Journal of Computer Applications
计算机应用,2018,38(2):557-562
ISSN 1001-9081
CODEN JYIIDU
2018-02-10
http://www.joca.cn

文章编号:1001-9081(2018)02-0557-06
DOI:10.11772/j.issn.1001-9081.2017081943

基于 Docker 的云资源弹性调度策略

彭丽苹,吕晓丹*,蒋朝惠,彭成辉
(贵州大学 计算机科学与技术学院,贵阳 550025)
(* 通信作者电子邮箱 18984170078@126.com)

摘　要:针对云资源弹性调度问题,结合 Ceph 数据存储的特点,提出一种基于 Docker 容器的云资源弹性调度策略。首先,指出 Docker 容器数据卷不能跨主机的特性给应用在线迁移带来了困难,并对 Ceph 集群的数据存储方法进行改进;然后,建立了一个基于节点综合负载的资源调度优化模型;最后,将 Ceph 集群和 Docker 容器的特点相结合,利用 Docker Swarm 实现了既考虑数据存储、又考虑集群负载的应用容器部署算法和应用在线迁移算法。实验结果表明,与一些调度策略相比,该调度策略对集群资源进行了更细粒度的划分,实现了云平台资源的弹性调度,并在保证应用性能的同时,达到了合理利用云平台资源和降低数据中心运营成本的目的。

关键词:弹性调度;应用迁移;数据放置策略;Swarm 编排器;Docker 容器;Ceph 集群
中图分类号:TP393.027　　**文献标志码**:A

Elastic scheduling strategy for cloud resource based on Docker

PENG Liping, LYU Xiaodan*, JIANG Chaohui, PENG Chenghui
(*College of Computer Science and Technology, Guizhou University, Guiyang Guizhou* 550025, *China*)

Abstract: Considering the problem of elastic scheduling for cloud resources and the characteristics of Ceph data storage, a cloud resource elastic scheduling strategy based on Docker container was proposed. First of all, it was pointed out that the Docker container data volumes are unable to work across different hosts, which brings difficulty to apply online migration, then the data storage method of Ceph cluster was improved. Furthermore, a resource scheduling optimization model based on the comprehensive load of nodes was established. Finally, by combining the characteristics of Ceph cluster and Docker container, the Docker Swarm orchestration was used to achieve container deployment and application online migration in consideration of both data storage and cluster load. The experimental results show that compared with some scheduling strategies, the proposed scheduling strategy achieves elastic scheduling of the cloud platform resources by making a more granular partitioning of the cluster resources, makes a reasonable utilization of the cloud platform resources and reduces the cost of data center operations under the premise of ensuring the application performance.

Key words: elastic scheduling; application migration; data-place policy; Docker Swarm; Docker container; Ceph cluster

0　引言

云计算技术迅速发展,基于云平台的应用也层出不穷。云平台通过虚拟化技术将计算机资源整合成资源池,以按需付费的方式实现了用户对计算资源的弹性需求[1]。云计算发展至今,虚拟化技术一直是云平台中的关键技术,而容器技术则是近年来兴起的一种虚拟化技术[2],它的出现给传统虚拟化技术带来了挑战,为构建高效的云平台提供了新思路[3-6]。在 Docker、Linux Container(LXC)、Warden、OpenVZ 等众多容器技术中,人们最为看好的是 Docker 容器技术[7],阿里巴巴、京东、亚马逊、谷歌、微软等国内外大型运营商已经建立了基于 Docker 容器技术的云平台。Docker 容器技术是一种操作系统层虚拟化技术,与传统的虚拟机技术不同,它不需要运行客户机操作系统,容器以进程的形式运行在宿主机操作系统中,这也使得容器具有比传统虚拟机更轻便、灵活、快速部署的优点[8]。另外,Docker 利用 Union FS 技术,采用分层存储架构,实现了计算资源的复用,大大提高了计算机资源利用率[9]。武志学[10]详细介绍了 Docker 技术,并将 Docker 和虚拟机两种虚拟化技术进行了对比,最后指出 Docker 技术将会成为云计算的核心技术。事实表明他这种说法是正确的,现今各大云计算运营商正在大量地构建基于 Docker 容器技术的云平台。云平台资源的弹性调度问题一直是云计算发展过程中的热点问题,而应用在线迁移技术是实现云平台弹性扩展的重要技术,因此,研究 Docker 云平台应用的在线迁移技术,进而实现云平台资源的弹性调度具有重大意义。

存储系统是云计算数据中心最主要的资源,因此如何合理运用云平台存储资源就显得尤为重要。Ceph[11]是一个开源的软件定义分布式文件存储系统,被广泛用于构建大型云平台的后端存储系统[12]。它不仅支持块存储、文件存储和对

收稿日期:2017-08-09;**修回日期**:2017-09-23。　**基金项目**:贵州省基础研究重大项目(黔科合 JZ 字【2014】2001-21)。
作者简介:彭丽苹(1990—),女,湖南郴州人,硕士研究生,主要研究方向:云计算、大数据;　吕晓丹(1970—),男,上海人,副教授,硕士,主要研究方向:云计算、算法设计、数据分析;　蒋朝惠(1965—),男,四川广安人,教授,硕士,主要研究方向:云计算、信息安全、数据库、软件工程;彭成辉(1990—),男,湖南娄底人,硕士研究生,主要研究方向:云计算、数据挖掘。

象存储三种存储模式,还具有低成本、高可扩展性、高可靠性等优点,但 Ceph 的数据副本存储算法——CRUSH 算法[13] 在合理利用资源方面存在着不足,沈良好等[14] 就从节约系统能耗的角度对这一问题进行了阐述。CRUSH 算法是一种伪随机 Hash 散列算法,总是尽可能地将数据平均存储到 Ceph 集群的对象存储设备(Object-based Storage Device, OSD)中,使得除了在应用高峰期以外,集群中大多数点都处于低负载状态,这就造成了资源的浪费,并增加了系统能耗[15]。

针对上述问题,结合 Ceph 和 Docker 容器的特点,提出了一种基于 Docker 的云平台资源弹性调度策略。首先对前人的工作进行总结,并指出其中的不足;然后对 Ceph 集群的数据副本存储策略进行改进,并建立了一个资源调度优化模型;在此基础之上,提出了基于 Docker 云平台的应用容器部署算法和应用在线迁移算法;最后,在综合考虑数据存储、资源负载和应用服务性能的基础上,利用 Docker Swarm 容器编排工具,实现了应用容器的动态部署和应用的在线迁移,从而达到了对云资源进行弹性调度的目的。

1 相关工作

针对云平台资源调度问题,学者们做了大量研究。Kang 等[16] 针对 Docker 云平台数据中心的能耗问题,提出了一种负载感知的异构容器集群能效管理方法。该方法利用 K-medoid算法对云平台应用进行分类,并在此基础上建立了一个异构集群的能效模型,最后用不同类型的应用进行实验测试,表明该调度方法能在保证服务性能的情况下节约数据中心能耗,降低云平台运营成本。Meng 等[17] 针对容器应用在不同阶段所需计算资源不同所造成的资源浪费以及集群动态扩展带来的性能损失问题,提出了一种基于时序分析的资源预测模型。该模型根据历史负载数据对容器下一时隙所需的计算资源进行预测,然后根据预测值利用容器技术实现应用的快速弹性收缩,达到了合理部署云平台资源的目的。Guan 等[18] 提出了一种基于 Docker 容器的面向应用的数据中心资源分配方法。该方法针对不同应用的资源使用情况,综合考虑物理机资源和容器资源,建立了一个资源优化模型,并考虑到 Docker 容器资源分层复用的特点,实现了对数据中心资源的合理调度。何松林[19] 利用 Docker 虚拟化技术构建了一个可弹性扩展的弹性集群,并针对集群的资源使用率问题提出了基于容器和集群节点负载数据的调度策略、弹性收缩容器时的宿主机调度策略和改善用户响应延迟的负载预测调度策略。

以上学者们提出的调度策略虽然在一定程度上提高了数据中心资源的利用率,但他们要么只考虑容器部署时的资源调度问题,要么只考虑了容器应用迁移过程中的资源调度问题,都没有考虑到集群应用数据存储对容器部署和应用迁移造成的影响。本文从应用容器部署和应用迁移的角度出发,在同时考虑数据存储和集群节点负载并保证服务性能的前提下,实现了云平台资源的弹性调度。

2 问题描述与建模

2.1 改进的 Ceph 集群数据存储策略

在基于 Ceph 集群的 Docker 容器云平台中,假设 Ceph 集群采用三个副本的数据存储策略,集群中的节点分为 A 区和 B 区,A 区用于应用平稳期的业务需求,B 区用于应用高峰期的业务需求。其中:A 区有 $2R$ 个机架,编号为 $rack_i(0 \leqslant i < 2R, i \in \mathbf{Z})$;$B$ 区有 R 个机架,编号为 $rack_j(0 \leqslant j < R, j \in \mathbf{Z})$。假定每个机架中有 h 台服务器,每台服务器用 $host_k$ $(0 \leqslant k < h, k \in \mathbf{Z})$ 表示,每台服务器中有 N 个 OSD,编号为 $osd_n(0 \leqslant n < N, n \in \mathbf{Z})$,则数据中心节点的目录树结构如图 1 所示。由图 1 可知,$zone.\ rack_i.\ host_k.\ osd_n$ 可定位到集群中一个具体节点中的 OSD,即表示的数据中心 Zone(Zone = A 或 B) 区的第 $i+1$ 台机架中,第 $k+1$ 台服务器中编号为 n 的 OSD。

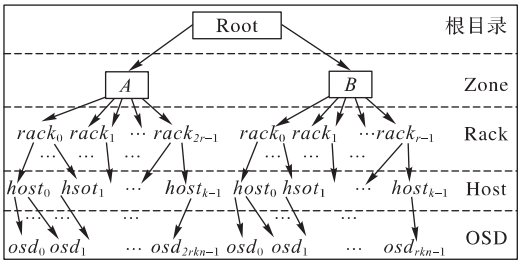

图 1 数据中心目录树结构

Fig. 1 Directory tree structure of datacenter

假定一个 Ceph 客户端数据被封装成对象后映射到 Q 个放置组(Placement Group, PG)中,PG 编号为 $pg_p(0 \leqslant p < Q, p \in \mathbf{Z})$,用 $M_i = \sum_{p=0}^{Q} pg_p\ (i = 1,2,3)$ 表示由用户所有 PG 组成的第 i 个完整数据副本,其中 M_1 所在节点的 OSD 为主 OSD。将 Ceph 集群的隔离域设置成机架(Rack),同时将 M_1、M_2、M_3 分别存放在三个不同机架的服务器中,其中包括 A 区的服务器两台、B 区的服务器一台。另外,为了保证数据的可用性并减少 OSD 数据同步产生的延迟,将 A 区机架的邻域设为 $\delta(0 \leqslant \delta < 2R, \delta \in \mathbf{Z})$,即若 A 区的 $rack_i$ 中存放了某个客户的第一个数据副本,则客户的另一个数据副本存放的机架必然为 $rack_j = \{rack_j / \lceil \max(0, i-\varphi) \leqslant j \leqslant \min(i+\varphi, 2R-1), j \in \mathbf{Z} \text{ 且 } j \neq i\}$。改进的容器云平台架构及数据存储方式如图 2 所示。图中虚线箭头所指处表示该集群节点存储了数据,用粗斜体标出;实线箭头更加细化地表示了节点 $host_0$ 中 OSD 的数据存储情况,存储了应用数据的 OSD 用粗斜体标出,以便与其他未存储数据的 OSD 区分开来。所有的服务器 host 既是 Ceph 集群中的 node 节点也是 Swarm 集群中的 Docker 节点。

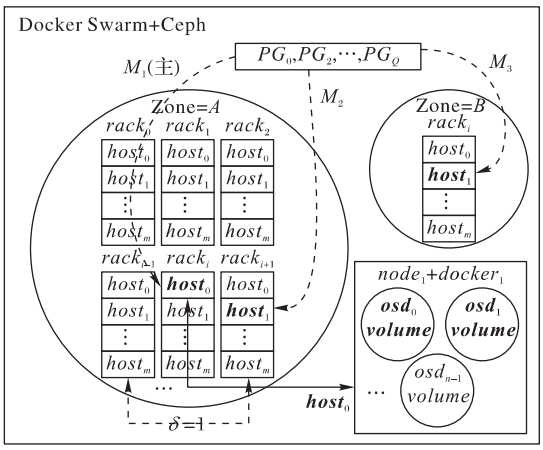

图 2 改进的 Docker 云平台架构及数据存储策略

Fig. 2 Architecture and data storage strategy of improved Docker cloud platform

2.2　系统建模

基于 2.1 节改进了数据存储策略的 Ceph 集群和各个集群节点的资源负载情况,建立了一个资源调度优化模型,具体如下:

$$W_i^{*\,t} = \frac{1}{3}(W_{\mathrm{m}} + W_{\mathrm{c}} + W_{\mathrm{n}}); \tag{1}$$

$$W_i = \lambda W_{\mathrm{m}} + \alpha W_{\mathrm{c}} + \beta W_{\mathrm{n}}; \tag{2}$$

$$\lambda + \alpha + \beta = 1; \tag{3}$$

$$color = \begin{cases} \text{green}, & 0 \leqslant W_i \leqslant 20\% \\ \text{blue}, & 20\% < W_i \leqslant 80\% \\ \text{red}, & 80\% < W_i \leqslant 100\% \end{cases} \tag{4}$$

其中: $W_i^{*\,t}$ 表示部署应用容器 i 后节点 $host_t$ 的综合负载; W_i 表示某节点针对某个应用 i 的综合负载使用情况; W_{m}、W_{c}、W_{n} 分别表示一个集群节点的内存、CPU 和网络使用率; 由于不同类型的应用对资源的需求情况不一样,同一个集群节点,对于 I/O 密集型应用可能已经处于高负载状态,但对于 CPU 密集型应用或网络密集型应用就处于低负载状态,因此引入参数 λ、α、β,针对不同类型的应用 i 对 W_i 进行调整; $color \in \{\text{green}, \text{blue}, \text{red}\}$ 用来标记集群节点针对某个应用 i 的状态, $color$ = green 表示该节点处于空闲状态, $color$ = blue 表示该节点处于低负载状态, $color$ = red表示该节点处于高负载状态。

3　调度策略

3.1　应用容器部署算法

基于第 2 章的内容,以下将给出一种既考虑数据存储、又考虑节点负载情况的应用容器部署方法。在云平台采集内存、CPU 和网络利用率的具体数据,假设采集数据的时间间隔为 S, 一共采集 D 次,对采集的数据进行整理和计算,求得各种资源使用率的平均值即为 W_{m}、W_{c}、W_{n} 的值。对将要部署的应用进行类型判断,看该应用对内存、CPU 和网络带宽的需求情况,分别设置参数 λ、α、β 的比例。例如对于 CPU 密集型应用可设置 λ: α: β = 3: 5: 2,则由式(3) 可以计算出 λ、α、β 的具体值,再结合式(2) 便可计算出 W_i 的值。在得到 W_i 的值后,便可根据式(4) 将集群中的节点进行分类,尽量将应用部署在 $color$ = blue 的节点上,为了保证应用性能,可在同一个主机上运行多个应用 i 的容器。将 $color$ = green 的节点在 Docker Swarm 中的状态设置为 drain,这样 Swarm 便不会将容器应用部署到该节点上,之后将该类节点设置为休眠状态,以节约数据中心资源。关于 $color$ = red 的集群节点,将不会部署新的应用到该类节点上。应用 i 容器部署算法如图 3 所示。图中应用 i 容器会部署失败,是因为云平台中没有满足该资源调度模型的节点,此时应添加更多的物理服务器到云平台中。此外把应用 i 容器部署在节点 $rack_k.host_j$(第 $k+1$ 个机架上的第 $j+1$ 台主机) 上之前,要把该应用的数据副本 M_1 存储在该节点上,剩下的 M_2 和 M_3 按照 2.1 节提出的数据存储策略存储到相应的云平台节点中。

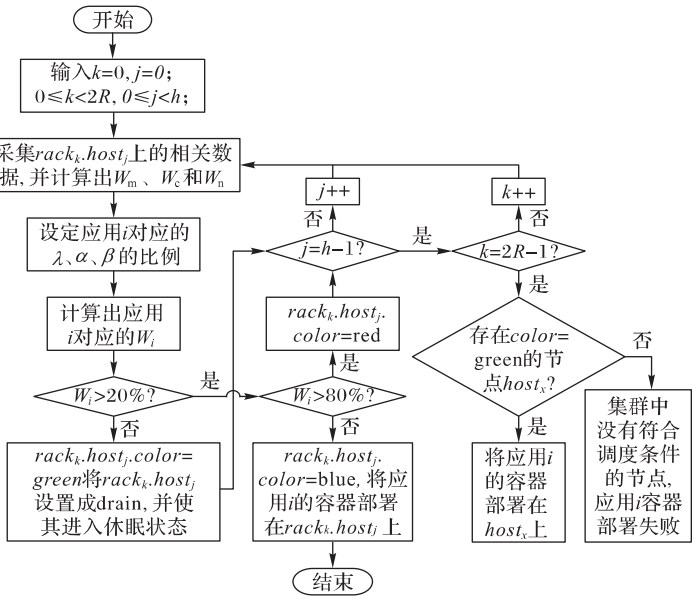

图 3　应用 i 容器部署流程

Fig. 3　Flow chart of container deployment for application i

3.2　应用迁移算法

对于处于 $color$ = red 状态的节点,当该类节点在一段时间内的平均资源利用率出现大于一定量 W(W 为常量,可由云平台管理员根据云平台情况设定) 时,要对应用进行迁移,否则会因为各应用容器计算资源匮乏而使得应用性能下降,具体的应用迁移算法如下:

假设某机架中 $host_t(0 \leqslant t < h, t \in \mathbf{Z})$ 上的应用容器需要进行迁移至目的主机 $host_d(0 \leqslant d < h, d \in \mathbf{Z})$; $host_t$ 中运行的容器总数为 C^t, $Container_i^t$ 为 $host_t$ 中第 i 个部署的应用容器编号; 用 $W_i^t(0 \leqslant i \leqslant C^t, i \in \mathbf{Z})$ 表示节点 $host_t$ 针对第 i 个应用容器的综合负载值,且 $W_0^t = 0$; 用 $W_i^{*\,t}$ 表示节点 $host_t$ 部署了第 i 个应用容器后的综合负载; $\Delta P_i^t = W_i^{*\,t} - W_{i-1}^{*\,t}$ 表示在节点 $host_t$ 上部署了 $Container_i^t$ 后 $host_t$ 增加的综合负载值,则

$$\overline{\Delta P^t} = \frac{1}{C^t} * \sum_{i=1}^{Ct} \Delta P_i^t = \frac{1}{C^t} \sum_{i=1}^{Ct} (W_i^{*\,t} - W_{i-1}^{*\,t})$$

表示部署了应用容器后服务器节点的综合负载平均增长量; 向量 $\boldsymbol{\varepsilon} = (\varepsilon_1, \varepsilon_2, \varepsilon_3)$ 表示容器迁移时对内存、CPU 和网络资源的容忍度,常量 ε_1、ε_2、ε_3 表示节点允许内存、CPU 和网络使用率达到的最大值,由云平台管理源根据情况设定; $\boldsymbol{W} = (W_{\mathrm{m}}, W_{\mathrm{c}}, W_{\mathrm{n}})$ 表示节点在当前状态下的内存、CPU 和网络使用情况; 用集合 S 表示筛选过的不符合条件的 ΔP_i^t, 初始时 $S = \varnothing$; $P = \{\Delta P_1^t, \Delta P_2^t, \cdots, \Delta P_i^t, \cdots, \Delta P_k^t\}$, $k = C^t$; $|\overline{\Delta P^t} - \Delta P_T^t| = \min\{|\overline{\Delta P^t} - \Delta P_i^t|\} \leqslant \mu$, 其中 T 为符合迁移条件容器中迁移代价最小的应用容器编号,令 $t[i++] = T$, 则数组 $t[\,]$ 记录了迁移代价由小到大的应用容器编号,其中 μ 为常数,表示能接受的应用迁移代价的最大值,可由云平台管理员视情况而设定; 用 $Container_T^t$ 表示节点 $host_t$ 中选定的需要迁移的应用容器; 用 $\boldsymbol{W}_T^c = (W_{T\mathrm{m}}^c, W_{T\mathrm{c}}^c, W_{T\mathrm{n}}^c)$ 表示 $Container_T^t$ 中内存、CPU 和网络的使用状态; 用 $host_x^A(0 \leqslant x < h$ 且 $x \in \mathbf{Z})$ 表示 A 区中存储了 $Container_T^t$ 应用数据的另一个节点, $\boldsymbol{W}^{A.x} = (W_{\mathrm{m}}, W_{\mathrm{c}}, W_{\mathrm{n}})$ 表示 $host_x^A$ 的资源使用情况; 用 $host_x^B$ $(0 \leqslant x < h$ 且 $x \in \mathbf{Z})$ 表示 B 区中存储了 $Container_T^t$ 应用数据的节点, $\boldsymbol{W}^{B.x} = (W_{\mathrm{m}}, W_{\mathrm{c}}, W_{\mathrm{n}})$ 表示 $host_x^B$ 的资源使用情况。查找迁移目的主机的具体算法如下:

输入　S、P、μ、$\boldsymbol{\varepsilon}$;

输出　$host_d$。

步骤 1　在集合 $P - S$ 中查找,得到符合迁移条件的

$Container_T^t$，并得到对应的 $\boldsymbol{W}_T^c = (W_{Tm}^c, W_{Tc}^c, W_{Tn}^c)$；

步骤 2　查看 Ceph 集群的 Crush map，找到 $host_x^A$，得到 $\boldsymbol{W}^{A,x} = (W_m, W_c, W_n)$，并判断 $\boldsymbol{W}^{A,x} + \boldsymbol{W}_T^c \leqslant \boldsymbol{\varepsilon}$ 是否成立，若成立，则输出 $host_d = host_x^A$，查找结束；

步骤 3　若 $S \neq P$，令 $S = S \cup \{\Delta P_T^t\}$，并跳到步骤 1 开始执行；

步骤 4　若 $S = P$，则令 $y = 0, T = t[y++]$；

步骤 5　若 $y \neq C^t + 1$，查看 Crush map，找到 $host_x^B$，得到 $\boldsymbol{W}^{B,x}$，并判断 $\boldsymbol{W}^{B,x} + \boldsymbol{W}_T^c \leqslant \boldsymbol{\varepsilon}$ 是否成立，若成立，则输出 $host_d = host_x^B$，查找结束；

步骤 6　若 $y = C^t + 1$，查找结束。

若得到了 $host_d$，用 commit 命令将 $Container_T^t$ 打包成镜像 $Container_{T^*}^t$，并上传至本地仓库；使节点 $host_d$ 处于运行状态，并在 $host_d$ 上用 $Container_{T^*}^t$ 的镜像部署一个应用容器 $Container^d$，并确保 $host_d$ 中的所有 OSD 都是 $Container^d$ 的数据卷，然后杀死 $host_t$ 中的 $Container_T^t$ 容器进程，应用迁移成功。若没有得到 $host_d$，即 $y = C^t + 1$，则表示集群中没有符合迁移条件的容器或目的主机，应用迁移失败，则应适当增大 μ，或向云平台中增加一定数量的集群节点。

4　实验及对比分析

4.1　实验环境

为了验证该调度策略的可行性和有效性，在基于 Ceph 集群的 Docker 云平台上进行了实验测试。实验中使用 9 台戴尔 R710 服务器。服务器的部署环境如下：A 区包括 $2R = 6$ 台物理服务器，分别位于三个不同的机架(Rack)中，则每个机架中 2 台，即 $h = 2$；B 区包括 $R = 3$ 台物理服务器，位于 3 个不同的机架中；服务器均采用 Centos7(64 位)操作系统，8 核处理器，2 张千兆网卡和 32 GB 内存，磁盘容量为1 TB，做了 RAID5。所使用的 Ceph 版本为 Jewel(10.2.9)，每个服务器中将 3 个目录文件作为 node 节点的 OSD，则 $N = 3$，数据副本数为 3，使用的 Docker 版本为 17.06.0-ce，每台服务器既是 Ceph 集群的 node 节点，也是 Swarm 集群的 Docker 节点。为了更真实地模拟云平台环境，在某些服务器上运行了占用不同服务器计算资源的虚拟机，并用以下两个应用进行实验：

①用 Java 语言编写简单的计算程序，该程序中用到了乘法、除法和开根号的算数运算，将程序所需的数据先存放在文件中，将中间计算结果写入 OSD 中，数据大小约为 8.5 GB，并将开始时间和结束时间输出到屏幕上；

②用 Tomcat 部署一个 Java Web 应用，该应用实现对磁盘中文档的搜索功能，实验用的数据文档存放在集群节点的 OSD 中，并将搜索到的文档路径返回到屏幕上。

4.2　实验过程及对比分析

由于是在集群中部署第一个应用，集群节点处于低负载状态，所以将 $rack_0.host_1$、$rack_1.host_1$ 和 $rack_2.host_1$ 直接置为休眠状态，接下来只要考虑 $rack_0.host_0$、$rack_1.host_0$ 和 $rack_2.host_0$ 的资源负载情况。从 Linux 系统中的/proc 目录中获取相关的性能参数，采集数据时间间隔为 10 min，一共采集 10 次。对采集的数据进行整理后，计算出内存、CPU 和网络的平均使用率，如表 1 所示。表中 $rack_0.host_0$、$rack_1.host_0$ 的内存和 CPU 平均使用率较高，是因为我们在这两个节点上运行了占用不同系统资源的虚拟机，$rack_0.host_0$ 的网络利用率最高是由于 $rack_0.host_0$ 是集群的管理节点，集群中的其他节点要与其进行通信。另外，由表 1 中的数据结合式(1)可计算出各节点的综合负载 $W_0^{*\,t}$ 分别为 20.7%、26.3% 和 13.0%。

表 1　集群节点资源平均使用率　%

Tab. 1　Average resource usage of cluster nodes　%

节点	W_m	W_c	W_n	W_i
$rack_0.host_0$	43.1	22.9	8.3	20.7
$rack_0.host_1$	40.9	30.3	7.8	26.3
$rack_0.host_2$	20.2	11.6	7.3	13.0

先选择应用①来进行实验测试。这显然是一个 CPU 密集型应用，设 λ：α：β = 2：3：1，由调度优化模型中的式(3)可得 $\lambda = \frac{1}{3}, \alpha = \frac{1}{2}, \beta = \frac{1}{6}$，结合式(2)可得该三个节点对该应用的综合负载值分别为 24.2%、30.0% 和 13.8%，则各节点的状态为 blue、blue 和 green，所以按照 3.1 节提出的应用容器部署算法，应该尽量将应用部署在节点 $rack_0.host_0$ 和 $rack_1.host_0$ 上，让节点 $rack_2.host_0$ 进入休眠状态。选择将应用部署在 $rack_1.host_0$ 上，则按照本文提出的调度策略，应先将应用①所需的数据存储在节点 $rack_1.host_0$ 的 OSD 中。设邻域 $\delta = 1$，则按照 2.1 节提出的存储策略可将该应用数据 M_1、M_2、M_3 分别存放在 A 区的 $rack_1.host_0$、$rack_0.host_0$ 和 B 区的任意一台节点中，其中 $rack_1.host_0$ 为主 OSD 所在节点，其余两个节点为副 OSD 所在节点。由于 Ceph 集群中 Ceph 客户端只写主 OSD，次 OSD 由主 OSD 负责写，主次 OSD 之间会通过 peering 过程同步数据，且具有自我修复能力，因此在写主 OSD 时，应将 B 区的任意一个节点从休眠状态切换成运行状态。将 crush map 作如下修改：没写出来的 osd 的权重与 osd.26 的权重一样，都为 0.000。

```
gRoot setA1{
id -1
alg straw
hash 0
item osd.0 weight 0.010
item osd.1 weight 0.010
item osd.2 weight 0.010
…
item osd.6 weight 0.010
item osd.7 weight 0.010
item osd.8 weight 0.010
…
item osd.18 weight 0.010
item osd.19 weight 0.010
item osd.20 weight 0.010
…
item osd.26 weight 0.000
}
```

将应用数据存储好后，就在节点 $rack_1.host_0$ 中部署应用①，应用①执行过程中要把中间结果写入到主 OSD 中，便于查看因应用迁移带来的损失。然后按照之前的数据处理方法，得出 $rack_1.host_0$ 当前状态下的 $W_m = 62.6\%, W_c = 71.3\%, W_n = 47.0\%$，按照 3.2 节提出的应用容器迁移算法，令 $t = rack_1.host_0$，则可计算出 $W_1^{*\,t} = \frac{1}{3} \times (62.6\% + 71.3\% + 47.2\%) = 60.4\%$，$\Delta P_1^t = W_1^{*\,t} - W_0^{*\,t} = 60.4\% - 26.3\% = 34.1\%$，设 $\boldsymbol{\varepsilon} = (\varepsilon_1, \varepsilon_2, \varepsilon_3) = (0.9, 0.9, 0.9)$，此

时 $\boldsymbol{W}=(W_m, W_c, W_n)=(62.6\%, 71.3\%, 47.0\%) \leqslant \boldsymbol{\varepsilon}$。按照部署应用①容器的步骤,在节点 $rack_1.host_0$ 上部署应用②。设λ: α: $\beta=1$: 1: 2,并得到对应的 $W_1=\frac{1}{4}\times 62.6\%+\frac{1}{4}\times 71.3\%+\frac{2}{4}\times 47.0\%=57.0\%$,记录进行 10 次文件搜索时的内存、CPU 和网络利用率,经计算得出 $rack_1.host_0$ 此状态下的平均资源利用率分别为 $W_m=79.3\%$, $W_c=84.1\%$, $W_n=83.7\%$, $W_2^{*\,t}=\frac{1}{3}\times(79.3\%+84.1\%+83.7\%)=82.4\%$, $\Delta P_2^t=W_2^{*\,t}-W_1^{*\,t}=82.4\%-60.4\%=22.0\%$, $\overline{\Delta P^t}=\frac{1}{2}\times(34.1\%+22.0\%)=28.05\%$。为了达到应用迁移算法中应用迁移的条件,调整虚拟机的资源利用情况,使得 $W_m=0.95\geqslant\varepsilon_1=0.9$,即使得 $rack_1.host_0$ 节点满足应用迁移条件。由于 $|\Delta P_1^t-\overline{\Delta P^t}|=|\Delta P_2^t-\overline{\Delta P^t}|=6.05\%$,故可设 $T=1$,即选择节点 $rack_1.host_0$ 中的应用①进行迁移,则要迁移的容器为 $Container_1^t$,用 stats 命令查看 $Container_1^t$ 的运行状态,得到该容器中的平均资源使用值 $W_1^c=(40.6\%, 43.4\%, 30.7\%)$,利用应用容器迁移算法,得出迁移的目的主机为 $host_d=A.rack_0.host_0$,即为 A 区的第 1 个机架中的第 1 台服务器。应用①在各种情况下的完成时间对比如表 2 所示。从表 2 中可以看出,不发生迁移的情况下,应用运行在物理机上的完成时间最短,而在虚拟机中运行的完成时间最长,这是因为 Docker 是基于 Linux 内核的操作系统层虚拟化技术,而虚拟机则需要通过客户机来调用宿主机中的计算资源,相对来说比较慢,所以容器比虚拟机快,但比物理机慢,是因为容器除了需要额外的启动时间外,容器访问数据卷的速度也比物理机直接访问磁盘的速度低; 而在发生应用迁移的情况下,应用在容器中运行的完成时间要比运行在虚拟机中的时间稍长,这说明该应用迁移算法会使得应用的性能略有下降,但也在用户可接受的范围之内。

表 2　应用①完成时间对比

Tab. 2　Completion time comparison of application ①

运行环境		完成时间/s
容器	发生迁移	217
	不发生迁移	184
虚拟机	不发生迁移	196
物理机	不发生迁移	155

为了进一步验证该调度策略的有效性,在改进前和改进后的 Docker 云平台上运行不同数量的应用①容器,对集群中处于运行状态的服务器数量进行了对比,对比结果如表 3 所示。从表 3 可以看出,在部署相同数量应用的情况下,优化后集群中处于运行态的节点数量要比优化前的少。优化前的集群在部署第 8 个应用时开始启用 B 区的服务器,而优化后的集群在部署第 12 个应用时,运行态的节点数突然快速增加,并开始启用 B 区的服务器,这是因为 A 区有一台节点宕机,而且此时 A 区没有满足条件的目的主机,所以需要把应用迁移到 B 区。而优化前的集群在部署第 8 个应用时就启用 B 区的服务器,可能是因为容器编排器 Swarm 的平均部署算法所致,从这里也可以看出,本文提出的调度算法对集群资源进行了更细粒度的划分,并能够在一定程度上减少数据中心处于运行态的服务器数量,以此达到了降低数据中心运营成本的目的,而且在集群相对稳定的情况下节约效果更明显。

表 3　集群中处于运行态节点数对比

Tab. 3　Comparison of the number of cluster nodes in running state

应用数	处于运行态节点数		应用数	处于运行态节点数	
	优化前	优化后		优化前	优化后
2	6	2	10	7	5
4	6	2	12	8	7
6	6	3	14	9	9
8	6	4			

Docker 有 Compose、Machine 和 Swarm 三种官方容器编排工具,但前两种都是对单主机上的容器进行编排,而本文研究的是集群 Docker 容器的调度问题,所以这里不作讨论。另外,Kubernetes 是现今比较流行的一种开源集群容器编排工具,它与 Swarm 都采用 Go 语言开发,两者具有可比性。因此,为了测试本文提出的 Docker Swarm 容器编排工具与其他几种编排工具的性能差异,将其与原生 Swarm 和 Kubernetes (V1.7) 进行了实验和对比。在改进了存储方案的 Ceph 集群中进行实验,将 10 个应用①容器部署在该集群中,实验共进行了 30 次,对采集到的数据分别求平均值,结果如表 4 所示,其中 Swarm_opt 表示本文改进的 Swarm 容器编排工具。需要指出的是,由于 Kubernetes 的容器复制控制器(Replication Controller) 可以通过复制而产生相同的容器副本,而 Swarm 是从仓库中拉取镜像生成容器,两者的差异对应用的完成时间影响很大,为了避免这种差异对实验结果造成的影响,实验中并没有使用 Kubernetes 的容器复制控制器功能。此外,Kubernetes 中的最小调度单位是 Pod,而 Swarm 是以 Container 为最小调度单位的,因此实验中,每个 Pod 中只包含一个 Container,每个应用都是一个作业(job)。

表 4　不同容器编排工具性能对比

Tab. 4　Performance comparison of different Docker orchestration

容器编排工具	开启节点数 M	集群负载 P/%	完成时间/s	
			不迁移(T)	迁移(T^*)
Swarm	7	79.3	261	360
Swarm_opt	5	92.4	368	437
Kubernetes	7	80.7	272	381

由表 4 可以看到,当在集群中部署 10 个应用①时,使用 Swarm、Swarm_opt、Kubernetes 三种不同容器编排工具,集群中开启的节点数分别为 M(Swarm) = 7, M(Swarm_opt) = 5, M(Kubernetes) = 7,结合集群负载来看,这也体现了本文提出的容器编排器 Swarm_opt 在提高资源利用率上具有一定的优势。而在 M(Swarm) = M(Kubernetes) = 7 的情况下,集群负载 P(Kubernetes) > P(Swarm),这可能是因为两者的实现机制决定的,因为 Kubernetes 比 Swarm 的实现机制更复杂。另外,在应用不发生迁移时,应用的完成时间从小到大依次为 T(Swarm_opt) > T(Kubernetes) > T(Swarm),本文提出的容器调度算法用的时间较长,这是由于集群中开启的节点数较少,各节点的资源在达到更高利用率的同时也稍微降低了节点的性能,故应用完成时间较长,但在节点开启数量减少 25.87% 的情况下,每个应用增加的完成时间仅增长了 4%,而在非实时的应用中这种结果是很可观的。而 T(Kubernetes) > T(Swarm) 是因为 Kubernetes 中容器的启动速度比在 Swarm 中小,从而使得 Kubernetes 中应用的完成时

间更长。而在应用发生迁移的情况下,与不发生应用迁移时的应用完成时间的增长量从小到大依次为:

$$[T^*(\text{Swarm\_opt}) - T(\text{Swarm\_opt})] > [T^*(\text{Swarm}) - T(\text{Swarm})] > [T^*(\text{Kubernetes}) - T(\text{Kubernetes})]$$

从这里可以看出本文提出的调度算法在应用发生迁移时付出的代价最小,而由于 Kubernetes 中的容器启动慢问题,在应用发生迁移时,它付出的代价最大。

5 结语

本文改进了 Ceph 集群的存储策略,建立了一个资源调度优化模型,并在此基础上提出了云资源弹性调度策略,针对每个应用对资源需求情况的不同,对云平台资源进行了细粒度的划分,实现了云资源的弹性调度。该策略在迁移应用不发生迁移的情况下,一定程度上节约了集群资源,达到了合理利用云资源和降低数据中心运营成本的目的;而在发生应用迁移时,应用的性能虽稍有所下降,但与原生的 Swarm 和 Kubernetes 容器编排工具相比,仍能体现一定的优势。如何选择应用迁移时机,进一步减少应用迁移代价,是将来要做的工作。

参考文献:

[1] 刘鹏. 云计算[M]. 3 版. 北京: 电子工业出版社,2015: 1 - 2. (LIU P. Cloud Computing [M]. 3rd ed. Beijing: Publishing House of Electronic Industry, 2015: 1 - 2.)

[2] TURNBULL J. The Docker Book: Containerization is the New Virtualization [M]. Seattle: Amazon Digital Services Inc, 2014: 4 - 7.

[3] ADUFU T, CHOI J, KIM Y. Is container-based technology a winner for high performance scientific applications? [C]// APNOMS 2015: Proceedings of the IEEE 2015 17th Asia-Pacific Network Operations and Management Symposium. Piscataway, NJ: IEEE, 2015: 507 - 510.

[4] TIHFON G M, KIM J, KIM K J. A new virtualized environment for application deployment based on Docker and AWS [C]// ICISA 2016: Proceedings of the 2016 Information Science and Applications, LNEE 376. Singapore: Springer, 2016: 1339 - 1349.

[5] 郝庭毅,吴恒,吴国全,等. 面向微服务架构的容器级弹性资源供给方法[J]. 计算机研究与发展,2017,54(3):597 - 608. (HAO T Y, WU H, WU G Q, et al. Elastic resource provisioning approach for container in micro-service architecture [J]. Journal of Computer Research and Development, 2017, 54(3): 597 - 608.)

[6] 张怡. 基于 Docker 的虚拟化应用平台设计与实现[D]. 广州: 华南理工大学软件学院,2016: 21 - 35. (ZHANG Y. The design and implementation of a virtualized application platform based on Docker [D]. Guangzhou: South China University of Technology, College of Software, 2016: 21 - 35)

[7] KANG D-K, CHOI G-B, KIM S-H, et al. Workload-aware resource management for energy efficient heterogeneous Docker containers [C]// TENCON 2016: Proceedings of the 2016 IEEE Region 10 Conference. Piscataway, NJ: IEEE, 2016: 2428 - 2431.

[8] 杨保华,戴王剑,曹亚仑. Docker 技术入门与实战[M]. 2 版. 北京: 机械工业出版社,2017: 9 - 10. (YANG B H, DAI W J, CAO Y L. Docker Technology Entry and Actual Combat [M]. 2nd ed. Beijing: China Machine Press, 2017: 9 - 10.)

[9] MIELL I, SAYERS A H. Docker in Practice [M]. 2nd ed. Greenwich, Connecticut, USA: Manning Publications, 2016: 124 - 157.

[10] 武志学. 云计算虚拟化技术的发展与趋势[J]. 计算机应用, 2017,37(4): 915 - 923. (WU Z X. Advances on virtualization technology of cloud computing [J]. Journal of Computer Applications, 2017, 37(4): 915 - 923.)

[11] WEIL S A, BRANDT S A, MILLER E L, et al. Ceph: a scalable, high-performance distributed file system [C]// OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation. Berkeley, CA: USENIX Association, 2006: 307 - 320.

[12] SINGH K. Ceph Cookbook [M]. Birmingham: Packt Publishing Ltd, 2016: 53 - 54.

[13] WEIL S A, BRANDT S A, MILLER E L, et al. CRUSHR: controlled, scalable, decentralized placement of replicated data [C]// SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. New York: ACM, 2006: Article No. 122.

[14] 沈良好,吴庆波,杨沙洲. 基于 Ceph 的分布式存储节能技术研究[J]. 计算机工程,2015,41(8): 13 - 17. (SHEN L H, WU Q B, YANG S Z. Research on distributed storage energy saving technologies based on Ceph [J]. Computer Engineering, 2015, 41(8): 13 - 17.)

[15] 彭丽苹,吕晓丹,蒋朝惠. 基于 Ceph 集群的能耗管理策略研究[J/OL]. 计算机工程与应用 (2017-08-10) [2017-09-22]. http://kns.cnki.net/kcms/detail/11.2127.TP.20170810.0852.004.html. (PENG L P, LYU X D, JIANG C H, Ceph cluster based energy consumption management strategy study [J/OL]. Journal of Computer Engineering and Applications (2017-08-10) [2017-09-22]. http://kns.cnki.net/kcms/detail/11.2127.TP.20170810.0852.004.html.)

[16] KANG D-K, CHOI G-B, KIM S-H, et al. Workload-aware resource management for energy efficient heterogeneous Docker containers [C]// TENCON 2016: Proceedings of the 2016 IEEE Region 10 Conference. Piscataway, NJ: IEEE, 2016: 2428 - 2431.

[17] MENG Y, RAO R, ZHANG X, et al. CRUPA: a container resource utilization prediction algorithm for auto-scaling based on time series analysis [C]// PIC 2016: Proceedings of the 2016 International Conference on Progress in Informatics and Computing. Piscataway, NJ: IEEE, 2016: 468 - 472.

[18] GUAN X, WAN X, CHOI B Y, et al. Application oriented dynamic resource allocation for data centers using Docker containers [J]. IEEE Communications Letters, 2017, 21(3): 504 - 507.

[19] 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: 浙江理工大学信息学院,2017: 36 - 60. (HE S L. Research on construction of elastic cluster based on Docker and resource pre-scheduling strategy [D]. Hangzhou: Zhejiang Sci-tech University, School of Information Science and Technology, 2017: 36 - 60.)

This work is partially supported by Major Program for Basic Research of Guizhou Province (黔科合 JZ 字【2014】2001-21).

PENG Liping, born in 1990, M. S. candidate. Her research interests include cloud computing, big data.

LYU Xiaodan, born in 1970, M. S., associate professor. His research interests include cloud computing, algorithm design, data analysis.

JIANG Chaohui, born in 1965, M. S., professor. His research interests include cloud computing, information security, database, software engineering.

PENG Chenghui, born in 1990, M. S. candidate. His research interests include cloud computing, data mining.