计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(1):195−203 [doi: 10.15888/j.cnki.csa.008251]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于组合权重 TOPSIS 的 Kubernetes 调度算法①

张文辉, 王子辰

(桂林电子科技大学 计算机与信息安全学院, 桂林 541004)
通信作者: 王子辰, E-mail: wzcwangzichen@foxmail.com

摘　要: Kubernetes 是比较流行的开源容器编排引擎, 其默认调度算法只考虑了 CPU 和内存两种性能指标, 且采用统一权重计算候选节点得分, 无法满足各异的 Pod 应用需求. 本文在此基础上扩展了 Kubernetes 性能指标, 增加了带宽、磁盘、IO 速率 3 种指标, 并通过 AHP (analytic hierarchy process, 层次分析法) 计算主观权重和 EW (entropy weight, 熵权法) 根据 Pod 应用部署过程中节点的性能指标的资源利用率实时计算资源指标的客观权重. 两种权重相结合应用到改进的 TOPSIS (technique for order preference by similarity to an ideal solution, 逼近理想解排序方法) 多属性决策方法中来选择合适的候选节点. 实验结果表明, 随着部署 Pod 数量的增多, 在集群负载较大的情况下, 综合负载的标准差和 Kubernetes 默认调度算法相比提升 18%.

关键词: Kubernetes; 资源调度; 组合权重; 云计算; TOPSIS

引用格式: 张文辉,王子辰.基于组合权重 TOPSIS 的 Kubernetes 调度算法.计算机系统应用,2022,31(1):195–203. http://www.c-s-a.org.cn/1003-3254/8251.html

Kubernetes Scheduling Algorithm of TOPSIS Based on Combined Weight

ZHANG Wen-Hui, WANG Zi-Chen

(School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract: Kubernetes is a popular open-source container orchestration engine. Its default scheduling algorithm only considers CPU and memory and uses unified weight to calculate the score of candidate nodes, which cannot meet the requirements of different Pod applications. In view of this, the paper expands the Kubernetes performance indexes, with bandwidth, disk capacity, and IO rate added. The subjective weight is calculated by the analytic hierarchy process (AHP) and the objective weight of resource indexes is calculated by the entropy weight (EW) method in real time according to the resource utilization rate of performance indexes of nodes in the Pod application deployment process. We combine the two weights and apply them to a multi-attribute decision algorithm based on the improved technique for order preference by similarity to an ideal solution (TOPSIS) to select appropriate candidate nodes. The experiment results show that with the increase in the deployed Pod number, the standard deviation of the integrated load increases by 18% compared with that of the Kubernetes default scheduling algorithm under the condition of a large cluster load.

Key words: Kubernetes; resource scheduling; combined weight; cloud computing; TOPSIS

1　概述

近几年, 随着云计算技术的发展, 在云上部署应用服务有利于降低成本和提高效率, 但却引出了资源利用率低、部署和重启应用服务等待时间长等问题, 而 Docker 作为容器技术的代表, 为应用程序提供了更加轻量级和易于部署的解决方案, 如今已是云计算领域

① 基金项目: 国家自然科学基金 (61966007); 认知无线电与信息处理教育部重点实验室项目 (CRKL180201, CRKL180106); 广西无线宽带通信与信号处理重点实验室项目 (GXKL0619204, GXKL06200116)
收稿时间: 2021-03-18; 修改时间: 2021-04-16; 采用时间: 2021-04-26; csa 在线出版时间: 2021-12-17

的热点.

Kubernetes 是容器编排技术的代表, 是 Google 公司 Borg 项目的一个开源版本, 它基于 Docker 容器技术, 目的是实现资源管理的自动化, 以及跨多个数据中心的资源利用率最大化. Kubernetes 具有完备的集群管理能力, 包括多层次的安全防护和准入机制、多租户应用支撑能力、透明的服务注册和服务发现机制、可扩展的资源调度机制等[1]. 在 Kubernetes 集群中, 以 Pod 作为资源调度的基本单位, 而 Pod 中可以容纳多个容器, 集群根据其生命周期进行管理, 满足了产品内运行程序的需求, Kubernetes 已成为大规模容器化应用程序部署的事实标准[2]. 近年来, 大多数互联网公司的相关技术人员相继把 Kubernetes 运行在重要业务上, 同时越来越多的微服务也使用 Kubernetes 进行部署和管理.

Kubernetes 资源调度是其核心的模块, 不合理的资源调度策略会造成云平台整体资源利用率低, 用户的部署请求得不到快速响应, 服务质量下降, 同时也提高服务商的软硬件设施成本.

Kubernetes 调度策略首先根据用户提交的 Pod 应用的最小资源需求过滤掉不符合要求的节点, 其次以节点剩余 CPU 和内存的资源利用率为评分指标, 对候选节点评分, 选择得分最高的节点进行部署. 该策略存在两处不足: (1) 只考虑了 CPU 和内存两种资源指标, 无法满足各异的应用需求; (2) 采用相同权重计算候选节点得分, 易造成资源使用的过度倾斜.

关于 Kubernetes 资源调度问题, 已经有学者在优化节点负载、提高集群资源利用率和减少资源成本等方面做了大量研究. Menouer 等[3] 在 Docker Swarm 中利用 TOPSIS 多属性决策方法结合 Spread 和 Bin Packing 原理优化了多个资源指标下容器的调度问题; El Haj Ahmed 等[4] 通过容器应用的时间线和执行的历史信息来优化容器应用的部署; Imdoukh 等[5] 提出了一种基于多目标遗传算法进行优化的调度算法 MOGAS (many-objective genetic algorithm scheduler), 该调度算法与基于蚁群算法优化的调度算法相比较效果更佳; Zhang 等[6] 通过结合蚁群算法和粒子群优化算法改进了 Kubernetes 调度模型, 不仅减少了总的资源成本和节点最大负载, 也使得应用部署更加平衡; 孔德瑾等[7] 提出一种基于资源利用率进行指标权重自学习的调度机制, 提升了集群资源均衡度和资源综合利用率; Li 等[8] 提出 BDI (balanced-disk-IO-priority) 算法来动态调度容器应用以改善节点间磁盘 I/O 平衡, 提出 BCDI (balanced-CPU-disk-IO-priority) 算法动态调度容器应用以解决单节点 CPU 和磁盘 I/O 负载不平衡的问题; 吴双艳[9] 通过灰色预测算法对资源负载进行预测, 对容器云平台弹性伸缩系统优化来提高服务质量; Dua 等[10] 提出一种可供选择的任务调度算法, 为特定类型任务打上标签, 并基于标签将任务迁移实现集群负载均衡; Zheng 等[11] 提出一种基于 Docker 集群的自定义 Kubernetes 调度器调度策略, 使用优化的预选算法模型和优选算法模型改进 Kubernetes 默认调度策略, 提高集群调度的公平性和调度效率.

虽然国内外学者针对 Kubernetes 资源调度的研究已经取得了较多的成果, 但在异构环境下兼顾资源指标权重和资源指标本身都有些欠缺. 针对此问题, 本文主要在崔广章等[12]、龚坤等[13] 的基础上, 选择 CPU、内存、带宽、磁盘容量和 IO 速率的实时资源利用率作为评价指标, 并将基于层次分析法和熵权法得到的组合权重应用到 TOPSIS 多属性决策算法中, 致力于将 Pod 应用部署到最合适的节点上.

2 Kubernetes 默认调度策略

Kubernetes Scheduler 是 Kubernetes 集群的调度器, 用于将用户创建的 Pod 按照特定的调度算法和调度策略绑定到集群中某个合适节点上. 其调度过程可以分为两个阶段, 分别是预选和优选阶段.

预选阶段主要是根据用户提交 Pod 应用的最小资源需求过滤掉不满足需求的节点, Kubernetes 也提供了多种预选策略供用户选择, 如 PodFitsResources、PodsFitsPorts 等.

优选阶段主要是在预选阶段的基础上, 采集剩下节点上 CPU 和内存的空闲利用率进行评分, 选择评分最高的节点作为部署节点, 最后将 Pod 应用绑定到该节点上. 同样的, Kubernetes 也提供了几种优选策略:

1) LeastRequestedPriority, 该策略用于从候选节点中选出资源消耗最小的节点, 即 CPU 和内存空闲资源越多, 评分越高, 其计算公式如下:

$$score = \left[\frac{(Scpu - Ncpu)}{Scpu} \times 10 + \frac{(Smem - Nmem)}{Smem} \times 10\right] \Big/ 2 \tag{1}$$

2) BalancedResourceAllocation, 该策略用于从候选

节点中选出 CPU 和内存使用率最均衡的节点, 即 CPU 和内存使用率越接近, 评分越高, 其计算公式如下:

$$score = 10 - abs\left(\frac{Ncpu}{Scpu} - \frac{Nmem}{Smem}\right) \times 10 \quad (2)$$

上述两种策略, 均只考虑了 CPU 和内存, 且策略中 CPU 和内存利用率是由 Pod 应用的需求来衡量调度的优先级的, 无法准确反映节点实际资源使用情况, 也会影响节点的整体资源均衡性.

3) SelectorSpreading, 该策略将相同标签选择器选取的 Pod 应用尽可能散开部署到多个节点上, 节点上该标签选择器匹配的 Pod 应用数目越少, 则该节点的评分越高, 使用标签选择器的资源对象有: Service, Replication Controller, ReplicaSet 等.

此外, 调度策略还包括 NodePerferAvoidPods, InterPodAffinity, NodeAffinity, TaintToleration 等.

上述公式中, *Scpu* 和 *Smem* 分别表示节点上总的 CPU 和内存容量, 而 *Ncpu* 和 *Nmem* 分别表示节点上已被使用的 CPU 和内存的容量加上将要部署的 Pod 应用的 CPU 和内存的容量之和.

3 组合权重的 TOPSIS 调度策略

由于 Kubernetes 默认调度算法仅仅考虑 CPU 和内存, 没有考虑到带宽、IO 速率、磁盘容量等资源的需求, 无法对带宽敏感型、IO 速率敏感型等 Pod 应用进行合理的资源调度, 同时两个评分指标都采用相同权重, 无法满足 Pod 应用各异的资源需求, 当 Pod 应用部署的数量逐步增加时, 可能会造成其他指标如带宽、IO 速率等资源过度浪费.

在评价指标方面, 考虑到评价指标应该有效且有代表性, 因此本文选择 CPU、内存、带宽、磁盘容量、IO 速率作为评价指标.

在权重方面, 本文利用熵权法[14] 对层次分析法[15] 得到的主观权重进一步优化, 有效避免了层次分析法权重的主观性和熵权法权重的客观性.

最后将组合权重应用到 TOPSIS[16] 多属性决策方法中, 来计算 Pod 应用调度方案解和理想最优解及最劣解之间的距离, 通过理想贴合度的大小排序, 为 Pod 应用选择最合适的节点进行部署.

3.1 资源信息获取

本文主要采集两种资源信息:

(1) 节点当前的各个资源指标的资源利用率.

(2) 已部署的 Pod 应用在节点上占用的资源份额.

通过在集群中每个节点上部署 Proxy 监控代理, 用于采集上述两种资源信息, 具体流程如图 1 所示. 首先准备两个数据库, 分别用于存储监控节点的资源利用率和 Pod 的 CPU、内存、带宽、磁盘、IO 速率占用率信息. 在控制器模块逐个分析节点 Node 标识和 Pod 应用标识, 控制器可获取到 Proxy 代理的 IP、端口, 向 Proxy 代理发送监控命令, 之后采集实验需要的监控信息存放到对应的数据库中.

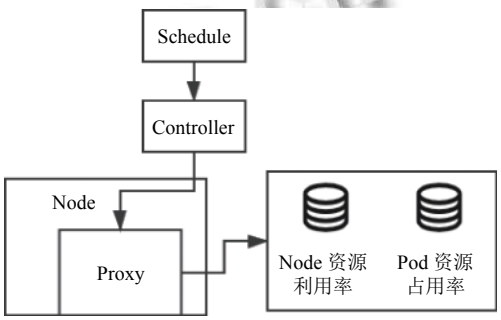


图 1 采集资源利用率

3.2 层次分析法求权重

层次分析法是一种使人们的思维过程和主观判断实现规划化的方法, 可以使因素的不确定性得到很大程度降低, 不仅简化了系统分析与计算工作, 而且有助于决策者保持其思维过程和决策过程原则的一致性, 是一种确定权重的科学方法, 其计算步骤如下:

Step 1. 构造资源指标判断矩阵

假设集群中有 m 个资源指标, 通过两两比较确定指标重要程度, 进而得出判断矩阵 $A = (a_{ij})_{m\times m}$, i=1, 2, …, m; j=1, 2, …, m, 重要程度的定义如表 1 所示.

表 1 相对重要程度表

标度	含义
1	两个元素相比, 具有相同重要性
3	两个元素相比, 前者比后者稍微重要
5	两个元素相比, 前者比后者明显重要
7	两个元素相比, 前者比后者强烈重要
9	两个元素相比, 前者比后者极端重要
倒数	若i对j的重要性之比为a_{ij}, 则j对i为a_{ji}=1/a_{ij}
2, 4, 6, 8	上述相邻判断的中间值

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mm} \end{pmatrix} \quad (3)$$

矩阵中, m 表示资源指标的个数, a_{ij}表示相对于第 j 个资源指标, 第 i 个资源指标的重要程度, 且$a_{ii}=1$.

本文以 5 种资源指标 CPU, 内存, 带宽, 磁盘, IO 速率为基础, 给出判断矩阵如下, 选取理由如表 2 所示.

表 2 部分指标对比结果

指标对比	标度值	选取理由
CPU→内存	a_{12}=3	CPU重要性高于内存
内存→带宽	a_{23}=3	内存重要性高于带宽
带宽→磁盘	a_{34}=3	带宽重要性高于磁盘
磁盘→IO速率	a_{45}=1/3	磁盘重要性不如IO速率

$$A=\begin{pmatrix} 1 & 3 & 5 & 7 & 5 \\ 1/3 & 1 & 3 & 5 & 3 \\ 1/5 & 1/3 & 1 & 3 & 1 \\ 1/7 & 1/5 & 1/3 & 1 & 1/3 \\ 1/5 & 1/3 & 1 & 3 & 1 \end{pmatrix} \tag{4}$$

Step 2. 一致性检验

判断矩阵客观反映了不同资源指标之间的相对重要性, 但是并不能保证每个判断矩阵都是一致的. 例如, 指标 A_1 比指标 A_2 重要, 指标 A_2 比指标 A_3 重要, 那么指标 A_3 比指标 A_1 重要就不合理, 因此需要通过计算一致性检验 CI (consistency index) 和一致性比例 CR (consistency ratio) 来衡量判断矩阵是否完全一致, 具体计算公式如下:

$$CI=\frac{\lambda_{\max}-m}{m-1} \tag{5}$$

$$CR=\frac{CI}{RI} \tag{6}$$

其中, $\lambda_{\max}$为判断矩阵的最大特征根, m 为矩阵的阶数, RI 为平均一致性指标, 其取值如表 3 所示. 若$CR<0.1$, 则通过一致性检验, 否则判断矩阵需要修正.

表 3 平均随机一致性指标

m	1	2	3	4	5	6	7
RI	0	0	0.52	0.89	1.12	1.24	1.36

Step 3. 计算资源指标权重

本文选择常用特征值法来计算权重, 只需要计算符合$A\cdot W=\lambda_{\max}\cdot W$的特征向量, 然后$W\lambda_{\max}$将归一化, 即可得到权重$w_{Aj}$, 具体权重信息如表 4 所示.

表 4 AHP 资源指标权重

指标	CPU	内存	带宽	磁盘	IO速率
权重	0.504	0.245	0.102	0.045	0.102

3.3 熵权法求权重

熵权法基本思路是根据指标变异性的大小来确定客观权重. 该方法可以深刻反映出指标的区分能力, 若某个指标的信息熵越小, 表明指标值的变异程度越大, 提供的信息量越多, 在综合评价中所能起到的作用也越大, 其权重也就越大. 其计算步骤如下.

Step 1. 数据预处理

假设集群中有 n 个节点, m 个资源指标, 通过监控代理 Proxy 采集到的资源利用率的实时数据, 构造矩阵$X=\left(x_{ij}\right)_{n\times m}$, i=1, 2, ⋯, n; j=1, 2, ⋯, m, x_{ij}表示第 i 个节点上第 j 个资源指标的资源利用率.

$$X=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \tag{7}$$

Step 2. 资源指标归一化处理

考虑到每个资源指标的计量单位不同, 有必要在计算综合资源指标前, 进行标准化处理, 即把资源指标的绝对值转化为相对值.

正向型资源指标归一化, 用于处理效益型决策参数, 这类决策参数的特点是值越大效果越好, 其计算公式如下:

$$x'_{ij}=\frac{x_{ij}-\max\left(x_j\right)}{\max\left(x_j\right)-\min\left(x_j\right)} \tag{8}$$

负向型资源指标归一化, 用于处理成本型决策参数, 这类决策参数的特点是值越小效果越好, 其计算公式如下:

$$x'_{ij}=\frac{\max\left(x_j\right)-x_{ij}}{\max\left(x_j\right)-\min\left(x_j\right)} \tag{9}$$

其中, $\max\left(x_j\right)$和$\min\left(x_j\right)$分别表示第 j 个资源指标的最大值和最小值, i=1, 2, ⋯, n; j=1, 2, ⋯, m.

Step 3. 计算第 j 个资源指标下第 i 个节点的值所占该资源指标的比重P_{ij}, i=1, 2, ⋯, n; j=1, 2, ⋯, m.

$$P_{ij}=\frac{x'_{ij}}{\sum_{i=1}^{n}x'_{ij}} \tag{10}$$

Step 4. 计算第 j 个资源指标的熵值e_j, j=1, 2, ⋯, m.

$$e_j=-k\sum_{i=1}^{n}P_{ij}In\left(P_{ij}\right) \tag{11}$$

其中, $k=1/In(n)>0$, 满足$e_j \geq 0$.

Step 5. 计算资源指标的差异系数d_j, j=1, 2, ⋯, m.

$$d_j = 1 - e_j \tag{12}$$

Step 6. 计算各指标的权重w_{Ej}, j=1, 2, ⋯, m.

$$w_{Ej} = \frac{d_j}{\sum_{j=1}^{m} d_j} \tag{13}$$

3.4 计算组合权重

将熵权法得到的权重与层次分析法得到的权重结合, 用于避免层次分析法的主观影响和熵权法的客观影响, 其计算公式如下:

$$w_j = \frac{\sqrt{w_{Aj} \cdot w_{Ej}}}{\sum_{j=1}^{m} \sqrt{w_{Aj} \cdot w_{Ej}}} \tag{14}$$

其中, w_j为第 j 个资源指标的组合权重, w_{Aj}和 w_{Ej}分别表示通过层次分析法和熵权法得到的第 j 个资源指标的权重.

以表 4 得到的 AHP 权重为基础, 监控代理 Proxy 采集到的某个时间段的 5 种资源指标的实时利用率如表 5 所示, 根据表 5 的数据计算出熵权权重, 最后根据式 (14) 计算得到组合权重, 具体权重信息如表 6 所示.

表 5 资源实时利用率 (%)

节点编号	CPU	内存	带宽	磁盘	IO速率
1	33	25	28	37	44
2	41	35	28	47	29
3	33	71	23	66	50
4	40	30	15	40	20
5	46	56	36	56	39
6	33	33	33	33	22

表 6 详细权重信息

方案	CPU	内存	带宽	磁盘	IO速率
AHP	0.504	0.245	0.102	0.045	0.102
EW	0.182	0.168	0.247	0.177	0.224
组合权重	0.485	0.217	0.133	0.042	0.121

3.5 TOPSIS 确定最优节点

逼近理想解排序方法 (TOPSIS) 是一种有效的多属性决策方案, 通过从归一化的数据矩阵中构造出决策问题的正负理想解, 计算出方案与正负理想解的距离, 最终得到贴合度作为评价方案的优劣依据. 其计算步骤如下.

Step 1. 构造决策矩阵

假设集群中有 n 个节点, m 个资源指标, 通过监控代理 Proxy 采集到的资源利用率的实时数据加上 Pod 部署所占用的资源利用率, 构造决策矩阵$X=\left(x_{ij}\right)_{n\times m}$, i=1, 2, ⋯, n; j=1, 2,⋯, m.

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \tag{15}$$

其中, x_{ij}表示第 i 个节点上第 j 个资源指标的资源利用率.

Step 2. 归一化决策矩阵

采用极差标准化方法对决策矩阵进行归一化处理, 目的是消除决策参数不同量纲之间的影响.

$$x'_{ij} = \frac{\max\left(x_j\right) - x_{ij}}{\max\left(x_j\right) - \min\left(x_j\right)} \tag{16}$$

其中, $\max\left(x_j\right)$和$\min\left(x_j\right)$分别表示第 j 个资源指标的最大值和最小值, i=1, 2, ⋯, n; j=1, 2, ⋯, m.

Step 3. 构造加权决策矩阵

各异的 Pod 应用对节点资源指标的需求敏感程度不同. 大体上可以分为内存倾向型、CPU 倾向型、IO 速率倾向型, 带宽倾向型等等, 若是每个资源指标都采用相同权重, 无法满足各异需求的 Pod 应用, 因此有必要分别为节点的资源指标设置不同的权重.

基于此, 在考虑资源指标的差异性, 和避免了层次分析法人为的主观影响和熵权法带来的客观影响下, 本文通过结合层次分析法与熵权法得到组合权重, 并应用到决策矩阵中, 构造了加权决策矩阵$Z=\left(z_{ij}\right)_{n\times m}$, i=1, 2, ⋯, n; j=1, 2, ⋯, m.

$$z_{ij} = \frac{x'_{ij}}{\sqrt{\sum_{i=1}^{n} x'^{2}_{ij}}} \cdot w_j \tag{17}$$

其中, w_j是第 j 个资源指标的组合权重.

Step 4. 计算正负理想解

$$Z^{+} = (Z_1^{+}, Z_2^{+}, \cdots, Z_m^{+}) = (\max\{z_{1m}, z_{2m}, \cdots, z_{nm}\}) \tag{18}$$

$$Z^{-} = (Z_1^{-}, Z_2^{-}, \cdots, Z_m^{-}) = (\min\{z_{1m}, z_{2m}, \cdots, z_{nm}\}) \tag{19}$$

其中, Z^{+}表示加权决策矩阵的正理想解, 由所有候选节点上每种资源指标参数的最大值构成; Z^{-}表示加权决策矩阵的负理想解, 由所有候选节点上每种资源指标

参数的最小值构成.

Step 5. 计算每个候选节点到正、负理想解的距离

$$D^{+}=\sqrt{\sum_{j=1}^{m}\left(Z_j^{+}-z_{ij}\right)^2} \tag{20}$$

$$D^{-}=\sqrt{\sum_{j=1}^{m}\left(Z_j^{-}-z_{ij}\right)^2} \tag{21}$$

其中, D^{+}和D^{-}分别代表各候选节点到正负理想解的欧式距离.

Step 6. 计算每个候选节点与最优候选节点的相对贴合度S_i

$$S_i=\frac{D_i^{-}}{D_i^{+}+D_i^{-}} \tag{22}$$

其中, 相对贴合度S_i越大, 说明该候选节点越适合当前需要部署的 Pod 应用.

以表 5 和表 6 的数据为基础, 结合 TOPSIS 多属性决策算法可以得到相对贴合度 S, 如表 7 所示, 可以看出节点 1 的相对贴合度 S 最大, 因此将 Pod 应用部署到节点 1 最合适.

表 7 相对贴合度

节点编号	1	2	3	4	5	6
相对贴合度	0.77	0.46	0.64	0.59	0.10	0.76

4 实验验证与分析

为测试本文提出的 Kubernetes 调度算法的性能, 进行实验验证. 所有实验均由 PyCharm 2020.2 编程实现, 并基于平台: Windows 10, Intel(R) Core(TM) i7-8565U CPU 1.80 GHz, 16 GB 内存.

4.1 实验环境

在仿真环境下, 模拟一个包含 60 个节点的 Kubernetes 集群, 集群中节点分为 3 种类型, 每种类型 20 个来模拟异构的环境, 节点具体信息如表 8 所示.

表 8 节点资源信息

CPU (core)	内存 (MB)	带宽 (Gb/s)	存储 (GB)	IO速率 (MB/s)
16	24576	30	800	180
32	32768	30	1000	200
64	49152	40	1200	220

同样的, 鉴于容器应用资源多样化的需求, 本文按照 CPU 敏感型、内存敏感型、带宽敏感型、存储敏感型、IO 速率敏感型以及无倾向类型 6 类容器应用构造了 Pod 资源需求, 表 9 为部分 Pod 资源需求.

表 9 部分 Pod 资源需求表

CPU (core)	内存 (MB)	带宽 (Gb/s)	存储 (GB)	IO速率 (MB/s)
0.1	150	0.05	14.5	1.8
0.3	680	0.4	1.2	0.9
0.8	210	0.5	4.2	1.2
0.2	290	0.02	2.3	4.6

4.2 实验评价指标

假设 Kubernetes 集群中有 n 个节点, 每个节点上有 m 种资源. 资源的实时利用率是通过监控代理 Proxy 采集获取得到的. $U(i,j)$表示节点 i 上资源 j 的资源利用率; $U_{\text{avg}}(i)$表示节点 i 上各个资源利用率总和的平均值; $S(i)$表示节点 i 上各个资源利用率的标准差, 即资源失衡度; S_{avg}表示所有节点的平均资源失衡度. S_{avg}的值越小, 代表集群中各个资源的利用率越平衡, 就不容易出现资源倾斜, 如此就可以部署更多的 Pod 应用, 具体计算公式如下:

$$U_{\text{avg}}=\sum_{j=1}^{m}\frac{U(i,j)}{m} \tag{23}$$

$$S(i)=\sqrt{\sum_{j=1}^{m}\left(U(i,j)-U_{\text{avg}}(i)\right)^2} \tag{24}$$

$$S_{\text{avg}}=\sum_{i=1}^{n}\frac{S(i)}{n} \tag{25}$$

4.3 实验结果和分析

在上述 Kubernetes 集群中, 分别采用 Kubernetes 的 LeastRequestedPriority (LRP) 策略与 BalancedResourceAllocation (BRA) 策略和本文提出的组合权重 TOPSIS 调度算法 (CWT), 从资源平衡度、CPU、内存、带宽、IO 速率角度来对比 3 种算法的表现.

1) 集群资源失衡度

集群失衡度变化曲线如图 2 所示, 当 Pod 数量小于 1 200 时, 此时集群负荷相对较低, LRP 和 BRA 策略与 CWT 算法的资源失衡度相差不大, 但随着部署的 Pod 应用数量越来越多, 可以很明显的发现 CWT 算法开始发挥作用, 它的资源失衡度明显好于 LRA 策略.

在集群资源整体快达到饱和时, CWT 算法优于 BRA 策略. 这是由于 CWT 算法不仅考虑了集群节点的 5 种资源指标, 还分别考虑了各个资源指标的权重

情况, 这就降低了集群中单个节点出现单个资源用尽而其他资源大量剩余的可能. 尤其是在集群资源整体饱和的情况下, CWT 算法的资源失衡度比 LRA 策略整体下降 18%, 比 BRA 策略整体下降 7.7%, 这说明 CWT 算法在集群资源饱和的情况下可有效的调节集群资源平衡度.

2) CPU、内存资源利用率

图 3 和图 4 反映了在 Pod 应用数量为 7 200 时, CWT 算法、LRP 策略和 BRA 策略下各个节点的 CPU 和内存资源利用率的情况. 在 BRA 策略下, CPU 资源利用率和内存资源利用率好于 CWT 算法, 且远远好于 LRP 策略, 这是由于 BRA 策略会选取 CPU 和内存资源使用率最接近的节点进行部署. 而在 LRP 策略下, 有 3 个节点 CPU 资源已经饱和, 1 个节点的内存资源已经饱和, 饱和率达到 6.7%, 这意味着这 4 个节点上所有 Pod 应用的处在资源受到限制的环境, 严重影响了集群的整体健康. 在 CWT 算法下, CPU 和内存的资源利用率波动比 BRA 策略稍微大一些, 这是由于 CWT 算法综合考虑了 5 种资源指标, 相较于 BRA 策略只考虑 CPU 和内存两种资源指标, CWT 算法则考虑的更加全面, 更加能适应实际生产环境的需求.

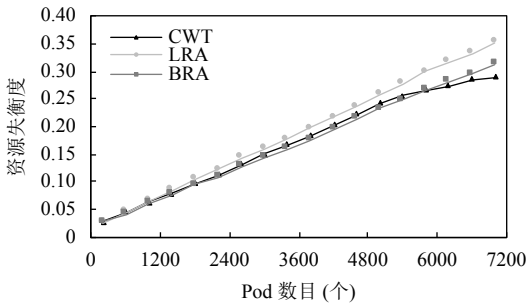
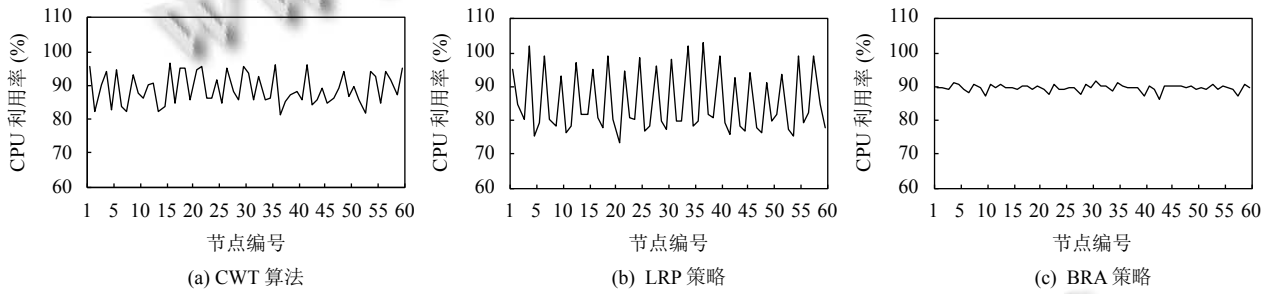
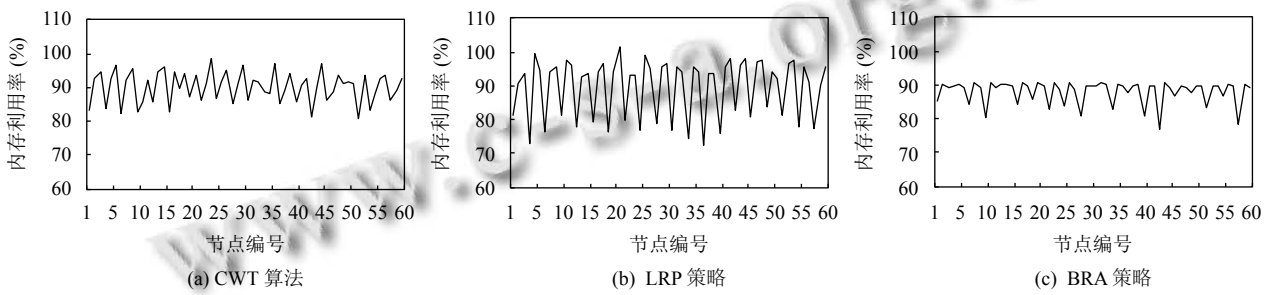


图 2 集群资源失衡度变化



(a) CWT 算法 (b) LRP 策略 (c) BRA 策略

图 3 不同策略下的 CPU 利用率



(a) CWT 算法 (b) LRP 策略 (c) BRA 策略

图 4 不同策略下的内存利用率

3) 带宽、IO 速率资源利用率

图 5 和图 6 反映了在 Pod 应用数量为 7 200 时, CWT 算法、LRP 策略和 BRA 策略下各个节点的带宽和 IO 速率资源利用率的情况. 由于 LRP 策略与 BRA 策略只考虑 CPU 和内存, 未考虑带宽和 IO 速率等因素, 随着 Pod 应用数量增加, 在这两种策略下资源失衡度上升, 集群中部分节点出现资源倾斜, 尤其是带宽和 IO 资源利用率, 很明显可以看出集群中节点之间利用率波动较大.

在 LRP 策略下, 带宽利用率最大的节点与利用率最小的节点相差 48%, IO 资源利用率最大的节点与利用率最小的节点相差 24%, 而在 BRA 策略带宽利用率最大的节点与利用率最小的节点相差 47%, IO 资源利用率最大的节点与利用率最小的节点相差 41%, 甚至

有多个节点的带宽和 IO 资源利用率已超过 100%, 这说明这些节点的带宽和 IO 资源已经完全饱和, LRP 策略下饱和率达到 25%, BRA 策略下饱和度达到 18.3%. 若是带宽和 IO 速率敏感型的 Pod 应用被部署在这些带宽和 IO 资源饱和的节点上, 会带来网络和 IO 读写的拥堵.

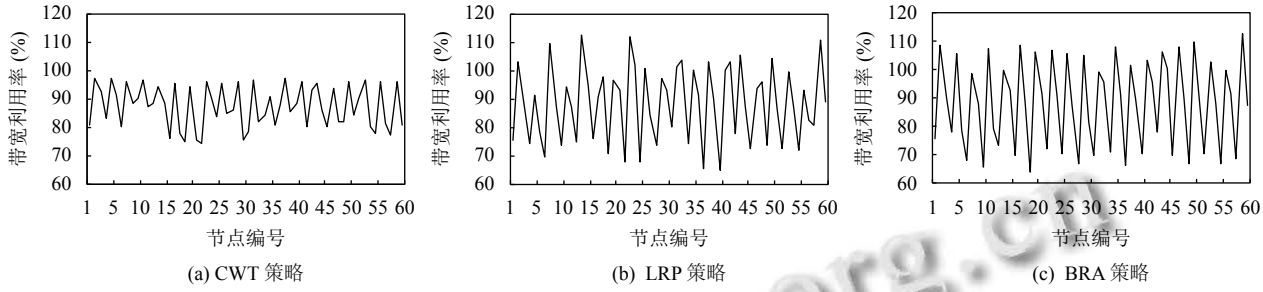


图 5 不同策略下的带宽利用率

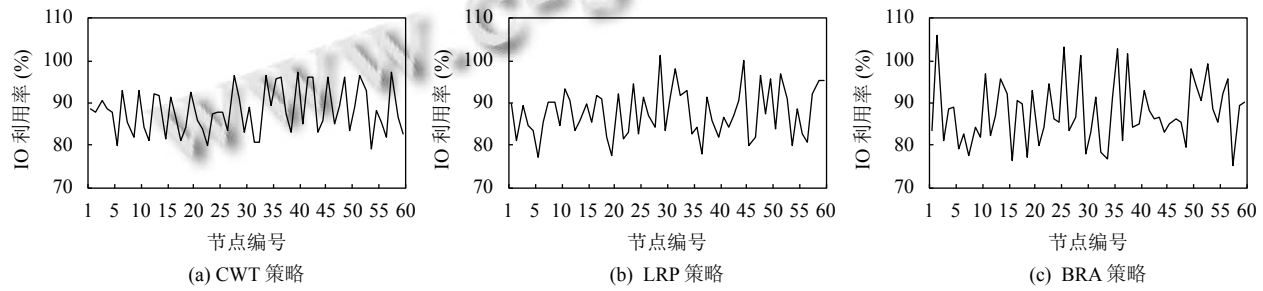


图 6 不同策略下的 IO 利用率

在 CWT 算法下, 全部节点的带宽和 IO 资源利用率都在 75% 到 95% 的区间内震荡, 没有出现带宽资源利用率超过 100% 的情况, 相较于 LRP 策略和 BRA 策略, CWT 算法更加保障了集群的带宽稳定.

5 结论与展望

本文针对 Kubernetes 默认调度算法仅考虑 CPU、内存两种资源指标, 且对需求各异的 Pod 应用采用相同权重的调度策略进行了改进, 通过增加带宽、磁盘、IO 速率 3 项指标, 结合层次分析法与熵权法得到每个资源指标的组合权重, 并应用到 TOPSIS 多属性决策方法中, 为 Pod 应用选择合适的节点进行部署, 有效提高了集群整体的资源平衡度, 避免了集群中节点上单个资源耗尽而其他资源尚有剩余的情况, 通过实验验证, 证明了组合权重 TOPSIS 调度算法的有效性和合理性. 下一阶段将考虑集群的动态调度与多租户情况相结合, 使得集群资源调度更加高效, 集群资源更加平衡, 集群资源利用率更高.

参考文献

1 龚正, 吴治辉, 崔秀龙, 等. Kubernetes 权威指南: 从 Docker 到 Kubernetes 实践全接触. 4 版. 北京: 电子工业出版社, 2019.

2 Burns B, Grant B, Oppenheimer D, *et al*. Borg, omega, and Kubernetes: Lessons learned from three container-management systems over a decade. Queue, 2016, 14(1): 70–93. [doi: 10.1145/2898442.2898444]

3 Menouer T, Darmon P. New scheduling strategy based on multi-criteria decision algorithm. 2019 27th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). Pavia: IEEE, 2019. 101–107. [doi: 10.1109/EMPDP.2019.8671594]

4 El Haj Ahmed G, Gil-Castiñeira F, Costa-Montenegro E. KubCG: A dynamic Kubernetes scheduler for heterogeneous clusters. Software: Practice and Experience, 2021, 51(2): 213–234. [doi: 10.1002/SPE.2898]

5 Imdoukh M, Ahmad I, Alfailakawi M. Optimizing scheduling decisions of container management tool using many-objective genetic algorithm. Concurrency and

Computation: Practice and Experience, 2020, 32(5): e5536. [doi: 10.1002/cpe.5536]

6 Zhang WG, Ma XL, Zhang JZ. Research on Kubernetes' resource scheduling scheme. Proceedings of the 8th International Conference on Communication and Network Security. Qingdao: ACM, 2018. 144–148. [doi: 10.1145/3290480.3290507]

7 孔德瑾, 姚晓玲. 面向 5G 边缘计算的 Kubernetes 资源调度策略. 计算机工程, 2021, 47(2): 32–38. [doi: 10.19678/j.issn.1000-3428.0058047]

8 Li D, Wei Y, Zeng B. A dynamic I/O sensing scheduling scheme in kubernetes. Proceedings of the 2020 4th International Conference on High Performance Compilation, Computing and Communications. Guangzhou: ACM, 2020. 14–19. [doi: 10.1145/3407947.3407950]

9 吴双艳. 基于 Docker 容器调度优化方法的研究 [硕士学位论文]. 郑州: 郑州大学, 2019.

10 Dua A, Randive S, Agarwal A, *et al*. Efficient load balancing to serve heterogeneous requests in clustered systems using kubernetes. 2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC). Las Vegas: IEEE, 2020. 1–2. [doi: 10.1109/CCNC46108.2020.9045136]

11 Zheng GS, Fu Y, Wu TT. Research on docker cluster scheduling based on self-define Kubernetes scheduler. Journal of Physics: Conference Series, 2021, 1848: 012008. [doi: 10.1088/1742-6596/1848/1/012008]

12 崔广章, 朱志祥. 容器云资源调度策略的改进. 计算机与数字工程, 2017, 45(10): 1931–1936. [doi: 10.3969/j.issn.1672-9722.2017.10.009]

13 龚坤, 武永卫, 陈康. 容器云多维资源利用率均衡调度研究. 计算机应用研究, 2020, 37(4): 1102–1106. [doi: 10.19734/j.issn.1001-3695.2018.09.0764]

14 王宗杰, 郭举. 基于熵权层次分析法的云平台负载预测. 计算机工程与设计, 2021, 42(1): 263–269. [doi: 10.16208/j.issn1000-7024.2021.01.038]

15 叶珍. 基于 AHP 的模糊综合评价方法研究及应用 [硕士学位论文]. 广州: 华南理工大学, 2010.

16 彭定洪, 黄子航, 王铁旦, 等. 面向云计算部署方案评价的区间犹豫模糊双重妥协评价方法. 计算机集成制造系统, 2021, 27(6): 1768–1779. [doi: 10.13196/j.cims.2021.06.022]