实验室中安装提供服务。

Docklet 是为拥有中小型数据中心的企业或科研机构设计的轻量级云平台，使用户能方便地共享使用有限的计算资源。Docklet 基于 ClaaS（Cluster as a Service）概念<sup>[5]</sup>，这一创新性的概念不同于传统云计算三级服务：SaaS（软件即服务），PaaS（平台即服务），IaaS（基础设施即服务）。其含义是集群即服务，即将完整的虚拟集群提供给用户，使得用户可以方便的利用虚拟集群进行大数据的计算和在其上搭建分布式的网络应用。

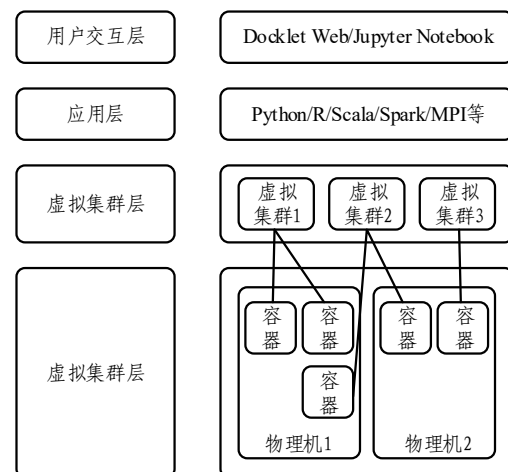


图3 Docklet 架构

Docklet 的架构如上图 3 所示，其核心是基于容器技术和软件定义网络技术的 Linux 容器集群。这些容器集群即形成了用户使用的虚拟集群，Docklet 在原本的 LXC 容器基础上进行了进一步封装，使用户不需要关心容器集群的配置细节，也不需要关心容器网络的构建，对于用户而言，这些容器集群就好像真实的 Linux 集群一样。这些集群支持安装运行各种单机和分布式 Linux 应用，形成个人的云端工作区。Docklet 集成了 Jupyter，使得用户能通过使用浏览器，经由 Jupyter Notebook 访问自己的工作区，进行界面友好的 python 和 R 等语言的编程，还可通过

web terminal 直接操作集群，就如同在那些 Linux 集群上启动了终端进行操作一样。而且，用户还可以仅通过浏览器进行集群管理、镜像管理、集群监控等等操作。

此外，Docklet 是跨云的平台，其底层可以由多个地理位置不同的数据中心所支持。在 Docklet 后端，通常有一个 Super Master 进程负责 Web 页面的渲染和用户请求的分发，它将用户的 Http 请求发往具体的数据中心物理集群进行执行。每个 Docklet 的物理集群中，通常有一个 Master 进程和多个运行在不同物理机上的 Worker 进程。

## 5.2 实现中的问题及解决方案

可以看出，Docklet 平台所面临的使用场景和需求与本文所提出的组网方法是很契合的。这一组网方法可以在 Docklet 平台中自然的实现，但是大规模应用场景下的设计的实现仍存在一些问题。这里简要阐述了一些较为关键的问题及相关的解决方案。

为了支持在线的工作空间，用户的 Http 请求需进入到容器内部的 Jupyter 服务器端。但是由于在 Docklet 平台中的容器只能分配内部 IP 地址，因此需要由反向代理服务器将用户请求代理进入容器内部。不过，在分布式网关的情况下，对于那些网关不在代理服务器所在物理机的容器集群，代理服务器是无法访问那些容器集群内的节点的。而分布式反向代理是这一问题的解决方案。这个解决方案中每个物理机上都存在着反向代理服务器，且将反向代理服务与网关绑定，对于每个用户网关内的容器节点的反向代理服务都由网关所在物理机上的反向代理服务器负责。用户通过访问其网关所在物理机上的反向代理服务器来访问他的容器，从而访问他的云上工作空间。

另一方面，在分布式网关的情况下，不同的反向代理服务器和 Docklet Web 服务器很可能不在同一台物理机上，具有不同的 IP 地址，除非域名解析时将这些 IP 地址放到同一个一级域名

下（但通常这相当麻烦），那这些地址是在不同的域名下的。而由于安全策略的限制，浏览器通常不会传递不同域的 Cookie，这就使得用户认证无法完成。这便是 Cookie 的跨域传递问题。解决这个问题的方案如图 4 所示，在服务器的最前端再架设一级反向代理服务器，提供请求的统一入口，用户请求到达前端反向代理服务器时，再由反向代理服务器将请求分配到具体的后端服务器。

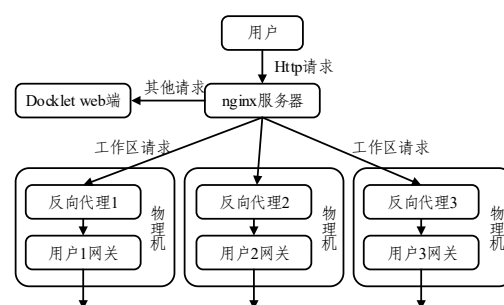


图4 nginx 反向代理下的用户请求处理

## 6 实验与评估

以下实验与评估的测试环境为：6.1 和 6.3 节实验基于两台拥有 400GB 物理内存、24 核 CPU（Xeon E5-2620 v3 @ 2.40GHz）的服务器。6.2 节实验除了包括上述两台服务器，还有两台拥有 128GB 物理内存、56 核 CPU（Xeon E5-2690 v4 @ 2.60GHz）的服务器。所有的 TCP 出口下行带宽测试软件使用的是 speedtest-cli 0.3.4，而内部 TCP 通信带宽测试使用的是 iperf 2.0.5，内部通信往返时延测试使用的是 ping。在 Docklet 中创建的每个容器节点配置为 1 个 CPU 和 4G 内存。

### 6.1 虚拟网络性能损失评估

本实验中，将对 Docklet 中实现的组网方法（大规模应用场景下的设计）相比原生网络的性能损失做测试评估。具体测试方法为（测试结果均取 3 次平均）：

(1) TCP 出口下行带宽测试：分别对原生物理机、实现了大规模应用场景下组网方法设计的 Docklet 的单容器节点（网关在同一物理机）和 Docker 容器（使用默认网络配置）进行 TCP 出

口下行带宽测试。

(2) 内部通信测试：对原生物理机的测试是测试两台物理机的带宽和时延。对 Docklet 中的容器测试则是对单个用户内的两个容器间的通信带宽和时延测试，即两个容器在同一局域网内，但在不同物理机上。对 Docker 容器的测试同 Docklet 容器，使用的网络配置是 overlay。

表 1 网络性能测试

测试节点	出口下行带宽	内部通信带宽	内部通信往返时延
原生物理机	9.86 Mbit/s	943 Mbit/s	0.168 ms
Docklet 容器	9.90 Mbit/s	911 Mbit/s	0.281 ms
Docker 容器	9.88 Mbit/s	911 Mbit/s	0.304 ms

从表 1 的结果可以看出，在 TCP 出口下行带宽上，本文提出的组网方法和 Docker 组网方法的性能相近，这两种方法的出口下行带宽与原生网络的相差在 0.5%以内。而在内部通信带宽上，本文提出的组网方法和 Docker 组网方法的相比原生网络性能损失大约同为 3.5%，且在内部通信的往返时延上，本文提出的组网方法有更小的时延。总的来看，本文提出的组网方法与原生网络性能接近，性能损失较小。

## 6.2 拟真场景下针对两种应用规模设计的带宽比较

本实验中，将模拟真实使用场景，并在这一场景下评估本文所提出的针对两种不同应用规模下的组网方法设计的 TCP 通信带宽。测试环境为四台高性能服务器，都可连外部网，配置如前所述。具体测试方法为：分别创建 1、5、10、20、50、100 个用户，每个用户创建两个容器，且在不同的物理机上。测试 TCP 出口下行带宽时，每个用户的其中一个容器都不断运行测试程序。而测试内部通信带宽时，每个用户的两个容器间不断运行测试程序。每个容器的测试数据取 3 次平均，最后对所有容器的测试数据取平均，得到的测试结果如表 2 所示。

从表 2 可以看出，用户数量比较少时，大规模应用场景下的设计和小规模应用场景下的设计

有比较接近的性能，但前者有着较好的平均带宽。而在用户数量越来越多时，两者的性能差距越来越大，在 4 台出口物理机和 10 个用户以上的情况下，前者比后者的带宽出口下行带宽大约提升了 3 倍。而在 20 个用户的情况下，内部 TCP 通信带宽大约提升了 2 倍。因此，在多用户使用情况下，相比小规模应用场景下的设计，大规模应用场景下的设计无疑有着更好的网络性能，能够给用户更好的用户体验。

表 2 拟真场景下的网络性能比较

用户数量	设计规模	出口下行带宽	内部通信带宽
5	小规模设计	3.47 Mbit/s	307.28 Mbit/s
	大规模设计	6.88 Mbit/s	532.87 Mbit/s
10	小规模设计	1.85 Mbit/s	115.53 Mbit/s
	大规模设计	5.04 Mbit/s	245.73 Mbit/s
20	小规模设计	0.94 Mbit/s	55.26 Mbit/s
	大规模设计	2.52 Mbit/s	166.63 Mbit/s
50	小规模设计	0.33 Mbit/s	*
	大规模设计	1.33 Mbit/s	130.33 Mbit/s
100	小规模设计	0.16 Mbit/s	*
	大规模设计	0.71 Mbit/s	114.44 Mbit/s

\* 由于内部通信量太大，测试工具难以工作。

## 6.3 大量 OVS 网桥和 GRE 隧道下的负载评估

本实验中，将评估在大量 OVS 网桥和 GRE 隧道数下的大规模应用场景下的设计的网络性能和 OVS 管理性能。测试环境为两台高性能服务器，配置如前所述。具体的测试方法为：在两台物理机上分别同时创建 0、100、200、300、400 和 500 个额外的 OVS 网桥和 GRE 隧道（以不同 key 值区分），然后在每次创建完整百个时，测试 OVS 创建一个 OVS 网桥和一个 GRE 端口的时间，以及测试在这种情况下在这两个物理机上的容器（同一虚拟子网内）的 TCP 通信带宽。其测试结果如表 3 所示，测试数据均取 3 次平均。

从表 3 可以看出，即便 OVS 网桥和 GRE 端



口数大量增加，本文所提出的组网方法中的容器内部通信带宽仍几乎不会变化（变化在 0.5% 以内）。但另一方面，创建网桥和端口的时间会增加。总而言之，单个物理机上 OVS 网桥和端口数的增加不会引起网络性能的改变，只会增加 OVS 管理的性能。

表 3 大量 OVS 网桥和 GRE 端口下的网络性能测试

OVS 网桥和 GRE 端口数	创建一个 OVS 网桥的时间	创建一个 GRE 端口的时间	TCP 内部通信带宽
0	0.080 s	0.041 s	911 Mbit/s
100	0.154 s	0.145 s	911 Mbit/s
200	0.316 s	0.202 s	912 Mbit/s
300	0.407 s	0.406 s	910 Mbit/s
400	0.537 s	0.680 s	913 Mbit/s
500	0.963 s	0.881 s	914 Mbit/s

这个实验也就证明大规模应用场景下的组网方法中的第三点局限是不显著的，这是因为网桥和端口数的上升只会引起 OVS 管理性能的改变（而且响应速度的增加尚且在大部分用户接受范围内），不会改变关键的网络性能。

#### 6.4 应用支撑分析实验

本实验中将测试本文提出的组网方法分别对两类应用的支撑情况，这两种应用分别是批处理型应用和长服务型应用。对于批处理型应用来说，其网络负载主要在内部网络上，而对于长服务型应用，其网络负载主要在出口网络上。批处理型应用选取了 MPI 程序作为测试应用，长服务型应用选取了 apache2 服务器作为测试应用。

##### 6.4.1 分布式快排 MPI 程序运行时间实验

本实验对一个分布式快排程序的分布式计算部分进行计时（即不含启动进程和结束进程的时间），该程序对不同数据量的整型数据进行分布式快排，采用二分归并的思想，将快排任务二分的分配给不同的进程节点进行运算，最后将数据合并到 0 号进程。可以分析出该程序的性能瓶颈主要在数据的传输上，因此通过该实验可以看出网络对应用的支撑情况。本实验分别得出了在原始物理机网络和其上的基于本文方案实现的容器网

络中的运行时间，其结果如下表 4 所示，表中结果取了 5 次平均。

从表 4 可以看出，本文提出的容器集群组网方法对于这种应用有着良好的支撑，对于小量的数据排序（1 千万），容器网络中的运行时间与原生网络的差距在 1 秒以内。而对于大量的数据排序（1 亿和 2 亿），容器网络中的运行时间与原生网络的差距在 10% 以内。

表 4 不同网络上的分布式快排 MPI 程序运行时间

数据量	所在网络	运行时间
1 千万	原生网络	1.17 s
	容器网络	1.51 s
1 亿	原生网络	12.06 s
	容器网络	13.31 s
2 亿	原生网络	32.14 s
	容器网络	33.82 s

##### 6.4.2 Apache2 服务器页面响应时间实验

本实验将测试不同网络下的 Apache2 服务器的网页响应时间。Apache 服务器版本为 2.4.18，采用 httping 为测试软件，版本号为 2.4。测试结果如表 5 所示，表中结果取了 10 次平均。

表 5 不同网络上的 Apache2 服务器的页面响应时间

所在网络	Apache2 服务器的页面响应时间
原生网络	31.5 ms
容器网络	37.5 ms

从表 5 的结果可以看出，本文提出的容器集群组网方法的确会降低 Apache2 服务器的性能，但是相比原生网络，其性能损耗在 20% 以内，尚可接受。因此，本文提出的容器集群组网方法对这种应用也有良好的支撑。

**结束语** 本文提出了一种面向多租户的 Linux 容器集群组网方法，并且介绍了在小规模和大规模应用场景下这种方法不同的设计，而且在实际的容器云平台 Docklet 中实现。

这种组网方法改进了 Kubernetes 的组网方法，简化了网络结构，并且引入了用户隔离，使得网络可以为多个用户服务。在小规模应用场景

下,采用集中式网关和共享式网络的设计,用户隔离采用 OVS 提供的 VLAN Tag 技术。而在大规模应用场景下,采用分布式网关和独享式网络的设计。总的来说,理论和实验证明,小规模应用场景下的设计结构更加简单、清晰,同时也有较低的管理负载,但在多用户使用下性能会比较差。而大规模应用场景下的设计结构较为复杂,同时有较高的管理负载,但是在多用户使用下性能更好。

此外,实验证明,这一组网方案所构成的虚拟网络与原生网络的性能相当的接近,其 TCP 出口下行带宽与原生网络相差在 0.5%以内,而 TCP 内部通信带宽大约只损失 3.5%。

未来的工作主要包括两个方面,一是比较和尝试其他 Linux 内核网络虚拟化技术,如 macvlan 和 ipvlan,为改善网络的性能采用更好的工具和技术。二是在多出口的情况下,引入负载均衡算法,通过网关迁移,将用户的网络负载更均衡的分配到不同物理机上,从而提高用户通信的平均带宽,改善用户体验。

### 参考文献

- [1] Felter W, Ferreira A, Rajamony R, et al. An updated performance comparison of virtual machines and linux containers[C]//Performance Analysis of Systems and Software(ISPASS), 2015 IEEE International Symposium On. IEEE, 2015:171-172
- [2] Seo K T, Hwang H, Moon I, et al. Performance comparison analysis of linux container and virtual machine for building cloud[J]. Advanced Science and Technology Letters, 2014, 66:105-111
- [3] Verma A, Pedrosa L, Korupolu M, et al. Large-scale cluster management at Google with Borg[C]//Proceedings of the Tenth European Conference on Computer Systems. ACM, 2015: 18.
- [4] Zhang Fuxing. The implement and framework of zhihu ten-thousand level containers platform[EB/OL].(2016-11-18)[2017-07-11]. <http://www.infoq.com/cn/presentations/platform-architecture-and-combatof-zhihu-container-platform>  
张阜兴. 知乎万级规模容器平台架构和实战[EB/OL].(2016-11-18)[2017-07-11]. <http://www.infoq.com/cn/presentations/platform-architecture-and-combatof-zhihu-container-platform>
- [5] Cui W, Zhan H, Li B, et al. Cluster as a Service: a Container based Cluster Sharing

Approach with multi-user Support[C]//Service-Oriented System Engineering (SOSE), 2016 IEEE Symposium on. IEEE, 2016: 111-118.

- [6] Bernstein D. Containers and cloud: From lxc to docker to kubernetes[J]. IEEE Cloud Computing, 2014, 1(3): 81-84.
- [7] Burns B, Grant B, Oppenheimer D, et al. Borg, omega, and kubernetes[J]. Communications of the ACM, 2016, 59(5): 50-57.
- [8] Marmol V, Jnagal R, Hockin T. Networking in containers and container clusters[J]. Proceedings of netdev 0.1, 2015.
- [9] The Kubernetes Authors. Kubernetes OpenVSwitch GRE/VxLAN networking [EB/OL].[2017-07-08]. <https://kubernetes.io/docs/admin/ovs-networking/>.
- [10] Merkel D. Docker: lightweight linux containers for consistent development and deployment[J]. Linux Journal, 2014, 2014(239): 2.
- [11] Docker Inc. Docker Documentation [EB/OL]. [2017-07-10]. <https://docs.docker.com/>

朱瑜坚, 电话: 18811464128。邮箱: [firmllyzhu@163.com](mailto:firmllyzhu@163.com)