适用于电网调控系统的轻量级容器技术实现

顾雯轩[1,2,3],高 原[1,2,3],顾文杰[1,2,3],邹 庆[1,2,3],董子奇[1,2,3]
(1.南瑞集团有限公司(国网电力科学研究院),江苏 南京 211106;
2.国电南瑞科技股份有限公司,江苏 南京 211106;
3.智能电网保护和运行控制国家重点实验室,江苏 南京 211106)

摘 要:电网调控系统通过广播/组播方式同步应用状态,传统的开源容器网络无法支持,且传统开源容器体积较大,导致应用迁移和启动耗时长,不能直接用于电网实时调控系统。同时开源容器不支持网络带宽限制,不满足实时系统的可靠性要求。通过研究现有容器技术的实现原理,提出一种轻量级的容器技术,能满足电网实时调控系统的数据同步要求。通过研究虚拟网桥技术实现了跨物理宿主机的广播/组播通信;结合Linux流量控制内核技术和控制组内核技术实现容器网络带宽限制;并通过对私有数据的封装降低容器体积,实现轻量化。最后通过测试验证,该轻量级容器在支持组播/广播通信的同时具备带宽限制的功能,且体积小,迁移快速方便,启动性能上优于传统容器。

关键词:轻量级容器;广播/组播;内核技术;带宽限制;虚拟网桥网卡
中图分类号:TP302 **文献标识码**:A **文章编号**:1673-629X(2021)02-0116-06
doi:10.3969/j.issn.1673-629X.2021.02.022

Implementation of Lightweight Container Technology for Power Grid Dispatching and Control System

GU Wen-xuan[1,2,3],GAO Yuan[1,2,3],GU Wen-jie[1,2,3],ZOU Qing[1,2,3],DONG Zi-qi[1,2,3]
(1.NARI Group Corporation (State Grid Electric Power Research Institute),Nanjing 211106,China;
2.NARI Technology Development Co.,Ltd.,Nanjing 211106,China;
3.State Key Laboratory of Smart Grid Protection and Control,Nanjing 211106,China)

Abstract:The power grid dispatching and control system implements broadcast or multicast to synchronize the application state,which is not supported by the network technology of traditional open source container. And the open source container is too large to be directly used in real-time system. At the same time,the open source container does not support the network bandwidth limitation and does not meet the reliability requirements of the real-time system. By studying the implementation principle of the existing container technology,a lightweight container technology is proposed,which can meet the data synchronization requirements of the power grid real-time control system. Through the research of virtual bridge technology,the broadcast/multicast communication across physical host is realized. The bandwidth limitation of container network is realized by combining Linux flow control kernel technology and Control Group technology,and the size of container is reduced by encapsulating private data. Finally,the test shows that the lightweight container has the function of bandwidth limitation while supporting multicast/broadcast communication,and it is small in size,fast and convenient in migration,and better in startup performance than the traditional container.

Key words:lightweight container;broadcast/multicast;kernel technology;bandwidth limitation;virtual network bridge and network card

0 引 言

近年来,容器化技术成为了云生态系统中最受关注的技术之一。容器能够提供资源隔离、资源限制功能,提高了系统的稳定性。容器通过镜像技术对应用运行环境进行封装,通过容器云管理平台可以方便地实现持续集成和持续部署,降低了系统运维的工作量。

随着特高压电网的大规模建设,电网调度控制(简称调控)系统实时数据规模快速增加,整个调控系统中的应用数量也大量增加,对系统基础资源管理能力和运维能力提出了更高要求。以往调控系统应用部署时不能有效限制应用使用的资源[1]。引入容器技术后,可以实现更细粒度的资源分配。通过容器的资源

收稿日期:2020-03-21 **修回日期**:2020-07-22
基金项目:国家重点研发计划项目(2017YFB0902600);国家电网公司科技项目(5700-201955450A)
作者简介:顾雯轩(1987-),女,硕士,工程师,研究方向为容器、网络通信、任务调度、云计算等。

限制和资源隔离功能,将业务服务限制在分配的资源范围内运行,在保证了系统资源充分利用的同时实现系统稳定运行。

但是,传统的开源容器[2-4]技术不能完全满足电网实时调控系统的要求。比如容器网络不支持广播,不具备限制网络带宽的功能,容器镜像较大导致应用迁移和启动耗时长。通过研究现有容器技术的实现原理,该文提出一种轻量级的容器服务技术,通过研究虚拟网桥技术实现了跨物理宿主机的广播/组播通信;结合 Linux 流量控制内核技术实现容器网络带宽限制,并通过对私有数据的封装降低容器体积,实现轻量化。

1　现有开源容器技术

1.1　资源限制和资源隔离

容器的实现主要依赖于 Linux 的内核技术 CGroup[5], NameSpace[6-7] 和联合文件系统[8]。CGroup(Control Group 控制组)是 Linux 内核提供的一种限制进程组使用宿主机物理资源的机制。通过给容器分配资源的控制机制,避免某个异常容器占用过多的宿主机资源,确保同时运行的多个容器对宿主机资源的合理利用。控制组可以提供内存、CPU、磁盘 IO 等资源的限制和查询管理,但是 net_cls 子系统不具备直接限制网络资源的功能,即不能限制某个容器内应用占用宿主机的带宽大小。

NameSpace(命名空间)是 Linux 针对实现容器资源隔离而引入的特性。每个容器都可以拥有自己唯一的命名空间,运行在其中的进程都像是在独立互不干扰的操作系统中运行一样,彼此不可见。这个机制保证了容器运行环境的相互独立,特别适合一个宿主机运行很多微服务的应用场景,能够保证每个服务间使用操作系统资源以及服务自身数据的互不干扰。

1.2　容器网络

开源容器 Docker 目前常用的三种网络插件为 Flannel[9]、Weave[10]、Calico[11]。Flannel 通过给每台宿主机分配一个子网的方式为容器提供虚拟网络,使用 UDP/VXLAN 封装 IP 包来创建 overlay 网络,并借助 etcd 维护网络的分配情况。Weave 创建一个连接多个 Docker 主机的虚拟网络,类似于一个以太网交换机,所有的容器都连接到这上面,互相通信。Calico 在主机上创建了一堆的虚拟网卡,其中一端在主机上,另一端在容器的网络命名空间里,然后在容器和主机中分别设置几条路由,来完成网络的互联。

这三种插件都是通过给容器建立子网,给容器分配子网 IP 的方式,使得容器间可以互通。容器分配的 IP 与宿主机不是一个网段,跨宿主机的容器间可以进行 TCP 和 UDP 通信。在一个宿主机内部,容器之间可以进行组播/广播通信,但是跨宿主机节点的容器之间无法进行组播/广播通信。

1.3　容器镜像

联合文件系统的引入为构建容器和快速迁移提供了有效机制。

一个典型的 Linux 系统运行需要两个文件系统:bootfs 和 rootfs。相同内核版本的不同 Linux 发行版本的 bootfs 都是一致的,并且是用户不可更改的。另一个 rootfs 文件系统包含典型的目录,包括/dev,/proc,/bin 等再加上一些需要运行用户程序的必要文件(配置文件,库文件,可执行文件)。这个文件系统在不同的 Linux 发型版本中是不同的,并且是用户可更改的。

当启动一个容器的时候,会将镜像[12-13]的内容加载到容器内部。镜像中的内容主要是 rootfs 文件系统以及一些用户程序依赖文件。从同一个镜像启动多个容器时,rootfs 文件系统会被加载重复加载多次,当在宿主机上启动大量容器时,相同 rootfs 文件的重复加载对内存资源是一种浪费,并且对容器的启动时间也有一定的影响。

一种最大化内存共享与最小化运行时环境的超轻量级容器[14]通过去除所有底层操作系统库文件且单独指定容器内应用依赖库的方法构建轻量化容器。这种方法是一种量身定制的方法,但是如果在电力系统中使用,需要每个应用都能准确指定所需要的依赖库,这对电力系统的研发人员具有一定挑战性,且通用性不好,实施难度大。

2　轻量级容器架构

该文提出一种轻量级容器的实现方法,将容器运行时的存储空间分为两部分:公共存储空间和私有存储空间。其中,公共存储空间共享宿主机的文件资源,私有存储空间存储容器应用或服务运行需要的相关文件资源。各个容器之间的私有存储空间相互隔离,互不干扰,并且容器之间共享宿主机的文件资源;在容器内部,可以同时看到宿主机文件资源和自身的文件资源。

每个轻量级容器拥有独立的工作目录,工作目录作为私有存储空间是相互独立的,容器只能访问自己的工作目录。各个轻量级容器在自身容器内部观察到的工作目录的路径和命名是一致的,这样保证了电力上层应用程序的通用性,即容器对应用程序是透明的。通过 Linux NameSpace 中的 mnt[15]挂载命名空间实现数据隔离,这样对某个轻量级容器的私有工作目录进行操作不会影响到宿主机或其他轻量级容器的相同目录。各个轻量级容器的工作目录在宿主机上基于不同的路径和目录进行管理,目录命名与容器命名相关联,

轻量级容器进行迁移打包的时候,只需要打包各自的工作目录即可,因此容器镜像的体积更小,整个镜像封装更加轻便。

轻量级容器结构如图 1 所示。

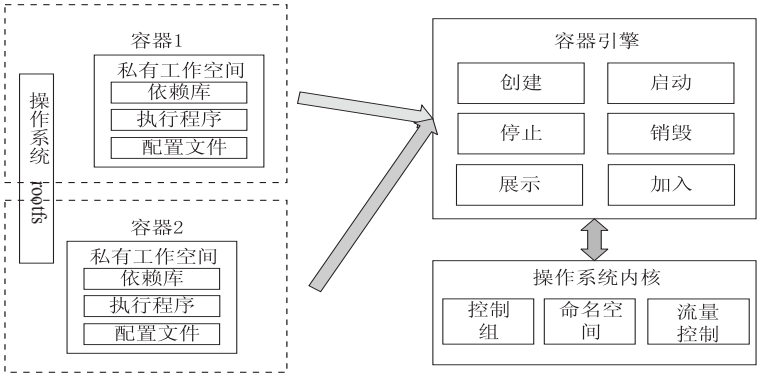


图 1 轻量级容器结构

操作系统 rootfs:宿主机文件系统包含 bootfs 和 rootfs。其中 bootfs 和 Linux 内核相关,所有容器共享。rootfs 包括常见的/dev,/proc,/etc,/usr 等,所有容器也共享 rootfs,在容器中看到就是宿主机的 rootfs。

轻量级容器:从存储空间来说包含两部分:容器中运行的可执行程序以及需要的依赖库,配置文件。这些都是存放在容器的私有工作目录中的,各个容器之间相互看不见;共享宿主机 rootfs 文件系统,这个为公共存储空间。

Linux 操作系统内核:宿主机的操作系统内核,所有运行在该宿主机上的容器都共享 Linux 操作系统内核。容器的资源限制,资源隔离以及流量控制功能都需要操作系统内核的支持。

轻量级容器引擎:负责管理容器的整个生命周期,包括创建,启动,停止,销毁,主要功能的实现都是依赖 Linux 内核技术,包括控制组和命名空间。

3 轻量级容器架构

以上通用技术实现的容器还不能直接满足电力实时调控系统的需求,原因是电力调控系统对可靠性和实时性要求非常高,所以整个系统底层建立了提供高可用管理和高速通信功能的基础平台,这个平台的高可用和通信模块需要使用广播和组播进行通信。并且使用容器后,在一个物理机上不能发生某一个容器抢占了这个物理机所有带宽的情况,否则实时系统的部分功能模块可能就会失去作用。同时因为系统实时性要求高,容器启动的时效性就要很高的要求,启动时间越短越好。因此该文基于以下讨论的三种技术实现了一种支持跨物理节点组播/广播,并能提供网络带宽限制功能的轻量化容器。

3.1 网络带宽限制技术

网络带宽限制技术是通过控制组(Control CGroup)的 net_ctl[16] 子系统和 Linux 的流量控制(Traffic Control)[16] 共同配合实现的。CGroup 子系统 net_cls 可以给发送 packet 打上 classid 的标签,用于过滤分类。有了这个标签,流量控制器(tc)可以对分属不同的 CGroup 进程发送的 packet 起作用。每个容器拥有独立的网络命名空间,且为容器内网卡配置的 IP 地址与宿主机的 IP 地址在一个网段上。如图 2 所示,为了对该容器的发送数据进行网络流量限制,需要在容器内的网卡上设置 TC 规则,其中 eth1 是容器内的网卡。然后在该容器的 CGroup 子系统 net_cls 下的相应目录内,将 tc class 规则的 classid 传给 net_cls.classid 文件。通过这样的操作,该容器内所有的输出网络流量都会限制在 tc class 规则限定的 value 值内,其中,value 为字符串,格式为数值+单位(kbps/mbps/kbit/mbit)。

tc qdisc add dev eth1 root handle major:0 htb

tc class replace dev eth1 parent major: classid major: minor htb rate value

tc filter add dev eth1 protocol ip parent major:0 prio 1 handle major: minor cgroup

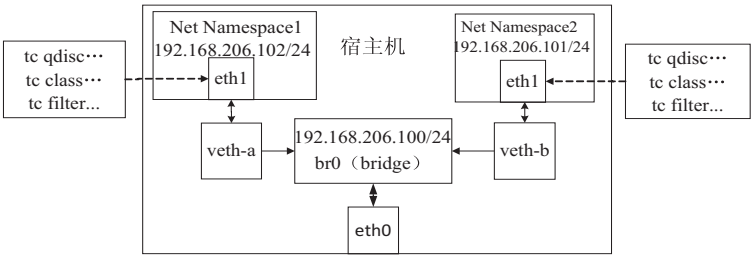


图 2 网络带宽限制原理

3.2　支持组播广播的容器通信技术

电网调控系统目前多采用多机主备冗余机制,该机制使用广播进行通信。调控系统的消息总线使用组播进行通信。因此,若需要轻量级容器运用到电网调控系统中,必须确保各个容器之间能够进行组播/广播通信,并且容器和物理节点之间也能对等地进行组播/广播通信。文中轻量级容器分配 IP 的方式结合虚拟网桥和虚拟网卡技术实现。

工作原理如图 3 所示。

(1)宿主机建立虚拟网桥 br0,将物理网卡 eth0 的 ip 地址赋予 br0,桥接到 br0;

(2)创建一对虚拟网卡 veth pair,其中一个虚拟网卡加入容器的网络命名空间中,并更名为 eth1,配置 IP 地址和路由。另一个网卡桥接到 br0 上。

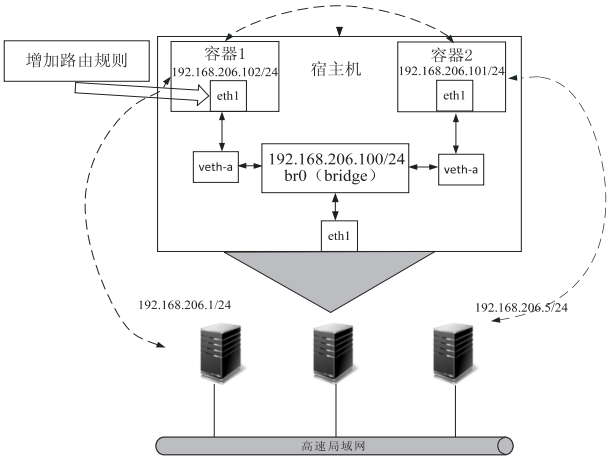
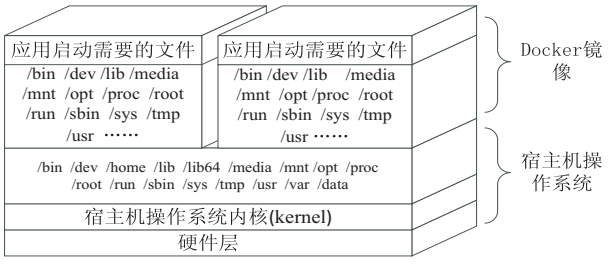


图 3　虚拟网桥/网卡实现原理

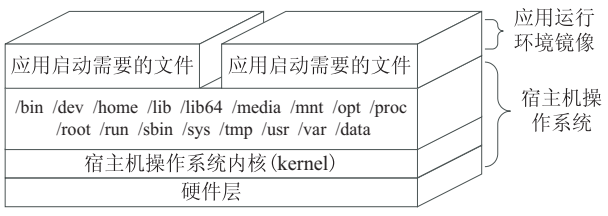
如图 3 所示,每个容器拥有独立的 IP 地址,与宿主机配置在同一网段上,容器和容器,容器和宿主机之间都能进行广播和组播通信。

3.3　容器轻量化技术

轻量化主要体现在相比 Docker 容器的镜像(如图 4(a)所示)体积更小,层级更少。如图 4(b)所示,轻量级容器内部虽然可以看到整个 Linux 的文件操作系统,但并不表示轻量级容器镜像包含 Linux 文件操作系统的整个信息。镜像中只存放了支持容器运行的一些必备文件,包括依赖库文件,程序执行文件和配置文件。创建容器的时候,会在宿主机上创建一个目录用于存放容器的私有数据,并将镜像中的文件拷贝至容器私有目录中。然后通过 mount 的 bind 操作将这个私有目录挂载至容器的相应目录中。对于不同的容器来说,将私有目录挂载至容器挂载空间的相同目录中,由于容器拥有独立的挂载隔离空间,这种操作是独立的,互不干扰的。不同容器的相同挂载目录中的内容是不一样的,在此目录中做文件修改,文件删除或文件增加操作都只限于本容器内部,不会对其他容器的挂载目录产生任何影响,同时对容器外面的宿主机而言也是不可见的。



(a)传统容器镜像结构



(b)轻量级容器镜像结构

图 4　容器镜像结构

4　性能分析

本节通过在网络性能、带宽限制能力、启动时间三方面将该文研究的轻量级容器与开源容器进行了对比,说明了轻量级容器在三个方面均优于开源容器。

4.1　带宽限制

结合控制组的 net_cls 子系统和 Linux 流量控制功能,可以实现对容器发送数据的带宽限制。通过 scp 跨主机拷贝大容量文本 Text 进行测试验证,测试分两组进行(见图 5):

第一组在主机 docker3 上直接通过 scp 拷贝 Text 文本至主机节点 docker1,通过 iftop 检测网卡,发送速率约为 800 Mb/s。

第二组测试是在轻量级容器中进行。首先启动容器时通过参数设置,对该容器的带宽值限制在 20 Mb/s。然后通过 scp 拷贝和第一组中相同大小的文件 Text 文本至主机节点 docker1,通过 iftop 检测,发送速

率控制在了 20 Mb/s 以内。

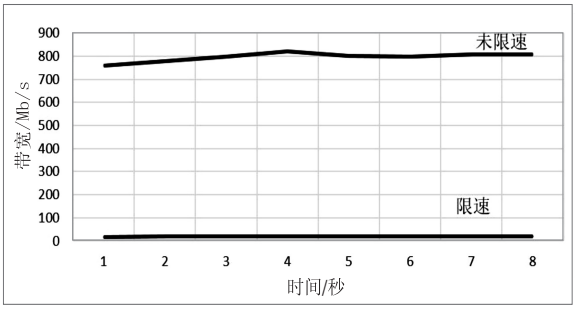
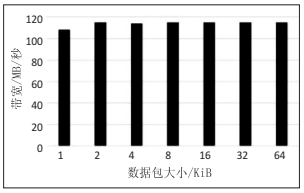


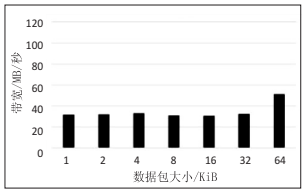
图 5 带宽使用对比

4.2 网络通信性能对比

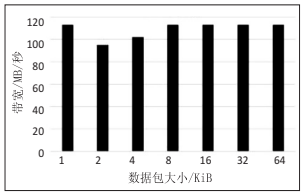
本节对目前 Docker 主流的网络开源组件和文中所述的轻量级容器网络组件进行性能对比。对比的方面包括带宽和时延,通过发送不同 KiB 大小的数据包镜像检测。采用测试 5 次取平均值的方法。



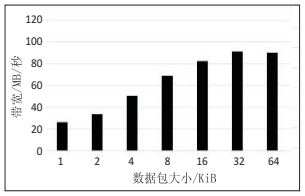
(a) host 宿主机带宽



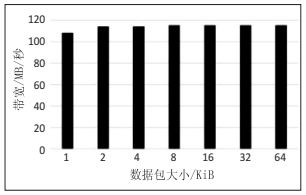
(b) Flannel 带宽



(c) Calico 带宽



(d) Weave 带宽

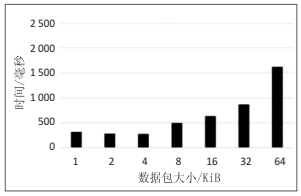


(e) 网桥模式带宽

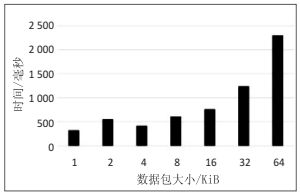
图 6 带宽性能测试

带宽性能检测结果如图 6 所示,图中的横坐标表示发送数据包的大小,从 1 到 64 递增,单位为 KiB;纵坐标为带宽,单位为 MB/sec。从图中对比可知,Flannel 和 Weave 的带宽性能较差,Calico 和网桥方式与主机性能差不多,其中网桥方式性能最优。

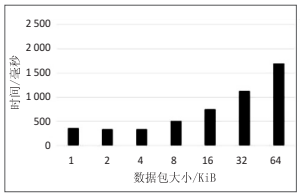
时延性能测试结果如图 7 所示,图中的横坐标表示发送数据包的大小,从 1 到 64 递增,单位为 KiB;纵



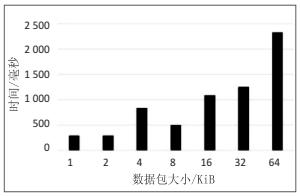
(a) host 宿主机时延



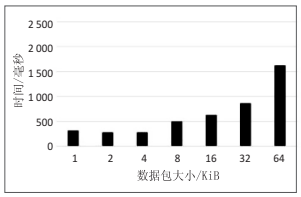
(b) Flannel 时延



(c) Calico 时延



(d) Weave 时延



(e) 网桥模式时延

图 7 时延性能测试

坐标表示时延,单位是 μs。从对比中可知,Calico 和网桥方式与主机性能差不多,且网桥方式性能最优,Weave 和 Flannel 会随着数据包越来越大而时延越来越大。

4.3 启动时间性能对比

这里从容器中的进程启动时间来对比分析一下文中的轻量级容器和 Docker 容器的性能。

测试中使用的 rocky_cloud:latest 开源 docker 容器包含了运行电网调控系统所需的最小化国产操作系统模块,轻量级容器因为与底层宿主机操作系统共享模块,所以大小要小很多。采用测试 5 次取平均值的方法,轻量级容器和 Docker 中只启动一个测试程序,见表 1。测试程序单独启动时需要 1 022 ms。

表 1 镜像大小对比

容器名	镜像名	镜像大小/Mb
Docker	rocky_cloud:latest	253
轻量级容器	测试镜像	10

通过轻量级容器启动测试程序需要 1 025 ms,Docker 启动该测试程序需要 2 133 ms。图 8 中,横坐标标识同时启动容器的个数,分别测试了同时启动 2 个容器,4 个容器,8 个容器,16 个容器,32 个容器时。纵坐标标识容器启动需要花费的时间,单位是 μs。从图 8 中分析可知,基本上用 Docker 启动进程需要花费的时间是轻量级容器的 2 倍。

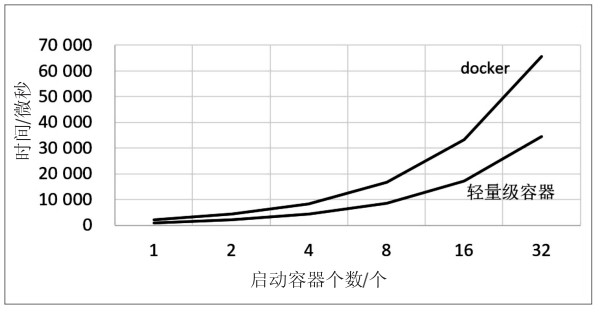


图 8 启动时间性能对比

5　结束语

提出了一种适用于电网调控实时系统的轻量级容器技术实现方法,在网络带宽限制,跨物理机通信方式和轻量化方面都进行了相关工作。主要的工作包括:结合 Linux 内核技术实现容器网络带宽的限制功能;结合虚拟网桥网卡技术,为容器分配独立的 IP 地址并支持跨主机的组播广播通信方式;提出了轻量级容器私有工作空间的概念,通过共享 Linux 文件操作系统的方式实现容器的轻量化。最后通过实验结果分析,该轻量级容器在支持组播/广播通信的同时具备带宽限制的功能,且体积小,迁移快速方便,启动性能上优于传统容器。

参考文献:

[1] 高　原,张　勇,宁　剑,等. 适用于电网调控系统的细粒度多机冗余机制的设计与实现[J]. 计算技术与自动化,2019,38(4):25-30.

[2] SENTHIL K S. Introduction to Linux containers[M]//Practical LXC and LXD. Berkeley:Apress,2017.

[3] BOETTIGER C. An introduction to Docker for reproducible research[J]. Operating Systems Review,2015,49(1):71-79.

[4] 卫　彪,刘成龙,郭　旭. 深入浅出 Docker 轻量级虚拟化[J]. 电子技术与软件工程,2016,23(10):252-252.

[5] 段　赫. 基于 LXC 容器资源优化的研究与实现[D]. 广州:华南理工大学,2016.

[6] 闫健勇, 龚　正, 吴治辉, 等. Kubernetes 权威指南:从 Docker 到 Kubernetes 实践全接触[M]. 北京:电子工业出版社,2017.

[7] 刘思尧,李　强,李　斌. 基于 Docker 技术的容器隔离性研究[J]. 软件,2015,36(4):110-113.

[8] 李　伟. 基于 Docker 的镜像组合技术研究与实现[D]. 广州:华南理工大学,2017.

[9] 何松林. 基于 Docker 的资源预调度策略构建弹性集群的研究[D]. 杭州:浙江理工大学,2016.

[10] 杜　军. Kubernetes 网络权威指南:基础、原理与实践[M]. 北京:电子工业出版社,2019.

[11] 毕小红,刘　渊,陈　飞. 微服务应用平台的网络性能研究与优化[J]. 计算机工程,2018,44(5):53-59.

[12] 华为 Docker 实践小组. Docker 进阶与实战[M]. 北京:机械工业出版社,2016.

[13] 吉　晨,石　勇,戴　明,等. 基于轻量级虚拟化环境的可信多级安全容器机制[J]. 计算机应用研究,2017,34(6):1770-1773.

[14] 张礼庆,郭　栋,吴绍岭,等. 一种最大化内存共享与最小化运行时环境的超轻量级容器[J]. 计算机研究与发展,2019,56(7):1545-1555.

[15] 李　荣. 系统管理员必读的容器入门指南[J]. 计算机与网络,2019,24(17):40-41.

[16] 王志伟,杨　超. 基于流量控制的 Docker 容器网络带宽控制机制[J]. 计算机应用,2019,39(12):3628-3632.

(上接第 115 页)

[J]. 计算机工程与设计,2017,38(9):2573-2579.

[5] 赵子晨,朱志祥,蒋来好. 构建基于 Dubbo 框架的 Spring Boot 微服务[J]. 计算机与数字工程,2018,46(12):2539-2543.

[6] 武海龙,李国平. 基于分布式架构管理的 B2C 商城设计与实现[J]. 电脑与信息技术,2019,27(4):25-28.

[7] 童二宝,彭战军. 基于分布式集群技术的 SSM 购物商城系统设计[J]. 软件,2019,40(8):123-126.

[8] 宋万洋. 基于 Dubbo 框架的分布式视频网站架构设计[J]. 软件导刊,2018,17(8):137-140.

[9] 王　悦,张　雷,钱英军. 基于 SpringBoot 微服务的 Spring Security 身份认证机制研究[J]. 电脑编程技巧与维护,2019,18(8):64-65.

[10] 张佳辰,刘晓光,王　刚. 多种存储环境下压缩数据库的缓存优化[J]. 计算机应用,2018,38(5):1404-1409.

[11] 李富合,高东林,曹宁生. 基于 RESTful 的中间件服务化体系结构及关键技术研究[J]. 舰船电子工程,2019,39(7):113-118.

[12] 黄强文,曾　丹. 基于 Spring Cloud 和 Docker 的分布式微服务架构设计[J]. 微型电脑应用,2019,35(6):98-101.

[13] 付琳琳,邹素雯. 微服务容器化部署的研究[J]. 计算技术与自动化,2019,38(4):151-155.

[14] 方　意,朱永强,宫学庆. 微服务架构下的分布式事务处理[J]. 计算机应用与软件,2019,36(1):152-158.

[15] 曹文彬,谭新明,刘　备,等. 基于事件驱动的高性能 WebSocket 服务器的设计与实现[J]. 计算机应用与软件,2018,35(1):21-27.