Journal of Computer Applications
计算机应用,2021,41(6):1581 - 1588
ISSN 1001-9081
CODEN JYIIDU
2021-06-10
http://www.joca.cn

文章编号:1001-9081(2021)06-1581-08
DOI:10.11772/j.issn.1001-9081.2020121913

面向动态负载的集群容器部署方法

尹　飞[1],龙玲莉[1],孔　峥[1],邵　涵[2],李　鑫[3*],钱柱中[2]
(1.江苏方天电力技术有限公司,南京 211102;　2.南京大学 计算机科学与技术系,南京 210023;
3.南京航空航天大学 计算机科学与技术学院,南京 211106)
(*通信作者电子邮箱lics@nuaa.edu.cn)

摘　要:针对集群负载动态变化引发容器频繁迁移的问题,提出了一种基于资源预留的容器部署方法。首先,设计了基于马尔可夫链模型的单容器资源需求动态变化描述机制,用于刻画单容器的资源需求情况;其次,基于单容器马尔可夫链模型分析了多容器资源动态变化情况,以刻画容器资源需求态势;随后,基于多容器马尔可夫链提出了面向动态负载的容器部署与资源预留算法;最后,基于容器资源需求特征的分析对所提算法的性能进行了优化。基于国产软硬件环境构建了仿真实验环境,仿真结果表明,在资源冲突率方面,所提方法的性能接近最优的峰值配置策略RP,但所需宿主机数量、容器动态迁移次数明显比其更少;在资源利用率方面,所提方法的宿主机使用数量略多于最优的谷值配置策略RV,但动态迁移次数更少,资源冲突率更低;相较于峰谷配置策略RVP,所提方法在综合性能方面更佳。

关键词:集群;容器;动态负载;整合与迁移;马尔可夫链
中图分类号:TP315　　**文献标志码**:A

Deployment method of dockers in cluster for dynamic workload

YIN Fei[1], LONG Lingli[1], KONG Zheng[1], SHAO Han[2], LI Xin[3*], QIAN Zhuzhong[2]
(1. *Jiangsu Frontier Electric Technology Company Limited*, *Nanjing Jiangsu* 211102, *China*;
2. *Department of Computer Science and Technology*, *Nanjing University*, *Nanjing Jiangsu* 210023, *China*;
3. *College of Computer Science and Technology*, *Nanjing University of Aeronautics and Astronautics*, *Nanjing Jiangsu* 211106, *China*)

Abstract: Aiming at the problem of frequent migration of containers triggered by dynamic changes of cluster workload, a container deployment method based on resource reservation was proposed. Firstly, a dynamic change description mechanism of single-container resource demand based on Markov chain model was designed to describe the resource demand situation of single container. Secondly, the dynamic change of multi-container resource was analyzed based on the single-container Markov chain model to describe the container resource demand state. Thirdly, a container deployment and resource reservation algorithm for dynamic workload was proposed based on the multi-container Markov chain. Finally, the performance of the proposed algorithm was optimized based on the analysis of container resource demand characteristics. The simulation experimental environment was constructed based on the domestic software and hardware environment, and the simulation results show that in terms of resource conflict rate, the performance of the proposed method has the performance close to the optimal peak allocation strategy named Resource with Peak (RP), but its number of required hosts and container dynamic migration number are significantly less; in terms of resource utilization rate, the proposed method has the number of hosts used slightly more than the optimal valley allocation strategy named Resource with Valley (RV), but has less dynamic migration number and lower resource conflict rate; compared with the peak and valley allocation strategy named Resource with Valley and Peak (RVP), the proposed method has better comprehensive performance.

Key words: cluster; docker; dynamic workload; consolidation and migration; Markov chain

0　引言

数据中心与集群以云计算的形式,根据用户动态的资源需求变化,“按需式”地为其提供可配置的资源获取服务,以实现索取资源的弹性伸缩,涵盖了传统的基础设施即服务(Infrastructure as a Service, IaaS)[1]、平台即服务(Platform as a Service, PaaS)[2]与新兴的无服务计算[3]等。这些服务模型的支撑在于基于容器化[4]的资源管理技术,以容器为上层应用提供相互隔离的计算资源。

为了应对大规模用户的访问请求,同种应用功能的多个运行实例往往被各自独立地部署于预先创建的容器之中[5]。继而,为了提升宿主机(承载容器的物理机)资源利用率,同时

收稿日期:2020-11-04;**修回日期**:2021-04-11;**录用日期**:2021-04-16。　**基金项目**:国家自然科学基金资助项目(61802182)。
作者简介:尹飞(1978—),男,安徽马鞍山人,高级工程师,主要研究方向:电力信息化、云计算;　龙玲莉(1984—),女,湖北武汉人,工程师,硕士,主要研究方向:电力信息化、云计算;　孔峥(1986—),男,江苏扬州人,助理工程师,主要研究方向:电力信息化、云计算;　邵涵(1994—),女,安徽黄山人,博士研究生,研究方向:分布式系统、计算机网络;　李鑫(1987—),男,江西南昌人,副教授,博士,CCF会员,主要研究方向:云计算、边缘计算;　钱柱中(1980—),男,江苏常熟人,教授,博士,CCF会员,主要研究方向:分布式系统、数据中心网络。

降低多个容器实例间的通信开销，容器会被整合到少量宿主机之上[6-7]。这样“紧致”的部署，能够在满足应用服务的同时，极大提升集群的资源利用率。

不仅如此，在运行过程中，应用的负载往往是动态变化的，文献[8-10]表明集群内各类应用的请求访问量在时间上的分布不均，如：淘宝在“双十一”期间的访问量会剧增；假期时社交网站的照片分享量会极大超出平时的均值水平；新闻事件也会使得应用的浏览量产生大幅度的波动。为此，集群更需要根据应用的负载变化，动态调整容器的资源配置，以满足应用业务的服务质量。

然而，为了整合宿主机资源[11-15]，现有的工作主要从按需式、被动式[16-18]的容器迁移入手。一方面，已有工作探讨了在宿主机上进行共享资源使用过程中，容器之间产生的资源干扰[9,19]；另一方面，一些研究在结合了共享资源使用特点[8,20]的基础上，进行容器整合的策略制定，提升宿主机的资源利用率。虽然也有工作通过预测及刻画应用各异的资源需求[21-22]，帮助容器进行更好的整合，但这些工作均带有较强的输入分布假设。

即使现有的容器化技术已经能较好地支持“应对式”的资源弹性伸缩和配置，但是对于动态负载来说，在线化的配置需求增加了作出合理容器部署的难度。一个简单的例子是，短期内“近乎完美”的容器整合可能造成未来更多容器热迁移。也就是，当容器负载过高而宿主机没有空闲资源进行拓展时，容器不得不采取热迁移的方式，将实例转移至其他空闲服务器。随之而来，热迁移不仅消耗大量计算与网络资源[23]，也会增加应用响应时间，对服务质量产生较大的影响。

结合针对集群负载的长期观察，容器的资源需求往往是此消彼长。如果能将这些此消彼长的容器整合在一台宿主机上，那么容器将会以较小的概率同时具有资源扩展需求。因此，只需为每一台宿主机预留少量资源，即可避免后续大部分的容器热迁移。然而，如何将合适的容器整合宿主机上，并确定合适的资源预留量是关键，其直接影响了宿主机的资源利用率与容器的服务质量。

虽然容器的资源需求量在不断变化，但是在弹性资源伸缩的时候，其申请增加或是释放的资源量往往具有最小的变化单位，如一个CPU核、1 GB内存等。相较于传统“非忙即闲”的资源模型，有限的资源使用状态及其间的变化关系更能作为细粒度描述容器资源动态变化的依据。例如，聊天应用包含有文字、语音等多种媒介，以至于信息传输负载的呈现也仅在这几个状态间切换。

为此，本文首先以细粒度的方式刻画容器的动态资源变化，即用马尔可夫链中的状态和状态间的转移，对应建模容器可能的资源需求量和不同资源需求量之间的切换。图1展示了一个具有3种资源需求状态的容器示例。然后，本文在基于资源预留的基础上提出面向动态负载的容器部署问题，以在限定动态迁移概率的约束下，最小化宿主机的使用数量。本文提出了面向动态负载的容器部署策略，并通过估计容器资源需求，提高部署效率。大规模基于国产软硬件的实验结果表明，所提出的部署策略能在提高宿主机资源利用率和减少容器动态迁移次数间取得平衡，在长效时间上以至多26台宿主机的增加换取至多7次容器迁移。

本文的主要工作有：1）用马尔可夫链建模容器的多状态资源需求变化；2）提出面向多状态负载的容器部署策略，利用资源需求量的估计值提高部署效率；3）进行大量实验，通过与其他部署策略比较，评估本文所提出的部署策略的性能。

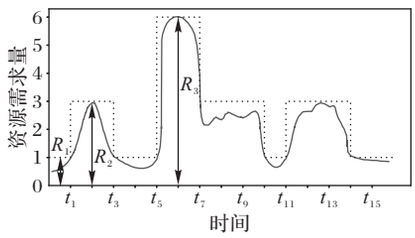

图1 容器随时间变化的资源需求

Fig. 1 Resource demand of dockers varying with time

1 相关工作

1.1 虚拟机/容器负载刻画

动态负载的变化是虚拟机/容器部署过程不可忽视的特征。由于容器是轻量级的虚拟机，因此，本文只考虑针对容器的相关部署与应用。文献[8]统计了Google集群中不同资源的利用率以及各自对应的总时间占比，表明了资源需求变化的动态性和可描述性；文献[19]研究了负载激增的模式，提出有关激增负载测试的方法；文献[9]调研了资源需求量波动对服务质量的影响；文献[20]优化了数据中心与集群针对容器负载变化而产生的重配置资源的开销。上述工作对资源配置的某方面开展了研究，但未考虑合理调整资源利用的方式。

近期工作主要致力于把变化的负载建模成为随机变量，即非确定量。文献[21]提供了在资源需求量服从特定随机分布下的近似算法；文献[22]主要研究了资源需求服从正态分布的情况；文献[14-15]提出针对资源需求量服从高斯分布的近似算法。这些相关方法可以提供一定的资源需求描述能力，但难以支撑容器部署的稳定性要求，易引发容器迁移。

然而针对服从特定概率分布的资源需求量而设计的算法不具有普遍性。虽然文献[24]把激增的资源需求量建模为二状态马尔可夫链，但该算法对资源需求量的状态数等有较大限制。

1.2 虚拟机/容器整合与迁移

作为提高资源利用率的重要方式，虚拟机/容器整合[11,25]已被学术界和工业界广泛地研究与认可。工业界代表性的产品有VMware分布式资源调度[12]和IBM（International Business Machine）服务工具[13]。大多数研究把这个问题视作：在服务器容量、服务等级协议等限制条件下最小化所使用的物理服务器数量。这些工作均将容器资源需求量表示为定值，使用一系列类似装箱的启发式策略，如降序首次适应（First Fit Decreasing, FFD）[6]、随机装箱等[14-15]，却无法应对集群环境中动态的容器资源变化需求。

当宿主机资源无法满足容器资源扩张需求时，容器热迁移随即进行。文献[16]以同时最小化迁移传输能耗及容器服务时延作为目标；文献[17]旨在设计具有时延保障的热迁移策略；另有文献[18]考虑集群间利用传输控制协议（Transmission Control Protocol, TCP）的路径多样性进行热迁移。但这类工作往往是被动地进行容器的调整，没有在长效时间维度上进行容器整合与迁移的优化。

针对上述工作的局限，本文使用多状态马尔可夫链为容器的资源需求量变化进行建模，以设计有效的面向动态负载

的容器部署策略。

2　系统模型

2.1　场景定义

集群有大量的物理机,通过 在物理机(宿主机)上部署容器来向用户提供服务。如图2所示,容器与宿主机的关系由部署策略决定。

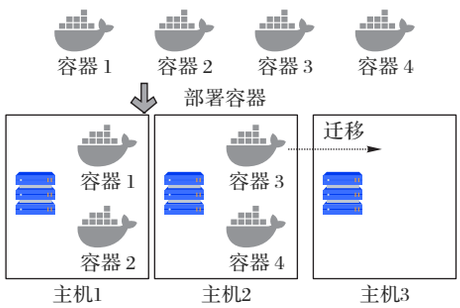

图2　容器部署示意图

Fig. 2　Schematic diagram of docker deployment

一方面,尽管用户通常会声明容器的最大资源需求量,但由于物理机的购置成本、维护成本和运行成本都较高,总是为每个容器预留足够的计算资源在经济上是不合理的。运营者总是希望尽可能减少宿主机的占用量。另一方面,若没有为容器预留一定的计算资源,当其资源需求量发生变化时,集群必须通过动态迁移来保证容器的性能不受影响,由此会带来较大的迁移代价。基于这两点考虑,本文旨在为集群设计一种容器部署策略,使得宿主机资源利用率较高、数量较少,同时把容器动态迁移次数控制在较小的范围内。

2.2　单容器资源需求模型

容器资源需求量是动态变化的,能被量化为多个状态(如图1所示)。本文将该特性称为容器的多状态特性,并用多状态的马尔可夫链进行建模,如图3所示。假设集群共创建了 n 台容器,并考虑第 k 台容器。用 N_k 表示第 k 台容器的状态总数,并令 $\boldsymbol{R}(k)=(R_1(k),R_2(k),\cdots,R_{N_k}(k))$,其中 $R_i(k)$ 表示第 k 台容器中状态 i 的资源需求大小。本文用 $P_{ij}(k)$ 表示第 k 台容器中状态 i 到状态 j 的转移概率,那么这些参数所构成的 $\boldsymbol{P}(k)=\left[P_{ij}(k)\right]_{N_k\times N_k}$ 是这台容器的状态转移概率矩阵。本文用 $S_{ij}(k)$ 表示第 k 台容器在两个连续时间段从状态 i 转换到状态 j 的次数,那么有 $P_{ij}(k)=S_{ij}(k)\Big/\sum_{j'=1}^{N_k}S_{ij'}$。以图1所示为例,分别用 $R_1(k)$、$R_2(k)$、$R_3(k)$ 表示由少到多的三种资源需求量,则 $S_{11}(k)=2,S_{12}(k)=2,S_{13}(k)=1$,且能计算转移概率为 $P_{11}(k)=0.4,P_{12}(k)=0.4,P_{13}(k)=0.2$。

一般来说,上述描述的马尔可夫链通常是非周期不可约的,因为实际应用中任两个状态都有可能相互切换。而任何状态都可能在下一刻保持不变,故存在有平稳状态的分布,即 $\boldsymbol{\mu}(k)=(\mu_1(k),\mu_2(k),\cdots,\mu_{N_k}(k))$,有:

$$\mu_j(k)=\sum_{i=1}^{N_k}\mu_i(k)P_{ij}(k) \tag{1}$$

因此,容器 k 的描述可由其资源需求量 $\boldsymbol{R}(k)$ 和平稳分布 $\boldsymbol{\mu}(k)$ 组成,构成的是 $2\times n$ 维矩阵,定义为 $\boldsymbol{V}(k)=\left[\boldsymbol{R}(k),\boldsymbol{\mu}(k)\right]^{\mathrm{T}},1\leqslant k\leqslant n$。

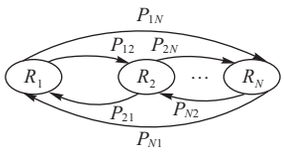

图3　基于马尔可夫链的容器资源状态模型

Fig. 3　Markov chain based resource state model for docker

2.3　多容器的复合资源需求模型

对于 n 台容器来说,每个 $\boldsymbol{V}(k)$ 有 N_k 种状态,每台容器的状态变化相互独立。因此,这 n 台容器一共有 $N=\prod_{k=1}^{n}N_k$ 种状态。用 $\boldsymbol{S}(k)$ 表示容器 $\boldsymbol{V}(k)$ 当前的所有状态,则这 n 台容器在当前的复合状态可以用 $\boldsymbol{S}=(S(1),S(2),\cdots,S(n))$ 来表示。那么,从状态 $\boldsymbol{S}$ 转移到 $\boldsymbol{S}'=(S(1)',S(2)',\cdots,S(n)')$ 的概率为:

$$P_{SS'}=P_{S(1)S(1)'}(1)\times P_{S(2)S(2)'}(2)\times\cdots\times P_{S(n)S(n)'}(n) \tag{2}$$

这里需要说明,本文将总共 N 种状态用 1~N 进行标号。更进一步,对于当前所处的状态 $(S(1),S(2),\cdots,S(n))$,本文用 i_1 表示 $S(1)$,i_2 表示 $S(2)$,以此类推。从而有状态转移矩阵 $\boldsymbol{P}=[P_{ij}]_{N\times N}$。这个马尔可夫链通常也是无周期不可约,故存在唯一的可达的平稳分布:

$$\boldsymbol{\pi}=\left(\pi_1,\pi_2,\cdots,\pi_N\right) \tag{3}$$

根据2.2节所述,容器 $\boldsymbol{V}(k)$ 的平稳分布定义为 $\boldsymbol{\mu}(k)=(\mu_1(k),\mu_2(k),\cdots,\mu_{N_k}(k))$。则 $\boldsymbol{\pi}$ 和 $\boldsymbol{\mu}(k)$ 的关系如定理所述。

定理1　令 $\pi_i'=\prod_{k=1}^{n}\mu_{i_k}(k)$,$\boldsymbol{\pi}'=(\pi_1',\pi_2',\cdots,\pi_N')$,有 $\boldsymbol{\pi}=\boldsymbol{\pi}'$,即 n 台容器平稳处于某一状态的概率等于所有容器各自平稳时处于对应状态概率的乘积。

证明　因为 $\boldsymbol{\pi P}=\boldsymbol{\pi}$ 解得的 $\boldsymbol{\pi}$ 是唯一的,所以只需要证明 $\boldsymbol{\pi}'\boldsymbol{P}=\boldsymbol{\pi}'$,即证 $\pi_j'=\sum_i\pi_i'P_{ij}$,也即:

$$\sum_i\pi_i'P_{ij}=\sum_i\left[\prod_{k=1}^{n}\mu_{i_k}(k)\right]\left[\sum_{k=1}^{n}P_{i_kj_k}(k)\right]=$$

$$\prod_{k=1}^{n}\left[\sum_{i_k}\mu_{i_k}(k)P_{i_kj_k}(k)\right]=\prod_{k=1}^{n}\mu_{j_k}(k)=\pi_j'$$

证毕。

定理1说明,在实际计算中,只需要预先算出每台容器的平稳状态分布,再进行简单乘法运算就能得到 n 台容器的平稳状态分布。

2.4　动态负载容器的部署与资源预留问题

集群有 m 台宿主机,第 l 台宿主机 H_l 的可用资源可以用其容量 C_l 来描述,即 $H_l=C_l,1\leqslant l\leqslant m$。容器的部署过程实际上是从容器到宿主机的一个映射过程,用矩阵 $\boldsymbol{X}=\left[X_{kl}\right]_{n\times m}$ 来表示,当 $\boldsymbol{V}(k)$ 放置在 H_l 上时有 $X_{kl}=1$,否则 $X_{kl}=0$。

令 $W(k,t)$ 表示容器 $\boldsymbol{V}(k)$ 在 t 时刻的资源需求量。理想情况下,所有宿主机在 t 时刻都能提供充足的计算资源。该约束条件可以表示为:

$$\sum_{k=1}^{n}X_{kl}W(k,t)\leqslant C_l;\ \forall l\in\{1,2,\cdots,m\} \tag{4}$$

特别地，在初始时刻，即$t=0$时，该约束满足。如果在t时刻该约束被破坏，则定义为发生了一次冲突。用$vio(l,t)$表示H_l是否在t时刻冲突，即：

$$vio(l,t)=\begin{cases}1, & H_l\text{在}t\text{时刻发生了冲突}\\0, & \text{其他}\end{cases} \tag{5}$$

冲突率为长时间运行过程中发生冲突的时间占总时间的比率，宿主机H_l的冲突率为：

$$CVR(l)=\frac{\sum_t vio(l,t)}{t} \tag{6}$$

CVR越小意味着容器动态迁移的几率越小。本文的策略目标是保证所有宿主机的冲突率保持在一个较低的值ρ以下，即使得：

$$CVR(l)\leqslant\rho;\ \forall l\in\{1,2,\cdots,m\} \tag{7}$$

上述约束为宿主机的性能约束。下面给出面向动态负载的容器部署问题的定义。

定义1 面向动态负载的容器部署问题是指，给出所有容器和宿主机参数$\boldsymbol{V}$和$\boldsymbol{H}$，找到一个从容器到宿主机的映射$\boldsymbol{X}=\left[X_{kl}\right]_{n\times m}$，使得在初始时刻满足容量约束以及在任意时刻满足性能约束的同时，总共使用的宿主机数量最少，也即：

$$\text{minimize:}\left|l\middle|1\leqslant l\leqslant m,\ \sum_{k=1}^{n}X_{kl}>0\right|$$

满足容量约束(式(4))，当$t=0$时；满足性能约束(式(7))，对所有t。

定理2 面向动态负载的容器部署是NP(Non Polinomial)完全的。

证明 通过将NP完全的装箱问题判定版本归约到所提出的面向动态负载的容器部署问题的判定版本来完成定理证明。装箱问题判定版本为：给定n个物品，第k个物品的大小是$g_k\in(0,1]$，能否用m个单位容积箱子装下物品？

对任意一个给定的装箱问题判定版本实例，构造一个面向动态负载的容器部署问题(判定版)实例：令N_k为任意正整数，$\forall k\in\{1,2,\cdots,n\}$；令$R_i(k)=g_k,\forall k,\forall i\in\{1,2,\cdots,N_k\}$；再令$C_l=1,\forall l\in\{1,2,\cdots,m\}$；令$\rho=0$。这样可在多项式时间内完成归约，因此所提问题是NP完全的。 证毕。

3 面向动态负载的容器部署与资源预留

3.1 容器的资源预留策略

在单台宿主机上部署n台容器，这n台容器一共有N个状态，用R_i表示标号为i的状态的资源需求。不失一般性，不妨设$R_1\leqslant R_2\leqslant\cdots\leqslant R_N$，则可以导出这$n$台容器最小所需容量$R_q$，其中$q$为满足如下关系的整数：

$$\sum_{i=1}^{q}\pi_i\geqslant 1-\rho>\sum_{i=1}^{q-1}\pi_i \tag{8}$$

此时的性能约束满足：

$$CVR=\lim_{t\to\infty}\frac{\sum_t vio(t)}{t}=\sum_{i=q+1}^{N}\pi_i\leqslant\rho \tag{9}$$

根据式(8)，判断n台容器能否同时放置在某台宿主机l上的算法$De(\boldsymbol{V},H_l)$，具体描述见算法1所示，其输出为能否放置的结果。

算法1 宿主机能否容纳容器(Determination)。

输入 特定宿主机l，容器的马尔可夫模型$\boldsymbol{V}=\{V(1),V(2),\cdots,V(n)\}$；

输出 n台容器能否放于特定宿主机l。

```
计算容器所有状态的资源需求量;
PI = 0;                                  //PI为资源需求大于H_l的概率
for i = 1 to N
  if R_i ≥ H_l
    PI = PI + π_i
  endif
endfor
如果PI > ρ,输出0(否);否则输出1(是)。
```

3.2 面向动态负载的容器部署策略

现在考虑多台宿主机的情况。将n台容器部署到m台宿主机上，基于FFD启发式算法设计部署策略，将n台容器按资源需求量的期望$\boldsymbol{R}(k)\cdot\boldsymbol{\mu}(k)$降序排序，并将$m$台容器按其容量降序排序。对给定的一个容器，依次尝试每个宿主机，若当前宿主机满足约束条件，则将容器部署在该宿主机上，否则尝试下一个宿主机直到这台容器被部署成功。算法MultiCons(n,m,d,$\boldsymbol{V}$,$\boldsymbol{H}$)的具体描述见算法2所示，其中向量$\boldsymbol{H}$包含所有可用的宿主机，d为单个宿主机上所能允许部署的最大容器数量。

记$N_{\max}$为n台容器各自的状态数的最大值，即$N_{\max}=\max N_k(1\leqslant k\leqslant n)$。容器排序时间复杂度为$O(n\,\mathrm{lb}\,n)$，宿主机排序复杂度为$O(m\,\mathrm{lb}\,m)$，部署一台容器的复杂度为$O(mN_{\max}^d)$，部署$n$台的复杂度为$O(nmN_{\max}^d)$。而容器部署算法总的复杂度由部署时间所决定，即$O(nmN_{\max}^d)$。$N_{\max}$一般是比较小的常数，$d$是确定的整数，故$N_{\max}^d$只是一个系数，算法的复杂度为$O(nm)$。然而，当实际运行中$nm$的规模不够大时，$N_{\max}^d$是一个相当大的系数，直接导致部署时间延长。为了缩短部署时间，在3.3节将提出一种估算最小资源预留量的预判方法。

算法2 面向动态负载的容器部署(MultiCons)。

输入 容器总数n，宿主机总数m；单宿主机允许部署最大容器数d；容器的马尔可夫模型$\boldsymbol{V}=\{V(1),V(2),\cdots,V(n)\}$；宿主机模型$\boldsymbol{H}=\{H_1,H_2,\cdots,H_m\}$。

输出 容器到宿主机的映射$\boldsymbol{X}=\left[X_{kl}\right]_{n\times m}$。

```
将V按R(k)·μ(k)降序排序;
将H降序排序;
for k = 1 to n
  for l = 1 to m                         //V(H_l)表示已放在H_l上的容器
    if De({V(H_l), V(k)}, H_l) = 1
      X_kl = 1;
      break;
    endif
  endfor
endfor
返回X = [X_kl]_{n×m}
```

3.3 面向动态负载的容器部署加速

在算法2中，如果前面l台宿主机上已经部署了比较多(不妨假设为$d-1$台)容器，其将不再容纳任何有较大资源需求的其他容器。但算法2在部署一台明显无法放置在这l台宿主机上的容器时，仍需要从第一台宿主机开始尝试，直到付出了$O(lN_{\max}^d)$的代价，才判断出前l台宿主机都无法容纳。显

然,这种情况是对计算的极大浪费。本文考虑寻找一个可以简单计算出的参数,来判断容器是否有很大概率无法部署在特定宿主机上。本文先考虑所有容器的资源需求为独立的二项分布,如表1所示。

表1　容器的资源分布

Tab. 1　Resource distribution of dockers

资源需求	概率
R_1	ρ
R_2	$1-\rho$

正态分布的累积概率为 $F_z(z_\alpha)=P(Z\leqslant z_\alpha)=1-\alpha$。现有一台宿主机上已部署 k 台容器,需要判断第 $k+1$ 台容器能否部署在该宿主机上。用 X 表示这 $k+1$ 台容器的总资源需求量,即 $X=X_1+X_2+\cdots+X_{k+1}$,根据中心极限定理,有:

$$P\left[X\leqslant R_q\right]=P\left[X_1+X_2+\cdots+X_{k+1}\leqslant R_q\right]\approx F_z\left(\frac{R_q-E(X)}{\sqrt{Var(X)}}\right)=1-\rho$$

本文分别用 $X_{\min}$、$X_{\max}$ 表示 X_k 的最小值和最大值,进而得到:

$$\begin{aligned}E(X)&=\sum_{i=1}^{k+1}\left(R_1(i)\,p+R_2(i)\left(1-p\right)\right)=\\&p\sum_{i=1}^{k+1}R_1(i)+\left(1-p\right)\sum_{i=1}^{k+1}R_2(i)=\\&pX_{\min}+(1-p)X_{\max}\end{aligned}\tag{10}$$

$$\begin{aligned}Var(X)&=\sum_{i=1}^{k+1}Var\left(X_i\right)=\sum_{i=1}^{k+1}\left(E\left(X_i^2\right)-E^2\left(X_i\right)\right)=\\&p\left(1-p\right)\sum_{i=1}^{k+1}\left(R_2(i)-R_1(i)\right)^2\approx\\&\frac{p\left(1-p\right)}{k+1}\left(\sum_{i=1}^{k+1}\left(R_2(i)-R_1(i)\right)\right)^2=\\&\frac{p\left(1-p\right)}{k+1}\left(X_{\max}-X_{\min}\right)^2\end{aligned}\tag{11}$$

不仅如此,当 $\xi=z_\rho\sqrt{\dfrac{p\left(1-p\right)}{k+1}}$ 时,有:

$$\begin{aligned}R_q&=z_\rho\sqrt{Var(X)}+E(X)\approx\\&\left(1-p+\xi\right)X_{\max}+\left(p-\xi\right)X_{\min}\end{aligned}\tag{12}$$

当 $R_q>H_l$ 时,可以判断第 $k+1$ 台容器无法放置在宿主机 l 上。这里 R_q 的计算有两个假定:1)所有的容器服从相似的概率分布。由于FFD将所有容器按照其资源需求量的期望排序,所以放置在同一台宿主机上的大多数容器具有相似的期望。在实际情况下,可以将容器按分布聚簇,再排序部署,使其更加准确。2)所有容器都是二状态的。在实际情况中,容器状态数会大于2,$X_{\max}$ 会导致实际结果偏大,在 $X_{\max}$ 前加上修正 α,得:

$$R_q\approx\alpha\left(1-p+\xi\right)X_{\max}+\left(p-\xi\right)X_{\min}\tag{13}$$

对于修正系数 α,可以利用容器的分布估算。先考虑只有一台的情况:对于二状态的容器,易知 $\alpha=1$。对于 N 状态的容器($N\geqslant 3$)有:

$$\alpha=\frac{1}{R_N}\cdot\frac{\sum_{k=2}^{N}R_kp_k}{\sum_{k=2}^{N}p_k}=\frac{1}{R_N}\cdot\frac{\sum_{k=2}^{N}R_kp_k}{\left(1-p_1\right)}\tag{14}$$

当有 n 台容器具有不一样的状态数时,有:

$$\alpha=\frac{\sum_{i=1}^{n}R_2'(i)}{\sum_{i=1}^{n}R_N(i)};\ R_2'=\frac{\sum_{k=2}^{N}R_kp_k}{\left(1-p_1\right)}\tag{15}$$

以式(14)、(15)计算出的 α 作为参考值,若在当前 α 下算法运行结果准确率比较低时,可以通过减小 α 来提高准确率;当运行时间较长且准确率较高时,可以通过增大 α 来缩短运行时间。据此,提出面向动态负载的容器加速部署策略(如算法3所示),该算法能够结合宿主机和容器各自不同的状态进行动态部署。

这里提出用中心极限定理估算 R_q 方法的合理性在于:一般来说,若 $n\geqslant 9\times\max\left(\dfrac{p}{1-p},\dfrac{1-p}{p}\right)$,中心极限定理就可以得到较好的拟合效果。在实际的部署中,当容器的数量比较小时,资源需求量和计算出的 R_q 都远小于 $\boldsymbol{H}$,所以估算效果的好坏不影响结果;但当容器的数量比较大时,这种方法的估算效果好,得到比较准确的结果。

算法3　容器的加速部署(QuickMultiCons)。

输入　容器总数 n;宿主机总数 m;单宿主机允许部署最大容器数 d;容器的马尔可夫模型 $\boldsymbol{V}=\{V(1),V(2),\cdots,V(n)\}$;宿主机模型 $\boldsymbol{H}=\{H_1,H_2,\cdots,H_m\}$。

输出　容器到宿主机的映射 $\boldsymbol{X}=\left[X_{kl}\right]_{n\times m}$。

```
将V按R(k)·μ(k)降序排序;
将H降序排序;
for k = 1 to n
  for l = 1 to m
    if 宿主机l上已有d台容器
      break;
    endif
    if R_q > H_l                          //由式(13)计算所得
      continue;
    endif
    if De({V(H_l), V(k)}, H_l) = 1
    // V(H_l)表示已放在H_l上的容器
      X_kl = 1;
      break;
    endif
  endfor
endfor
返回 X = [X_kl]_{n×m}
```

4　实验与结果分析

4.1　实验概述

为验证部署策略的有效性,在基于国产软硬件环境的基础上,结合开源系统,设计了一系列实验,验证不同参数配置下所提设计算法的效果。

4.1.1 测试床环境

基于国产软硬件环境及开源系统，本文部署并测试环境运行的可行性。主要实验环境配置及具体参数包括：测试床服务器为浪潮SN5160M4（IntelE5-2680V4 CPU*2、256 GB（8*32 GB RDIMMG）DDR4内存、1.2 TB热插拔SAS硬盘2.5"*6）；系统为中标麒麟（kernel-2.6.32；KVM-0.12）；资源管理系统为OpenStack R版本；语言环境为Python 3.8。

4.1.2 对比算法

在测试床的基础上，通过模拟程序行为实现了所提容器部署策略MULTI（MULTIple），并与三种常用的容器部署策略进行了比较：1）按谷值配置容器的策略RV（Resource with Valley）；2）按峰值配置容器的策略RP（Resource with Peak）；3）按谷值峰值的平均配置容器的策略RVP（Resource with Valley and Peak）。三种部署策略也都使用了FFD策略。在部署完成后，按照真实容器的分布，产生1 000组实时数据进行仿真，出现冲突时测试床将容器动态迁移到其余空闲宿主机，记录运行性能结果。

4.1.3 验证指标

衡量部署策略的性能的参数包括部署一定台数容器所需的宿主机数（Physical Machines，PM）、冲突率（Capacity Violation Rate，CVR）和动态迁移次数（MIGration，MIG），其中*PM*、*CVR*、*MIG*分别是1 000组数据产生宿主机数目的最大值、冲突率平均值、动态迁移次数最大值。

4.1.4 实验参数

实验涉及的参数包括：容器数量（*VM*）、容器资源需求相对大小（$R_1:R_2:\cdots$）、容器资源需求的平稳分布概率（$P_1:P_2:\cdots$）、容器资源需求量的绝对大小（$R_{\text{mean}}=\sum_i R_iP_i$）、宿主机容量（*H*）、宿主机允许的最大容器数（*d*）以及宿主机的性能约束（ρ）。

实验主要验证容器资源需求的分布区间（$R_1:R_2:\cdots$）、绝对大小R_{mean}和平稳分布（$P_1:P_2:\cdots$）在不同配置下对部署效果的影响，得到部署策略最优的参数配置范围。实验中容器有50%概率为二状态，50%概率为三状态。宿主机的容量*H*为（80, 100）的均匀分布，*d*设为16，ρ设为0.04。三状态容器和二状态容器的资源需求量平稳分布分别如表2和表3所示，容器资源需求量分布如表4所示。表2的含义是对每一台容器，其P_1在0~*P*内等概率随机取值，P_2在0~$(1-P_1)$内等概率随机取值，P_3取$1-P_1-P_2$。表3和表4的含义类似。

表2 三状态容器的资源需求量平稳分布

Tab. 2 Resource demand stationary distribution of dockers with three states

P_1	P_2	P_3
0~*P*	0~$(1-P_1)$	$1-P_1-P_2$

表3 二状态容器的资源需求量平稳分布

Tab. 3 Resource demand stationary distribution of dockers with two states

P_1	P_2
0~*P*	1~*P*

累计进行了4组综合实验：

1）调节容器资源需求量的绝对大小：固定$P=1$，*R*分别取3~7，从而改变R_{mean}。

2）调节容器资源需求的平稳分布概率：分别在*P*等于0.2、0.4、0.6、0.8、1.0的情况下进行实验，并通过改变*R*固定$R_{\text{mean}}=6.125$。

3）调节容器资源需求的分布区间：固定$P=1$和$R_{\text{mean}}=6.125$，改变*R*值调节容器资源需求。

4）部署策略运行的时间代价：取$P=1$和$R=5$，将10 000台容器分成100批，每批100台，按批次进行部署，并记录下每批容器部署的时间。

表4 容器资源需求量分布

Tab. 4 Distribution of docker resource demand

R_1	R_2	R_3
0.4*R*~*R*	*R*~2*R*	2*R*~3*R*

4.2 模拟实验结果

实验1的结果如表5~6、图4所示。RV使用最少的宿主机，动态迁移次数最多。RP动态迁移为零，使用的宿主机最多。RVP使用较少的宿主机，但动态迁移次数较多，冲突率会随容器条件的变化而产生较大起伏。MULTI使用少于RP的宿主机数，并且保持较少的迁移次数和较小冲突率。RV与RVP在$R<5$时的冲突率很小，$R\geqslant 5$时更能体现MULTI算法的性能效果。在现实中，一般宿主机上都会布置10台以上容器[24]，所以模拟实验的结果在$R=5$时更贴近真实的情况。随着*R*增大，MULTI的*PM*增长速度大于RVP的*PM*增长速度，这是因为*R*增大会导致单台宿主机上放置容器数量减少，即式（10）的*k*减小，得到更大的R_q。

表5 *PM*与*R*的关系

Tab. 5 Relationship between *PM* and *R*

R	*PM*			
	RV	RP	RVP	MULTI
3	63	68	63	63
4	63	83	64	67
5	66	102	71	80
6	79	124	83	99
7	91	145	97	117

表6 *MIG*与*R*的关系

Tab. 6 Relationship between *MIG* and *R*

R	*MIG*			
	RV	RP	RVP	MULTI
3	0	0	1	3
4	8	0	13	5
5	61	0	29	7
6	130	0	42	7
7	194	0	44	6

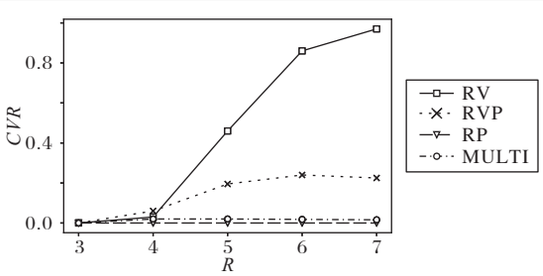

图4 *CVR*与*R*的关系

Fig. 4 Relationship between *CVR* and *R*

实验2的结果如表7~8、图5所示。随着P的增大,MULTI策略下的PM略有增大,但幅度比较小。增大的原因是:根据式(10)、(12),资源预留量R_q与容器资源需求的期望和方差有关,当保持R_{mean}恒定时,R_q的大小由容器资源需求的方差大小决定,经过简单计算可知基于表4的分布,P越大,容器资源需求的方差也越大,所需要的资源预留量越大,进而所使用的宿主机数量越多。RV、RP、RVP三种算法的PM增大是因为R的增大。RV、RVP的MIG和CVR都相当地大且不稳定,MULTI的MIG一直保持在较小数值7,CVR也均保持在0.04以下,意味着当容器需要资源扩张的时候,其与宿主机的冲突率一直保持较低水平,也能够体现在宿主机上提前预留资源的好处。

表7　PM与P的关系

Tab. 7　Relationship between PM and P

P	PM			
	RV	RP	RVP	MULTI
0.2	64	79	64	72
0.4	64	82	64	74
0.6	64	88	65	76
0.8	65	94	68	78
1.0	66	102	71	80

表8　MIG与P的关系

Tab. 8　Relationship between MIG and P

P	MIG			
	RV	RP	RVP	MULTI
0.2	44	0	66	6
0.4	47	0	57	7
0.6	49	0	47	7
0.8	50	0	35	7
1.0	61	0	29	7

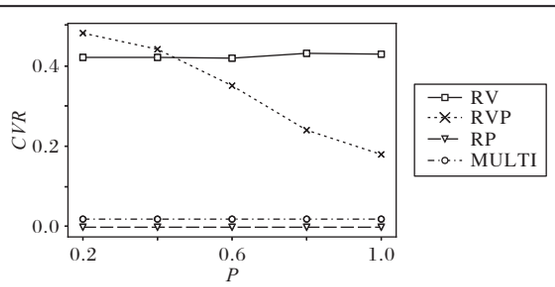

图5　CVR与P的关系

Fig. 5　Relationship between CVR and P

实验3的R分别取3、5、7时,资源需求量的分布如表9所示。实验结果如表10~11,四种算法的MIG和CVR都比较稳定,RVP的MIG与MULTI的MIG保持着4倍左右的差距。

表9　P不变时资源需求量的分布

Tab. 9　Distribution of Resource Demand with fixed P

R_1	R_2	R_3
2~3	3~12	12~21
2~5	5~10	10~15
2~7	7~8	8~9

实验4的结果如图6所示,部署时间与当前已放置的容器数量呈线性关系,当已部署的容器数量过大时,部署时间也会变得非常长,表明容器迁移开销也相应变大。

综上所述,实验1体现了容器资源需求分布的期望对部署结果的影响,实验2、3用两种方式调节了容器资源需求分布的方差并研究了它对部署结果的影响,进而一方面也验证了式(10)、(11)、(12)结论的可靠性。面向多状态负载的容器部署策略的最大优点在于能够保持动态迁移次数和冲突率均稳定在一个较小的范围。

表10　PM与R的关系

Tab. 10　Relationship between PM and R

R	PM			
	RV	RP	RVP	MULTI
3	66	123	78	91
5	66	102	71	80
7	64	84	66	72

表11　MIG与R的关系

Tab. 11　Relationship between MIG and R

R	MIG			
	RV	RP	RVP	MULTI
3	54	0	27	6
5	61	0	29	7
7	57	0	21	6

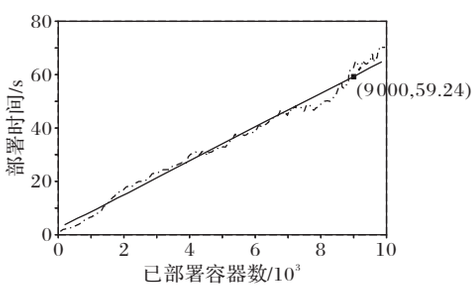

图6　部署时延与部署容器数量的关系

Fig. 6　Relationship between deployment delay and number of deployed dockers

5　结语

容器技术在集群中发挥着重要作用,为用户提供按需式的资源服务。然而,传统被动的容器整合或是迁移无法应对动态的容器负载,给集群容器部署带来挑战。本文针对长效时间上容器整合与迁移的整体优化,提出了利用多状态马尔可夫链模型描述容器资源状态,并在宿主机资源预留的基础上提出了部署策略,还进一步优化部署的时延。实验结果表明,本文提出的策略能够在宿主机使用数量和动态容器迁移次数间达到较好的平衡。本文主要针对静态场景下的动态部署问题进行了探讨,但所提出的方法难以适用于在线场景,且资源需求类型的描述一定程度上影响了资源的分配策略,而资源需求的类型数量难以确定,这两方面将是下一步研究的重点方向。

参考文献(References)

[1] XIE G, ZENG G, LI R, et al. Quantitative fault-tolerance for reliable workflows on heterogeneous IaaS clouds [J]. IEEE Transactions on Cloud Computing, 2020, 8(4): 1223-1236.

[2] SHARIFFDEEN R S, MUNASINGHE D T S P, BHATHIYA H S, et al. Workload and resource aware proactive auto-scaler for PaaS cloud [C]// Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing. Piscataway: IEEE, 2016: 11-18.

[3] KLIMOVIC A, WANG Y, STUEDI P, et al. Pocket: elastic ephemeral storage for serverless analytics [C]// Proceedings of the

2018 13th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2018: 427-444.

[4] YI B, WANG X, LI K, et al. A comprehensive survey of network function virtualization [J]. Computer Networks, 2018, 133: 212-262.

[5] HUANG Z, WU S, JIANG S, et al. FastBuild: accelerating docker image building for efficient development and deployment of container [C]// Proceedings of the 2019 35th Symposium on Mass Storage Systems and Technologies. Piscataway: IEEE, 2019: 28-37.

[6] AJIRO Y, TANAKA A. Improving packing algorithms for server consolidation [C]// Proceedings of the 2007 33rd International Computer Measurement Group Conference. Sicklerville: CMG, 2007: 399-406.

[7] RAMPERSAUD S, GROSU D. Sharing-aware online virtual machine packing in heterogeneous resource clouds [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(7): 2046-2059.

[8] ARDELEAN D, DIWAN A, ERDMAN C. Performance analysis of cloud applications [C]// Proceedings of the 2018 15th USENIX Symposium on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2018: 405-417.

[9] CORTEZ E, BONDE A, MUZIO A, et al. Resource central: understanding and predicting workloads for improved resource management in large cloud platforms [C]// Proceedings of the 2017 26th Symposium on Operating Systems Principles. New York: ACM, 2017: 153-167.

[10] LIU L, XU H. Elasecutor: elastic executor scheduling in data analytics systems [C]// Proceedings of the 2018 ACM Symposium on Cloud Computing. New York: ACM, 2018: 107-120.

[11] SILVA FILHO M C, MONTEIRO C C, INÁCIO P R M, et al. Approaches for optimizing virtual machine placement and migration in cloud environments: a survey [J]. Journal of Parallel and Distributed Computing, 2018, 111: 222-250.

[12] VMware. Distributed resource scheduler and management [EB/OL]. [2020-06-18]. https://www. vmware. com/products/vsphere. html.

[13] IBM. IBM system planning tool for POWER processor-based systems [EB/OL]. [2020-06-17]. https://www. ibm. com/support/pages/node/632621.

[14] WANG M, MENG X, ZHANG L. Consolidating virtual machines with dynamic bandwidth demand in data centers [C]// Proceedings of the 2011 IEEE International Conference on Computer Communications. Piscataway: IEEE, 2011: 71-75.

[15] BREITGAND D, EPSTEIN A. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds [C]// Proceedings of the 2012 IEEE International Conference on Computer Communications. Piscataway: IEEE, 2012: 2861-2865.

[16] RODRIGUES T G, SUTO K, NISHIYAMA H, et al. Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control [J]. IEEE Transactions on Computers, 2017, 66(5): 810-819.

[17] TSAKALOZOS K, VERROIOS V, ROUSSOPOULOS M, et al. Live VM migration under time-constraints in share-nothing IaaS-clouds [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(8): 2285-2298.

[18] LE F, NAHUM E M. Experiences implementing live VM migration over the WAN with multi-path TCP [C]// Proceedings of the 2019 IEEE International Conference on Computer Communications. Piscataway: IEEE, 2019: 1090-1098.

[19] MI N, CASALE G, CHERKASOVA L, et al. Injecting realistic burstiness to a traditional client-server benchmark [C]// Proceedings of the 2009 6th International Conference on Autonomic Computing. New York: ACM, 2009: 149-158.

[20] BAARZI A F, ZHU T, URGAONKAR B. BurScale: using burstable instances for cost-effective auto-scaling in the public cloud [C]// Proceedings of the 2019 ACM Symposium on Cloud Computing. New York: ACM, 2019: 126-138.

[21] YU L, CHEN L, CAI Z, et al. Stochastic load balancing for virtual resource management in datacenters [J]. IEEE Transactions on Cloud Computing, 2020, 8(2): 459-472.

[22] JIN H, PAN D, XU J, et al. Efficient VM placement with multiple deterministic and stochastic resources in data centers [C]// Proceedings of the 2012 IEEE Global Communications Conference. Piscataway: IEEE, 2012: 2505-2510.

[23] ZHANG S, QIAN Z, LUO Z, et al. Burstiness-aware resource reservation for server consolidation in computing clouds [J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(4): 964-977.

[24] ZHAO N, ALBAHAR H, ABRAHAM S, et al. DupHunter: flexible high-performance deduplication for docker registries [C]// Proceedings of the 2020 USENIX Annual Technical Conference. Berkeley: USENIX Association, 2020: 769-783.

[25] 李启锐,彭志平,崔得龙,等. 容器云环境虚拟资源配置策略的优化[J]. 计算机应用,2019,39(3):784-789. (LI Q R, PENG Z P, CUI D L, et al. Optimization of virtual resource deployment strategy in container cloud [J]. Journal of Computer Applications, 2019, 39(3): 784-789.)

This work is partially supported by the National Natural Science Foundation of China (61802182).

YIN Fei, born in 1978, senior engineer. His research interests include electric power informatization, cloud computing.

LONG Lingli, born in 1984, M. S., engineer. Her research interests include electric power informatization, cloud computing.

KONG Zheng, born in 1986, assistant engineer. His research interests include electric power informatization, cloud computing.

SHAO Han, born in 1994, Ph. D. candidate. Her research interests include distributed system, computer network.

LI Xin, born in 1987, Ph. D., associate professor. His research interests include cloud computing, edge computing.

QIAN Zhuzhong, born in 1980, Ph. D., professor. His research interests include distributed system, data center network.