文章编号：2095-2163(2020)04-0050-05 中图分类号：TP399 文献标志码：A

代价敏感的容器多目标资源放置优化算法

曹成成，李志聪
（哈尔滨师范大学 计算机科学与信息工程学院，哈尔滨 150025）

摘 要：自 Docker 问世以来，微服务也得到了快速的发展，企业、组织等纷纷使用微服务架构进行容器化开发。为了管理数以万计的容器应用，各种容器编排框架应运而生，但容器调度过程中带来的能耗高、资源利用率低等问题非常显著。研究合理的容器放置，能有效的减缓此类问题。针对 CPU、内存和带宽三类资源利用率低等问题，提出了容器多目标资源放置算法 CMR（Container Multi-target Resource）。实验结果证明 CMR 算法能够将容器放置到与自身资源请求大小最吻合的虚拟机上，对比 FF、LF、MF 和 RS 算法能够同时节省 CPU 能耗 34.0%，内存能耗 33.8%，带宽能耗 26.5%。

关键词：容器；多目标；资源利用率；能耗

Research on the Optimization algorithm of multi-target resource placement in cost-sensitive container

CAO Chengcheng, LI Zhicong
(School of Computer Science Technology and Information Engineering, Harbin Normal University, Harbin 150025, China)

【Abstract】 Since the advent of Docker, microservices have developed rapidly, and enterprises and companies have gradually begun to use the microservices architecture for container development. In order to manage tens of thousands of container applications, various container orchestration frameworks have emerged, which brings about high energy consumption and low resource utilization in the container scheduling process. Reasonable container placement can effectively alleviate such problems. We propose a container multi-target resource CMR (Container Multi-target Resource) for the low utilization of CPU, memory and bandwidth. The experimental results show that the CMR algorithm can place the container on the virtual machine that best matches the size of its own resource request. Compared with the FF, LF, MF and RS algorithms, it can save CPU power consumption by 34.0%, memory energy consumption by 33.8%, and bandwidth energy consumption by 26.5%.

【Key words】 Container; Multi-objective; Resource utilization; Power consumption

0 引 言

随着云计算的发展，容器作为一种新的虚拟化技术，愈加得到重视，尤其是 Docker 的出现更是让容器成为重要研究课题之一[1-3]。目前容器的编排技术仍存在资源利用率低，能耗较高的问题。

由于在云环境中，CPU 的能耗影响最大，所以大多数容器编排框架在容器的调度过程中都只考虑了 CPU 的能耗，导致在一个异构的云环境中，容器请求的资源之间差异较大，仅考虑 CPU 的利用率将会产生大量的资源浪费。本文提出了一种多目标资源 CMR（container multi-target resource）容器放置算法，根据虚拟机 CPU、内存和带宽的资源属性值大小对虚拟机进行分类，把虚拟机分为计算型、内存型和带宽型 3 类，对容器也进行以上 3 类划分，算出容器最佳放置位置。

1 相关工作

早期主要研究虚拟机与物理主机之间的协作关系，由于 Docker 的盛行，容器与虚拟机在不同环境下的性能比较，容器的管理、应用等成为研究方向，研究独立的 Docker 容器相对于独立虚拟机的性能开销情况，结果表明管理容器技术上可能会带来额外的开销[4]；有研究者采用新型的渗透算法研究微服务从云到雾、边缘和物联网（IoT）环境的迁移、部署和优化等[5]。

有好多高可用的容器集群[6]，如 Mesos 和 Docker Swarm，其部署方式主要根据主机 CPU 的利用率进行 spread、binpack 和 random 等策略进行放置，Kubernetes 作为目前容器编排框架的主流框架在编排过程中也只考虑到 CPU 和内存的使用情况。早在虚拟机放置算法中，就有文献介绍了基于能耗优化的多目标放置算法[7]。将多种因素加入到容器的编排以减少能耗优化成为一个研究方向。

本文使用 ContainerCloudSim 为容器云计算环境的建模和仿真提供了支持[8]。ContainerCloudSim 中

基金项目：国家自然科学基金项目（61202458，61403109）；黑龙江省自然科学基金项目（F2017021）。

作者简介：曹成成（1994-），男，硕士研究生，主要研究方向：云计算；李志聪（1972-），男，硕士，副教授，主要研究方向：云计算、人工智能、三支决策。

收稿日期：2020-02-28

的 MostFull、ListFull 和 RandomSelection 对应 Docker Swarm 调度的 3 种策略：spread、binpack 和 random。

2　CMR 容器放置算法

由于虚拟机带来的隔离机制，云环境采用在物理主机上搭建虚拟机，在虚拟机上搭建容器平台的混合虚拟化建模方式，解决数据中心能耗高以及 CPU、内存和带宽三类资源总体利用率低等问题。

2.1　数据中心模型

定义 1　主机列表 $Host = \{h_1, h_2, \cdots, h_i, \cdots, h_n\}$，其中一共有 n 个主机，h_i 表示第 i 个主机，$h_i = \{Cpu_i^h, Ram_i^h, Bw_i^h\}$，表示主机包含 3 维资源，分别是 CPU、内存和带宽，定义 h_i^j 为主机 i 上的第 j 维资源的大小，$j = 1,2,3$；分别表示为 $h_i^1 = Cpu_i^h$，$h_i^2 = Ram_i^h$，$h_i^3 = Bw_i^h$。

定义 2　数据中心 *Datacenter* 资源计算如公式(1)所示。

$$Datacenter = \sum_{i=1}^{n} h_i = \sum_{i=1}^{n} \{Cpu_i^h, Ram_i^h, Bw_i^h\}. \quad (1)$$

定义 $Datacenter^j$ 为数据中心第 j 维资源大小，$j = 1,2,3$；即 $Datacenter^1 = \sum_{i=1}^{n} Cpu_i^h$，$Datacenter^2 = \sum_{i=1}^{n} Ram_i^h$，$Datacenter^3 = \sum_{i=1}^{n} Bw_i^h$。

定义 3　容器 $C = \{c_1, \cdots, c_t, \cdots, c_q\}$，表示一共有 q 个容器，c_t 表示第 t 个容器，且 $c_t = \{Cpu_t^c, Ram_t^c, Bw_t^c\}$，$c_t^j$ 为容器 c_t 上第 j 维资源，$j = 1,2,3$；分别表示容器 c_t 的 CPU、内存、带宽的资源大小。

定义 4　虚拟机 $VM = \{vm_1, \cdots, vm_k, \cdots vm_m\}$，表示一共有 m 台虚拟机，其中 vm_k 为第 k 个虚拟机，且 $vm_k = \{Cpu_k^v, Ram_k^v, Bw_k^v\}$，$vm_k^j$ 为第 k 个虚拟机上的第 j 维资源，$j = 1,2,3$；分别表示虚拟机 vm_k 的 CPU、内存、带宽的资源大小。

每个任务都与容器绑定，每个容器都会被放置到一台合适的虚拟机上运行。考虑到多目标的资源利用率，需要将容器放置到虚拟机剩余资源与容器资源最为吻合的虚拟机上。为了解决以上问题，提出基于容器多目标资源利用率的放置算法来降低能耗。

2.2　CMR 算法描述

传统的放置算法中，只考虑在 CPU 利用率情况下的能耗计算，可能会导致资源利用不均衡、资源浪费。本文提出一种 CMR 算法，考虑 CPU 利用率、内存(RAM)和带宽(BW)三种因素，对比虚拟机的剩余资源和容器的请求资源，选择资源最匹配的虚拟机放置容器。CMR 算法的具体过程：

步骤 1　虚拟机分类

根据数据中心的整体属性，将虚拟机分为三类，即 CPU 型、内存型和带宽型。由于 CPU、内存和带宽的资源属性值各不相同，把虚拟机的剩余资源与数据中心对应资源属性值做商，商值越大，说明资源占比重，故把商值最大的作为该虚拟机的类型。

定义 6　虚拟机 vm_k 的第 j 维资源剩余可分配空间 $space_{k,j}$，如公式(2)所示。

$$space_{k,j} = allocable_k^j - allocated_k^j. \quad (2)$$

其中 $allocable_k^j$ 表示 vm_k 最大可分配的 j 维资源，$allocated_k^j$ 表示 vm_k 已分配的 j 维资源。

定义 7　虚拟机 vm_k 分类的指标 $VS_{k,j}$，其中 j 表示虚拟机资源维数，当 $j = 1,2,3$ 时分别表示为 CPU、内存和带宽，虚拟机的总数为 m，如公式(3)所示。

$$VS_{k,j} = \frac{space_{k,j}}{Datacenter^j}. \quad (3)$$

其中 $Datacenter^j$ 为数据中心的 j 维资源大小，首先分别计算当 j 等于 1,2,3 时的分类指标 $VS_{k,j}$，其次比较 $VS_{k,1}, VS_{k,2}, VS_{k,3}$，最大值的 j 资源属性作为 vm_k 的类型。例如：$VS_{k,3}$ 为最大值，则 $j = 3$，即 vm_k 的类型为带宽型虚拟机。

步骤 2　容器分类

容器资源同样有 3 个维度，分别是 CPU、内存和带宽。同样与数据中心的各个资源属性值做比，根据比值来判断容器属于哪一类。

定义 8　容器 c_t 分类的指标 $CS_{t,j}$，如公式(4)所示。

$$CS_{t,j} = \frac{c_t^j}{Datacenter^j}. \quad (4)$$

$CS_{t,j}$ 表示 c_t 的第 j 维属性与 Datacenter 的第 j 维属性的比值，比较 $CS_{t,1}, CS_{t,2}, CS_{t,3}$，选择出最大的值，其资源属性作为 c_t 的类型。

步骤 3　容器放置

虚拟机 vm_k 和容器 c_t 都完成类型分类后，放置容器到同类型的虚拟机上。分别在 $j = 1,2,3$ 时，计算 c_t 放置到 vm_k 的属性适应值列表为 $Value_{k,j}$，如公式(5)所示。

$$Value_{k,j} = \frac{c_t^j}{vm_k^j}, k = 1,2,..,m'. \quad (5)$$

其中 m' 表示与 c_t 相同类型 vm_k 的个数。计算容器放置到 vm_k 上的总适应值 SV_k，如公式(6)

所示。

$$SV_k = \sum_{f=1}^{3} Value_{k,j}. \quad (6)$$

取得 SV_k 最大值所对应的 k' 值，即 $\{k': \max(SV_k), k' = 1,2,\cdots,m'\}$，则该 $vm_{k'}$ 作为容器 c_t 将要放置的虚拟机。

经过以上三步骤计算后，选出与容器资源匹配度最高的虚拟机进行容器放置，充分使容器和虚拟机的剩余资源相契合，达到 CPU、内存、带宽充分利用的效果。资源充分利用，从而降低能耗。

算法 CMR 描述：

输入 q 个容器 $C = \{c_1,\cdots,c_t,\cdots,c_q\}$，$m$ 个虚拟机 $VM = \{vm_1,\cdots,vm_k,\cdots vm_m\}$。

输出 容器 c_t 放置到虚拟机 vm_k 的列表：$list = \{ < c_1,vm_a >, < c_2,vm_b >,6\cdots, < c_t,vm_i >,\cdots, < c_q,vm_j > \}$。

步骤 1 初始化容器位置标识 $t = 1$，$list = null$；

步骤 2 $k = 1$，计算 vm_k 的剩余资源 $space_{k,j}$，获得虚拟机分类指标 $VS_{k,j}$，$k = k + 1$，重复步骤 2，直到 $k = m$；

步骤 3 初始化 $SV_k = 0$，$maxSV = 0$，$selectVM = null$，虚拟机位置标识 $k = 1$；

步骤 4 根据 $VS_{k,j}$ 虚拟机分类，获得虚拟机分类列表 VM_{cpu}，VM_{ram}，VM_{bw}；

步骤 5 计算容器 c_t 分类指标 $CS_{t,j}$，判断容器类型；

步骤 6 根据容器类型选择同类型虚拟机列表 VM_z，$z = \{cpu \mid \mid ram \mid \mid bw\}$；

步骤 7 计算 SV_k 大小，如果 $SV_k > maxSV$，$maxSV = SV_k$，$selectVM = vm_k$；

步骤 8 如果 $k < m$，$k = k + 1$，返回步骤 4；

步骤 9 向 $list$ 中加入 $< c_t,selectVm >$；

步骤 10 如果 $t < q$，$t = t + 1$，转为步骤 2；

步骤 11 算法结束，输出 $list = \{ < c_1,vm_a >, < c_2,vm_b >,\cdots, < c_t,vm_i >,\cdots, < c_q,vm_j > \}$。

2.3 时间复杂度分析

CMR 算法的时间复杂度主要体现在容器的放置过程中，同样设容器的总数为 n，虚拟机的个数为 m，为每个容器选择最佳放置位置时，需要遍历一次虚拟机列表，对虚拟机进行分类。在计算容器与虚拟机的最佳适应度时，需要再遍历一次虚拟机列表。放置整个容器列表的迭代次数为 $O(n \times 2m)$，所以 CMR 算法的时间复杂度为 $O(N^2)$。

3 实 验

3.1 实验环境

实验使用云模拟工具包 CloudSim4.0 作为容器服务的提供者。基于 CloudSim4.0 中 Container 模块，扩展了容器放置策略进行实验，选用 3 组异构的服务器搭建数据中心。为评估上述算法，CPU 负载采用了来自 PlanetLab 的工作负载文件。这些负载文件为测试机器收集的随机选择的 10 天工作负载。每个容器被分配一个工作负载，该工作负载包含一天的 CPU 利用率数据，每 5 分钟报告一次。不失一般性，容器的内存、带宽大小根据一定的范围随机生成。服务主机和虚拟机配置如表 1 和表 2 所示。

其中，CPU 的能耗模型采用了与服务器配置相对应的功率模型，分别是 HP ProLiant ML110 G4、IBM server x3250、IBM server x3550[9]。

表 1 服务主机配置

Tab. 1 Service configuration

服务器类型	CPU 核数	每核 MIPS	内存/GB	带宽 Gbit/s	能耗模型
#1	2	1 860	4	1	HP ProLiant ML110 G4
#2	4	3 067	8	1	IBM server x3250
#3	12	2 933	12	1	IBM server x3550

表 2 虚拟机配置

Tab. 2 Virtual Machine Configuration

服务器类型	CPU 核数	每核 MIPS	内存 MB	带宽 Mbit/s
#1	1	930	256	80
#2	1	665	1024	150
#3	1	733	1024	100
#4	1	733	512	250

3.2 实验结论

利用三组实验来验证 CMR 算法，每组实验重复多次，取所有实验数据的平均值作为最终实验结果。

实验首先对比了 CMR 算法与 FistFit、MostFull、LeastFull、RandomSelection 四种算法在三组级别的数据中心的总能耗，如图 1 所示。其次分别对比了 CPU、内存和带宽随着数据中心量级的改变所带来的能耗变化情况，其中 CPU 能耗的比较如图 2 所示，内存成本的比较如图 3 所示，带宽成本的比较如图 4 所示。实验的 CPU 负载采用了来自 PlanetLab 的工作负载文件，内存和带宽采用了满载的策略。

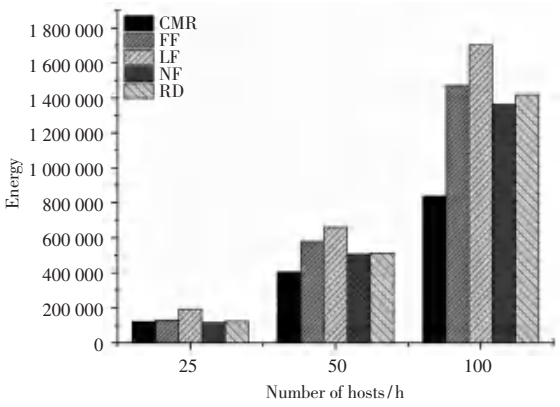

图 1　数据中心总能耗对比图

Fig. 1　Comparison chart of total energy consumption in data center

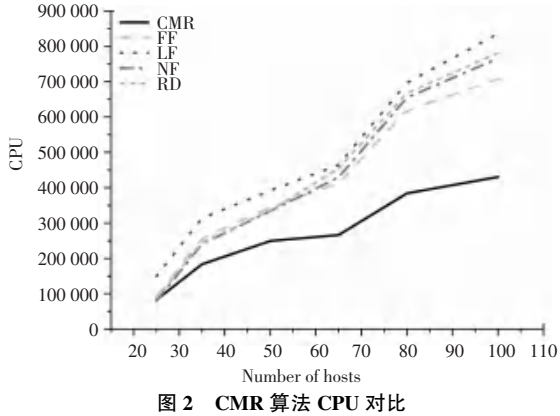

图 2　CMR 算法 CPU 对比

Fig. 2　CPU contrast of CIR

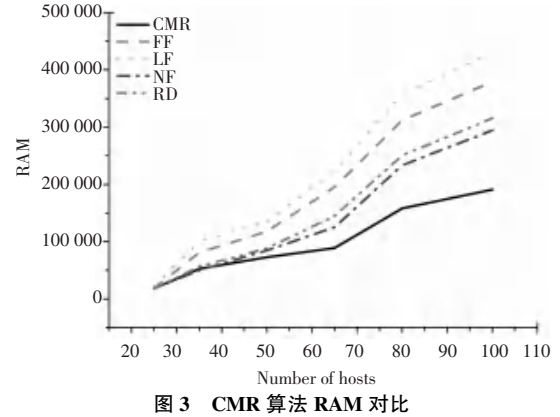

图 3　CMR 算法 RAM 对比

Fig. 3　RAM contrast of CIR

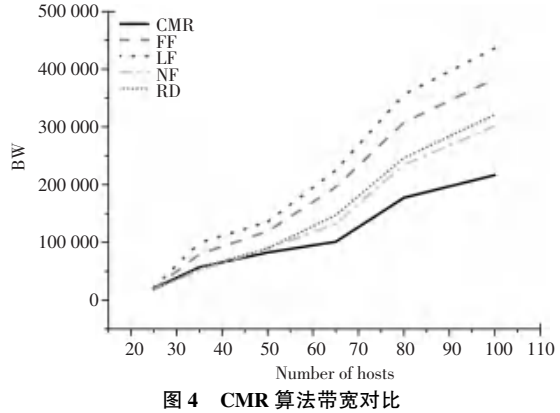

图 4　CMR 算法带宽对比

Fig. 4　BW Contrast of CIR

从图 1 可以看出，5 种算法第一组数据中心中能耗对比不太明显，在第 3 组数据中心中可以清晰的看出每个算法的优劣，其中 LF 算法的能耗最高，而 CMR 算法能耗最低，其他 3 种算法的总能耗较为均衡。CMR 算法在 3 组数据中心能耗的比较中，相比于其他四种算法能耗平均降低 28.6%，在数据中心主机达到 100 台时，CMR 算法相比其他算法能耗平均降低 43.6%。

从图 2 可以看出在 CPU 能耗方面 FF、MF、RS 这 3 种算法在不同级别的数据中心中的能耗都比较接近。FF 算法随着主机数的增加，能耗也有所降低，LF 能耗最高，而 CMR 算法比其他 4 个算法的 CPU 能耗都要低。单从 CPU 方面来看，能耗比其他四种算法平均降低了 34.0%。

在主机数量只有 25 台时，五种算法并没有明显的优劣，但随着主机数量的增加，CMR 的算法的优势便得以体现。在图 3 和图 4 的内存和带宽的能耗比较图中也可以看出，两种资源的变化曲线对比较为接近。在总体趋势上和 CPU 的能耗变化曲线也比较接近，随着数据中心主机数量的增加，能耗优化越明显。总体来说，CMR 算法相比于其他算法的内存平均能耗降低 33.8%，带宽能耗平均降低 26.5%。3 种不同类型的资源能耗同时降低也比较符合预期，达到了利用 CMR 算法提高三种资源的利用率来降低总体能耗的目的。

4　结束语

基于容器多目标资源的放置算法 CMR 对比 FF、LF、MF 和 RS 算法能够在充分利用数据中心 CPU、内存和带宽资源的同时，降低各个属性带来的能耗。从实验结果可以看出，CPU、内存和带宽三种资源的能耗都能得到降低，其中 CPU 资源能耗降低最为明显。

未来将在算法的优化、时间复杂度以及容器镜像复用等方面做更进一步的研究，并且将容器镜像复用、多目标的思想运用到容器的迁移等方面。

参考文献

[1] Docker-Build, Ship, and Run Any App, Anywhere, viewed 1 April 2019, URL http://www.docker.com.

[2] ARNOLDB, BASET S A, DETTORI P, et al. Building the IBM Containers cloud service [J]. IBM Journal of Research and Development, 2016, 60(2-3):9:1-9:12.

[3] COMBET, MARTIN A, DI PIETRO R. To Docker or Not to Docker: A Security Perspective [J]. IEEE Cloud Computing, 2016, 3(5):54-62.

[4] LI Z, KIHL M, LU Q, et al. Performance Overhead Comparison between Hypervisor and Container based Virtualization[J]. 2017.

（下转第 56 页）