计算机科学
COMPUTER SCIENCE

基于GMM的容器定制化调度策略

周凯, 王凯, 朱宇航, 普黎明, 刘树新, 周德强

引用本文

周凯, 王凯, 朱宇航, 普黎明, 刘树新, 周德强. 基于GMM的容器定制化调度策略[J]. 计算机科学, 2025, 52(6): 346-354.

ZHOU Kai, WANG Kai, ZHU Yuhang, PU Liming, LIU Shuxin, ZHOU Deqiang. Customized Container Scheduling Strategy Based on GMM[J]. Computer Science, 2025, 52(6): 346-354.

相似文章推荐（请使用火狐或 IE 浏览器查看文章）

Similar articles recommended (Please use Firefox or IE to view the article)

多云多副本的远程动态数据完整性检验方案

Remote Dynamic Data Integrity Checking Scheme for Multi-cloud and Multi-replica

计算机科学, 2025, 52(5): 345-356. https://doi.org/10.11896/jsjkx.240300027

基于局部加密网格的LBM程序负载均衡方法研究

Investigation on Load Balancing Strategies for Lattice Boltzmann Method with Local Grid Refinement

计算机科学, 2025, 52(5): 101-108. https://doi.org/10.11896/jsjkx.241100169

基于图强化学习的多边缘协同负载均衡方法

Graph Reinforcement Learning Based Multi-edge Cooperative Load Balancing Method

计算机科学, 2025, 52(3): 338-348. https://doi.org/10.11896/jsjkx.240100091

面向云数据中心基于改进A2C算法的任务调度策略

Task Scheduling Strategy Based on Improved A2C Algorithm for Cloud Data Center

计算机科学, 2025, 52(2): 310-322. https://doi.org/10.11896/jsjkx.240500111

SDN网络中基于分布式Sketch的流基数估计方法

Flow Cardinality Estimation Method Based on Distributed Sketch in SDN

计算机科学, 2025, 52(2): 268-278. https://doi.org/10.11896/jsjkx.240900059

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240900154

基于 GMM 的容器定制化调度策略

周 凯 王 凯 朱宇航 普黎明 刘树新 周德强
信息工程大学国家数字交换系统工程技术研究中心 郑州 450003
(zhoukai1247@126.com)

摘 要 在云计算环境中,随着容器数量和类型的不断增加,资源管理和调度复杂性也增加,如何有效地调度容器,优化资源利用率和集群性能成为一个重要的研究课题。现有的容器集群调度策略没有充分考虑容器的多样化需求,缺乏灵活性,难以实现针对不同场景的容器进行定制化调度,容易出现集群资源利用率低下、集群资源负载失衡等问题。为了满足容器多样化需求,提高集群资源负载均衡性,提出了一种基于 GMM(高斯混合模型)的容器定制化调度策略(Customized Container Scheduling Strategy Based on GMM,CS-GMM)。首先,根据容器的资源和属性需求进行分类,将其划分为不同的类型。其次,对于每一类容器,分别计算并分配不同的独立权重,依次将容器根据其类型调度到合适的节点,从而实现定制化调度。通过这种方式,满足了容器多样化需求,使不同类型的容器可以根据其特定需求得到最优的资源配置,避免了资源竞争和冲突,从而提高了集群资源的整体利用率和负载均衡度。实验结果表明,与 Kubernetes Scheduler 相比,该调度策略在多种容器调度场景下均表现出优越的性能,集群节点之间最大资源利用率差值降低 17.1%,容器调度成功率提升 19%,集群节点负载均衡度提升 57.51%。

关键词:云计算;容器调度;多样化;定制化;负载均衡
中图分类号 TP391

Customized Container Scheduling Strategy Based on GMM

ZHOU Kai,WANG Kai,ZHU Yuhang,PU Liming,LIU Shuxin and ZHOU Deqiang
National Digital Switching System Engineering & Technological R&D Center,Information Engineering University,Zhengzhou 450003,China

Abstract In cloud computing environments,as the number and types of containers continue to increase,resource management and scheduling complexity are increased. How to effectively schedule containers and optimize resource utilization and cluster performance has become an important research topic. The existing container cluster scheduling strategies do not fully consider the diverse needs of containers,lack flexibility,and are difficult to customize scheduling for containers in different scenarios. This can easily lead to problems such as low cluster resource utilization and imbalanced cluster resource load. In order to meet the diverse needs of containers and improve the load balancing of cluster resources,this paper proposes a customized container scheduling strategy based on GMM(Gaussian Mixture Model). Firstly,classify according to the resources and attribute requirements of the container,and divide it into different types. Secondly,for each type of container,different independent weights are calculated and assigned separately,and the containers are scheduled to appropriate nodes according to their types in turn,thereby achieving customized scheduling. In this way,the diverse needs of containers are met,so that different types of containers can get the optimal resource allocation according to their specific needs,avoiding resource competition and conflicts,thereby improving the overall utilization and load balancing of cluster resources. Experimental results show that compared with Kubernetes Scheduler,this scheduling strategy has shown superior performance in various container scheduling scenarios,with the maximum resource utilization difference between cluster nodes reduced by 17.1%,the container scheduling success rate increased by 19%,and the cluster node load balancing increased by 57.51%.

Keywords Cloud computing,Container scheduling,Diversification,Customization,Load balancing

1 引言

随着云计算的快速发展和广泛应用,容器技术已成为现代应用程序部署和管理的重要工具[1]。容器技术支持众多来自垂直行业的多样化业务[2],不同业务下的容器对资源和属性的要求各异。例如:在边缘调度场景下,边缘计算[3]类容器更加关注计算资源和低时延响应;边缘存储[4]类容器更加关注存储资源及其可靠性等。显然,容器业务的多样性直接导

到稿日期:2024-09-25 返修日期:2024-12-06
基金项目:河南省重大科技专项(221100210700-2)
This work was supported by the Major Science and Technology Program in Henan Province(221100210700-2).
通信作者:王凯(wangkai0508@126.com)

致了容器需求的多样化。而当前关于容器调度策略的研究主要侧重于基于集群节点资源状况进行调度[5-8],忽视了容器的具体需求,导致无法充分满足容器多样化需求,同时也影响了集群资源利用效率,导致集群资源负载失衡。因此,传统的容器调度策略亟需更新和适应,以有效应对这些多样化的需求和资源负载均衡的问题。

针对容器多样化需求和集群资源负载均衡的问题,本文提出了一种基于 GMM[9] 的容器定制化调度策略 CS-GMM。CS-GMM 整合了 GMM 的数据聚类和分析能力,并对其进行扩展(包括调度逻辑和决策流程),将原本用于统计模型领域的 GMM 应用于容器调度领域,综合考虑了容器的多样化资源和属性需求以及集群资源负载的均衡性,选择既满足容器需求又能均衡集群负载的节点。根据不同业务场景下的容器需求,对资源和属性的组合进行个性化的调度策略定制。这种动态适应特性确保了满足容器多样化需求,同时显著提升了集群的资源利用率和负载均衡性,从而维持优异的调度性能。

本文的贡献主要集中在以下几个方面:

1)系统地分析了当前研究在处理容器多样化资源和属性需求,以及在实现资源负载均衡方面的局限性。

2)构建了描述集群资源负载状态和调度问题的相关模型,提出了基于 GMM 的容器定制化调度策略 CS-GMM,旨在有效解决多样化容器调度问题。

3)构建了基于 CS-GMM 的容器定制化调度框架并将其与主流容器编排系统 Kubernetes 相结合。实验结果表明,这样的结合可以有效满足容器多样化需求,提升集群的资源利用率和负载均衡性。

2　相关工作

研究人员提出了多种容器集群调度策略,这些策略的核心目标主要涵盖两个方面:1)资源利用率的优化;2)非资源(即属性)的优化。

在资源利用率优化方面,文献[5]提出了一种基于多目标元启发式的算法来调度云环境中的作业,以解决资源可用性和波动的工作负载问题。文献[6]通过放置策略来提高资源利用率并降低能耗,提出了一种基于生物启发式算法的集装箱优化摆放策略。文献[7]采用多目标优化并行粒子群算法进行基于容器的微服务调度,对网络传输等资源负载均衡进行优化。文献[8]提出了一种基于 Kubernetes 的粒子群优化的大数据容器调度算法,以解决资源利用率低、集群弹性伸缩扩展时间较长的问题。文献[10]针对单个任务不能充分利用整张 GPU 的计算资源的问题,提出了一种 GPU 共享调度系统。文献[11]提出了一种基于 Docker Swarm 和微服务架构的大数据应用容器调度机制,以有效地管理在容器化环境中与大数据应用程序相关的不断增加的工作负载。文献[12]通过改进用户数字化请求的集装箱时间表来提高调度效率,并提出使用最佳容器管理策略来联合优化云中的负载失衡和资源问题。文献[13]提出了一种结合粒子群优化算法和灰狼算法的方法,利用快速收敛和全局优化的优点,提高了系统效率和资源分配,解决了负载失衡的问题。文献[14]提出了一个优先级感知的工作负载调度算法,以优化云资源的利用效率。文献[15]针对人工智能任务的容器集群,提出了一种容器云资源调度算法,减少了任务等待时间,均衡了集群资源负载。文献[16]提出针对短期深度学习应用程序进行优化的新型容器调度器,以更好地处理深度学习工作负载。文献[17]为了均衡容器集群的工作负载,减少容器资源瓶颈的出现,提出了一种基于 DQN 的容器集群调度优化算法。尽管上述研究工作考虑了多种资源的多目标优化问题,但其主要优化视角仍集中在从集群节点资源的角度提高资源利用率,即倾向于选择资源最丰富的节点进行容器调度,而没有充分考虑容器自身对资源的具体需求,选择匹配容器特性且使集群负载均衡的节点。例如,研究倾向于将容器调度到所有资源均较为空闲的节点上。然而,对于 CPU 资源需求较高的容器,将其调度到 CPU 资源较为空闲而其他资源相对紧张的节点,更能有效利用集群中的碎片化资源,并促进集群资源负载的均衡。此外,这些研究在优化资源利用率的同时,并未兼顾到容器对非资源需求的综合考量。

在非资源优化方面,文献[18]建立了一个容器容错部署模型,使得边缘计算系统在容器部署失败或因硬件故障导致部署速度下降时,仍能提供服务,提高了工厂的容错能力。文献[19]针对云计算环境下的科学工作流任务调度问题,提出了一种多目标优化方法,其目的是最大限度地减少完工时间,降低执行任务的成本。文献[20]针对集群环境中的容器放置问题,提出了一个集成了异构集群、微服务、容器和 4 个优化目标的系统模型。文献[21]研究了云服务器之间的容器迁移策略,提出了一种用于管理容器的两阶段框架,以最大限度地减少迁移成本。文献[22]制定并解决了一个应用混合整数线性规划的多目标优化问题,主要目标是通过使用容器来最小化流程制定的成本。文献[23]提出了一种任务调度方式,使工作流高效完成,减少了成本和流程时间。文献[24]提出了一种云资源调度模型,模型所需执行时间短,且稳定性好。文献[25]提出了一种基于深度强化学习的作业调度算法来提高作业的执行效率和平均响应时间。文献[26]提出了一种针对长时间运行应用程序类容器的调度算法,该算法优化了放置和调度目标。文献[27]针对边缘计算提出了一种基于图层依赖关系的容器调度算法,以减少任务的总完成时间。上述大量研究表明,目前对容器的非资源优化的研究需求日益增加,若仅关注容器调度时对节点底层资源利用率的优化,而没有考虑容器对节点的可靠性、成本等属性的需求,已经不能满足现代企业对于容器的要求。现代企业不仅要求高效的资源利用率,还需要确保容器调度时考虑节点的可靠性、成本等其他属性,以适用更为复杂和多样化的实际应用场景。

综合上述大量研究表明,当前的研究工作尚未从容器需求的视角出发,探索一种既能满足容器需求,又能有效避免集群资源碎片产生,提高集群资源负载均衡的调度策略。此外,仅对容器调度进行资源或属性的单一优化,无法根据容器多样化需求进行资源和属性的定制化调度,这一局限使得对于不同业务场景下拥有不同功能,进而对资源和属性的需求各不相同的容器,难以获得对其精准的调度策略。表 1 列出了本文策略与主要相关工作在以下 4 个方面的对比结果:资源

利用率优化、属性优化、基于容器需求均衡负载的能力，以及满足容器多样化需求的能力。具体如表1所列。

表1 本文策略与相关工作的对比

Table 1 Comparison of the proposed strategy with related work

方法	资源优化	属性优化	负载均衡	多样化需求
文献[11]	√	×	√	×
文献[12]	√	√	√	×
文献[13]	√	×	√	×
文献[14]	√	×	×	×
文献[15]	√	×	√	×
文献[16]	√	×	√	×
文献[17]	√	×	√	×
文献[18]	√	√	×	×
文献[19]	×	√	×	×
文献[20]	×	√	×	×
文献[21]	×	√	×	×
文献[22]	√	√	×	×
文献[23]	×	√	×	×
文献[24]	×	√	×	×
本文策略	√	√	√	√

为解决上述研究工作中存在的问题，本文提出了基于GMM的容器定制化调度策略。该策略从容器需求的角度出发，避免将节点资源作为主要调度依据，从而减少资源碎片化，避免集群负载失衡。同时，在优化集群资源负载的过程中，综合考虑了容器在不同业务场景下对多样化资源和属性的需求，满足了容器的多样化需求。

3 模型建立

CS-GMM策略的目标是在有效分配和管理资源的同时，满足容器的多样化需求。下面给出集群资源负载状态和容器多样化需求调度问题的相关定义。

定义1(容器和节点的集合) 容器集合 $C=\{c_1,c_2,\cdots,c_{|C|}\}$ 表示所有需要调度的容器，c_i 表示第 i 个容器，共有 $|C|$ 个容器需要调度。节点集合 $N=\{n_1,n_2,\cdots,n_{|N|}\}$ 表示集群中所有可供容器调度的物理节点，n_i 表示第 i 个节点，共有 $|N|$ 个物理节点可用。

定义2(资源类型和属性类型集合) 资源类型集合 $R=\{r_1,r_2,\cdots,r_{|R|}\}$ 包含了容器需要的资源类型，如CPU资源、内存资源、存储资源、网络带宽等，r_i 表示第 i 种资源类型，共有 $|R|$ 种资源类型。属性类型集合 $A=\{a_1,a_2,\cdots,a_{|A|}\}$ 包含了容器需要的节点属性类型，如可靠性、成本等，a_i 表示第 i 种节点属性类型，共有 $|A|$ 种属性类型。

定义3(容器资源需求矩阵) $\boldsymbol{D}=[d_{ij}]\in\mathbb{R}^{|C|\times|R|}$，其中，$d_{ij}$ 表示容器 c_i 对资源 r_j 的需求量。容器资源需求矩阵用于描述每个容器对各种资源的具体需求。

定义4(节点资源容量矩阵) $\boldsymbol{S}=[s_{ij}]\in\mathbb{R}^{|N|\times|R|}$，其中，$s_{ij}$ 表示节点 n_i 拥有的资源 r_j 的总量。节点资源容量矩阵用于描述每个节点各类资源的总量。

定义5(节点资源负载矩阵) $\boldsymbol{L}=[l_{ij}]\in\mathbb{R}^{|N|\times|R|}$，其中，$l_{ij}$ 表示节点 n_i 上已经使用的资源 r_j。节点资源负载矩阵用于描述每个节点上各类资源的使用情况。

定义6(容器属性需求矩阵) $\boldsymbol{E}=[e_{ij}]\in\mathbb{R}^{|C|\times|A|}$，其中，$e_{ij}$ 表示容器 c_i 对节点属性 a_j 的需求等级。将容器对属性的需求等级进行量化，用于描述每个容器对各种属性的具体需求。

定义7(节点属性等级矩阵) $\boldsymbol{P}=[p_{ij}]\in\mathbb{R}^{|N|\times|A|}$，其中，$p_{ij}$ 表示节点 n_i 在属性 a_j 上的等级值。将节点可提供的属性等级进行量化，用于描述每个节点上各种属性的具体情况。

定义8(容器-节点分配矩阵) $\boldsymbol{Z}=[z_{ij}]\in\{0,1\}^{|C|\times|N|}$，其中，$z_{ij}=1$ 表示将容器 c_i 分配到节点 n_j，否则 $z_{ij}=0$。容器-节点分配矩阵用于表示每个容器是否被分配到某个节点。

集群节点资源负载均衡度：用于衡量集群中某种资源类型在各个节点上的分布均匀程度。通过计算每个节点上该资源的实际使用率与该资源在所有节点上的平均使用率之间偏差的平方的平均值(即方差)，然后取其平方根(即标准差)，衡量资源分配的均衡性。负载均衡度的计算式可表示为：

$$LB_{r_j}=\sqrt{\frac{1}{|N|}\sum_{i=1}^{|N|}\left(\frac{l_{ij}}{s_{ij}}-\frac{1}{|N|}\sum_{i'=1}^{|N|}\frac{l_{i'j}}{s_{i'j}}\right)^2} \tag{1}$$

其中，r_j 表示资源类型，l_{ij} 表示节点 n_i 上已经使用的资源 r_j，s_{ij} 表示节点 n_i 拥有的资源 r_j 的总量。负载均衡度 LB_{r_j} 值越低，表明该资源在集群中分布得越均匀，从而集群整体性能和资源利用效率越高。

多样化资源约束：确保调度到某个节点的所有容器的各种资源需求的各自总和不超过该节点剩余的各种资源的各自容量，即所有分配到节点 n_j 的容器 c_i 对资源 r_h 的需求总和不超过该节点的剩余资源 $s_{jh}-l_{jh}$。如式(2)所示：

$$\sum_{i=1}^{|C|}z_{ij}\cdot d_{ih}\leqslant s_{jh}-l_{jh},\forall j\in\{1,2,\cdots,|N|\},\forall h\in\{1,2,\cdots,|R|\} \tag{2}$$

多样化属性约束：确保容器只能被调度到满足其属性需求的节点，即容器 c_i 对属性 a_h 的需求 e_{ih} 必须小于或等于节点 n_j 在该属性上的等级值 p_{jh}，确保每个容器的属性需求都能得到满足。如式(3)所示：

$$z_{ij}\cdot e_{ih}\leqslant p_{jh},\text{if}z_{ij}=1,\forall i\in\{1,2,\cdots,|C|\},\forall j\in\{1,2,\cdots,|N|\},\forall h\in\{1,2,\cdots,|A|\} \tag{3}$$

调度约束：确保每个容器至多被调度到一个节点，即一个容器不能同时调度到多个节点。如式(4)所示：

$$\sum_{j=1}^{|N|}z_{ij}\leqslant 1,\forall i\in\{1,2,\cdots,|C|\} \tag{4}$$

因此，容器多样化需求调度问题可以表示为：

$$\begin{aligned}&\text{Min } LB_{r_j}\\&\text{s.t. 式(2)}-\text{式(4)}\end{aligned} \tag{5}$$

4 基于GMM的容器定制化调度策略

在以上集群资源负载状态和调度问题相关定义的基础上，CS-GMM以容器需求为调度导向，能够灵活应对不同场景下的容器需求，确保任意资源和属性的组合得到满足，精准匹配最适合的节点进行调度。这不仅有效满足了容器的多样化需求，还合理地分配了集群节点的资源，从而显著降低了 LB_{r_j} 值，使集群资源负载更加均衡，提升了整体调度效率。

CS-GMM 通过输入历史样本容器资源和属性需求的信息序列,对 GMM 进行训练并生成有效的聚类结果。GMM 能够捕捉并利用容器之间在资源和属性需求上的细微差异,识别出不同容器之间的共性和差异性,精准地将容器根据其资源和属性需求进行聚类。每种类型代表一组具有相似需求模式的容器群体,不同的模式分别聚焦于不同的资源和属性需求。

在此基础上,将待调度容器输入 GMM 聚类得到其对应类型,CS-GMM 进一步计算其对应类型对于每种资源和属性的独立权重。具体来说,首先计算此类型中所属容器分别对于每种资源和属性的平均值,然后把资源和属性的平均值与所有样本容器的每种资源和属性的平均值进行对比,将所得出的比值作为此类型对于每种资源和属性的独立权重。每种类型可得到 $|R|+|A|$ 个独立权重,分别对应一种资源和属性。这些权重反映了不同类型对不同资源和属性的依赖程度,使得调度策略能更加精细地调整节点的资源和属性分配。最后,得到待调度容器对应类型的独立权重后,与集群中各节点的各资源的利用率和属性等级值进行加权打分,得分最高的节点即为可供调度的最佳节点。同时,将待调度节点加入样本数据库中,以此不断更新样本数据库,使得 CS-GMM 在之后的分类中能够不断优化分类结果以及各类型的独立权重。

基于 GMM 的容器定制化调度策略的基本原理如图 1 所示。下面将详细介绍基于 GMM 的容器定制化调度策略。

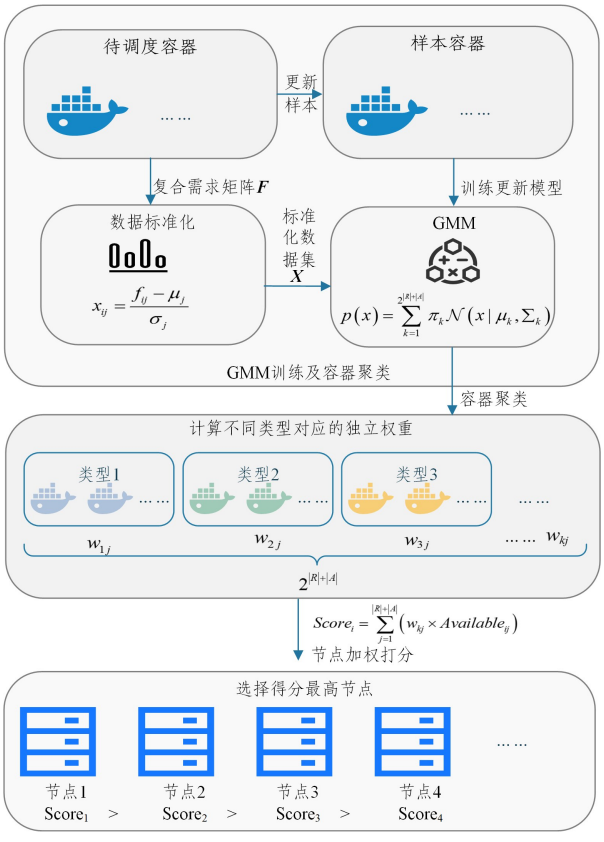


图 1　基于 GMM 的容器定制化调度策略的原理图

Fig. 1　Schematic diagram of a customized container scheduling strategy based on GMM

首先,使用历史样本容器的资源和属性需求信息序列作为训练数据,将容器资源需求矩阵和容器属性需求矩阵合并得到复合需求数据矩阵 $\boldsymbol{F}$,其维度为 $|G|\times(|R|+|A|)$,其中 $|G|$ 是样本数量,$|R|+|A|$ 是资源和属性需求种类的总数。$\boldsymbol{F}$ 的计算式为:

$$\boldsymbol{F}=[f_{ij}]\in\mathbb{R}^{|G|\times(|R|+|A|)} \tag{6}$$

为了消除不同量度单位和尺度对聚类效果的影响,确保数据在聚类分析中的有效性,对复合需求数据矩阵 $\boldsymbol{F}$ 进行了标准化处理,使 $\boldsymbol{F}$ 中每个元素具有零均值和单位方差。复合需求数据矩阵 $\boldsymbol{F}$ 中的每个元素 f_{ij} 标准化处理后的值 x_{ij} 如式(7)所示:

$$x_{ij}=\frac{f_{ij}-\mu_j}{\sigma_j} \tag{7}$$

其中,μ_j 是第 j 列的均值,σ_j 是第 j 列的标准差。标准化后的数据集由 X 表示:

$$X=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_{|G|}\} \tag{8}$$

其中,$\boldsymbol{x}_i=(x_{i1},x_{i2},\cdots,x_{i(|R|+|A|)})$ 为标准化后的数据。

其次,使用高斯混合模型(GMM)对标准化后数据集进行聚类。GMM 是一种概率模型,其优势主要体现在以下两方面:首先,GMM 能够精确地得出每个数据点属于各个分布的概率,较好地拟合容器组合中复杂的概率分布;其次,由于容器对各种资源和属性的需求体现出多样性,可以看作是不规则分布中的点,GMM 能够在不规则分布中有效地识别出不同资源和属性的需求模式,这对于容器分类至关重要。

GMM 假设标准化后数据集是由 K 个高斯分布组成的混合分布,每个高斯分布代表数据的一个潜在类型。本文将 K 设定为 $2^{|R|+|A|}$,即按资源和属性的需求量划分为高、低两个级别进行聚类,从而覆盖此类容器需求的所有资源和属性组合。例如,假设容器对存储资源和节点的可靠性存在需求,则可将存储资源的需求量分为高或低,分别用 1 和 0 表示;同样地,可靠性的需求量也分为高或低,同样用 1 和 0 表示。那么对应容器就可以划分为 2^2 类,即 4 类:11,10,01,00。对应地,GMM 概率密度函数计算式如下:

$$p(\boldsymbol{x}_i\mid\theta)=\sum_{k=1}^{2^{|R|+|A|}}\pi_k\,\mathcal{N}(x_i\mid\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) \tag{9}$$

其中,标准化后数据 $\boldsymbol{x}_i$ 是模型的输入;π_k 是第 k 个高斯分布的混合权重,且满足 $\sum_{k=1}^{2^{|R|+|A|}}\pi_k=1$;$\mathcal{N}(\boldsymbol{x}\mid\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$ 是具有均值 $\boldsymbol{\mu}_k$ 和协方差矩阵 $\boldsymbol{\Sigma}_k$ 的多元正态分布的概率密度函数;$\theta=\{\pi_k,\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\}_{k=1}^{2^{|R|+|A|}}$ 是模型的参数集,通过 K-means[28] 算法初始化参数集,以使 GMM 更好地拟合数据。

接下来,通过期望最大化(EM)算法[29]更新模型参数集,EM 算法是一种迭代算法,用于含有隐变量的概率参数模型的最大似然估计或极大后验概率估计,每次迭代包含两个步骤:Expection-Step 和 Maximization-Step[30]。

Expection-Step 通过当前的参数集 $\theta^{(t)}=\{\pi_k^{(t)},\boldsymbol{\mu}_k^{(t)},\boldsymbol{\Sigma}_k^{(t)}\}_{k=1}^{2^{|R|+|A|}}$,计算每个数据点 $\boldsymbol{x}_i$ 属于每个高斯分布的后验概率 γ_{ik},计算式为:

$$\gamma_{ik}=P(k\mid\boldsymbol{x}_i)=\frac{\pi_k^{(t)}\,\mathcal{N}(\boldsymbol{x}_i\mid\boldsymbol{\mu}_k^{(t)},\boldsymbol{\Sigma}_k^{(t)})}{\sum_{k=1}^{2^{|R|+|A|}}\pi_j^{(t)}\,\mathcal{N}(\boldsymbol{x}_i\mid\boldsymbol{\mu}_j^{(t)},\Sigma_j^{(t)})} \tag{10}$$

而 Maximization-Step 根据计算出的每个数据点所属的高斯分布的概率 γ_{ik}，以及每个高斯分布的参数集 $\theta^{(t)}$，更新新一轮的参数集 $\theta^{(t+1)}$。其中，混合权重 π_k 代表第 k 个高斯分布在模型中的权重，通过下式更新：

$$\pi_k=\frac{1}{|G|}\sum_{i=1}^{|G|}\gamma_{ik} \tag{11}$$

均值向量 $\boldsymbol{\mu}_k$ 是第 k 个高斯分布的中心，描述了该分布在特征空间中的位置，通过下式更新：

$$\boldsymbol{\mu}_k=\frac{\sum_{i=1}^{|G|}\gamma_{ik}\boldsymbol{x}_i}{\sum_{i=1}^{|G|}\gamma_{ik}} \tag{12}$$

协方差矩阵 $\boldsymbol{\Sigma}_k$ 描述了第 k 个高斯分布的形状和方向，通过下式更新：

$$\boldsymbol{\Sigma}_k=\frac{\sum_{i=1}^{|G|}\gamma_{ik}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)(\boldsymbol{x}_i-\boldsymbol{\mu}_k)^{\mathrm{T}}}{\sum_{i=1}^{|G|}\gamma_{ik}} \tag{13}$$

EM 算法会重复计算 Expection-Step 和 Maximization-Step，直到对数似然函数收敛，以找到最适合数据的高斯混合模型的参数集，训练后的 GMM 将容器聚类为 $2^{|R|+|A|}$ 种类型。当得到新容器 $\boldsymbol{x}_{\mathrm{new}}$ 的调度请求时，CS-GMM 通过训练后的 GMM 对其进行聚类，其属于每种类型的后验概率由式(14)给出，其中具有最大后验概率的类型，即为该容器的聚类结果。

$$p(k|\boldsymbol{x}_{\mathrm{new}})=\frac{\pi_k\,\mathcal{N}(\boldsymbol{x}_{\mathrm{new}}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\sum_{k=1}^{2^{|R|+|A|}}\pi_j\,\mathcal{N}(\boldsymbol{x}_{\mathrm{new}}|\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)} \tag{14}$$

然后，通过聚类后的结果得出该类型对不同资源和属性的独立权重。具体方法如下：复合需求数据矩阵 $\boldsymbol{F}$ 的平均需求 $\bar{\boldsymbol{f}}$ 由式(15)计算得出，该值由所有样本容器的复合需求的平均值得出：

$$\bar{f}=\frac{1}{|G|}\sum_{i=1}^{|G|}f_i \tag{15}$$

对于聚类类型 $k(k=1,2,3,\cdots,2^{|R|+|A|})$，$|k_k|$ 表示第 k 种类型中的容器数，计算属于该类型的容器的复合需求平均值 $\bar{f}_k$，通过将所有属于该类型的 f_i 加总，然后除以该类型中容器的数量 $|k_k|$，得到该类型的平均复合需求。计算式如下：

$$\bar{f}_k=\frac{1}{|k_k|}\sum_{i=1}^{|k_k|}f_i \tag{16}$$

通过该聚类类型的每个资源或属性的平均值和所有样本对应资源或属性的平均值的比值，可得出此类型分别对每种资源和属性的独立权重 w_{kj}，计算式如下：

$$w_{kj}=\frac{\bar{f}_{kj}}{\bar{f}_j} \tag{17}$$

其中，$\bar{f}_{kj}$ 代表第 k 种聚类类型中所有容器对第 j 种资源或属性的平均需求量，$\bar{f}_j$ 代表所有样本容器对第 j 种资源或属性的平均需求量。

通过上述方法，得到该聚类类型对于每种资源和属性的独立权重，这些权重反映了属于该类型的容器对不同资源和属性的依赖程度。最后，根据节点当前的资源空闲率和属性等级值以及待调度容器的独立权重进行加权打分，节点 i 的得分 $Score_i$ 由下式给出：

$$Score_i=\sum_{j=1}^{|R|+|A|}(w_{kj}\times Available_{ij}) \tag{18}$$

其中，w_{kj} 是容器所属聚类类型 k 对第 j 类资源或属性的独立权重，$Available_{ij}$ 是节点 i 在第 j 类资源或属性的空闲率或属性等级值。选择得分最高的节点作为调度新容器的目标节点，得分的高低直接反映了节点满足新容器需求的程度。通过精确的独立权重管理策略，确保了资源和属性分配的最优化，从而显著提升了调度系统的整体性能和资源利用效率，满足了容器的多样化需求。基于 GMM 的容器定制化调度策略的实现过程如算法 1 所示。

算法 1 基于 GMM 的容器定制化调度策略实现过程

输入：样本容器复合需求矩阵 $\mathbf{F}$，待调度容器 $\mathbf{x}_{\mathrm{new}}$

输出：最高节点得分 Max Score_i

1. $\mathbf{F}$ 标准化处理得到 X 输入 GMM；
2. 通过 K-means 算法初始化 GMM 参数集 θ；
3. while 对数似然函数未收敛
4. Expection-Step 迭代 γ_{ik}；
5. Maxi mization-Step 迭代 θ；
6. end while
7. 输入 $\mathbf{x}_{\mathrm{new}}$ 聚类得到对应类型；
8. 计算其对应独立权重；
9. $\mathbf{x}_{\mathrm{new}}$ 加入 $\mathbf{F}$；
10. while j≤(|R|+|A|)
11. 通过式(18)计算节点得分 Score_i；
12. if Max Score_i<Score_i
13. Max Score_i=Score_i；
14. end if
15. j++；
16. end while

5 原型和实验

5.1 原型设计

本文在 Kubernetes 的基础上进行融合和扩展，设计了基于 CS-GMM 的容器定制化调度框架，该框架主要由 3 个模块组成：Kube-Apiserver，CS-GMM-Scheduler 和 Monitor，如图 2 所示。其中，Kube-Apiserver 和 Monitor 在原有功能的基础上，新增记录容器资源和属性状态的功能；CS-GMM-Scheduler 融合了 CS-GMM 调度策略，使得容器调度过程更加可控。以上模块通过 Kubernetes 提供的可扩展接口能力，无缝集成到 Kubernetes 系统中，保证了框架与现有 Kubernetes 生态的兼容性，实现了基于 GMM 的容器定制化调度策略。

首先，Monitor 模块会实时监测集群中各节点的资源负载状态信息和属性值，这些信息包括 CPU、磁盘等资源的使用情况以及节点的各种属性等级值。随后，用户将新容器的资源和属性请求提交到 Kube-Apiserver。CS-GMM-Scheduler 会持续监听 Kube-Apiserver，一旦检测到新增未调度容器，模块便启动选择过程，为该容器匹配合适的调度节点。

节点选择过程分为两个阶段：预选阶段和优选阶段。

首先,在预选阶段过滤掉不符合条件的节点,例如节点的资源或属性不满足容器需求的节点。接下来,在优选阶段,CS-GMM-Scheduler 会将符合条件的节点的资源和属性状态信息,以及待调度容器的资源和属性请求作为输入参数,CS-GMM-Scheduler 为容器分类并计算每个节点的得分,并输出得分最高的节点作为最佳调度目标。完成这一过程后,CS-GMM-Scheduler 将据此在集群中执行调度操作。

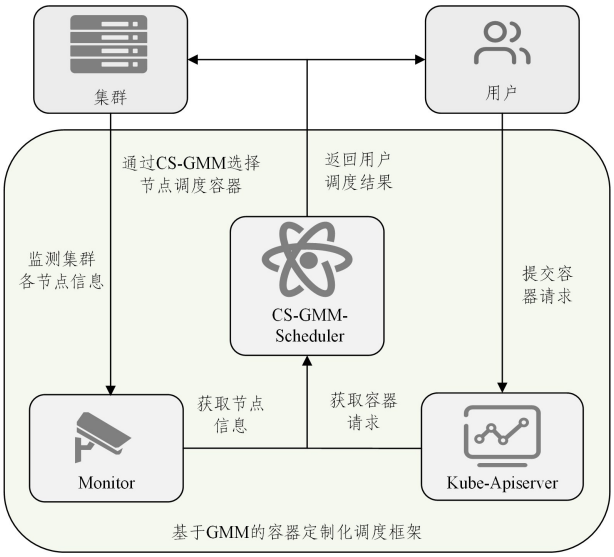


图 2　基于 CS-GMM 的容器定制化调度框架

Fig. 2　Customized container scheduling framework based on CS-GMM

5.2　实验

本文搭建了一个包含 1 个 Master 节点、3 个 Node 节点的 Kubernetes 集群[17,31],其中 Master 节点不进行容器调度,且每个节点因不同的软件因素、环境因素等,具有不同的可靠性等级,数字越大,代表节点可靠性越高。集群节点的详细资源配置信息如表 2 所列。

表 2　集群配置信息

Table 2　Cluster configuration information

节点	CPU	内存/GB	磁盘空间/GB	磁盘空间起始空闲率/%	节点可靠性等级
Master	4C 2P	8	200	95.3	2
Node01	4C 2P	8	200	96.7	2
Node02	4C 2P	8	200	92.7	3
Node03	4C 2P	8	200	96.6	4

CS-GMM 可满足容器多样化资源和属性需求,进行定制化调度。因此,基于边缘存储业务,在对数据存储可靠性有要求的存储密集型容器的历史数据的基础上,生成样本数据 1000 个,数据存储需求范围为 0～20 GB,可靠性需求范围为 4 级,作为初始样本数据对 CS-GMM 进行训练。经过训练,CS-GMM 能够对后续边缘存储业务容器进行精准分类和调度。在测试实验中,测试容器的需求范围包含在样本容器中,其磁盘空间需求依次从 2 GB 至 18 GB,共设置 9 个容器组,每组包含 4 个容器,每组内的 4 个容器的可靠性需求等级分别为 1 级、2 级、3 级和 4 级,数字越大,代表容器可靠性需求越高,且需要调度到满足其可靠性需求的节点上。容器池如表 3 所列。

表 3　容器池

Table 3　Container pool

容器组中容器磁盘需求/GB	容器数量	可靠性需求等级
2	4	1,2,3,4
4	4	1,2,3,4
6	4	1,2,3,4
8	4	1,2,3,4
10	4	1,2,3,4
12	4	1,2,3,4
14	4	1,2,3,4
16	4	1,2,3,4
18	4	1,2,3,4

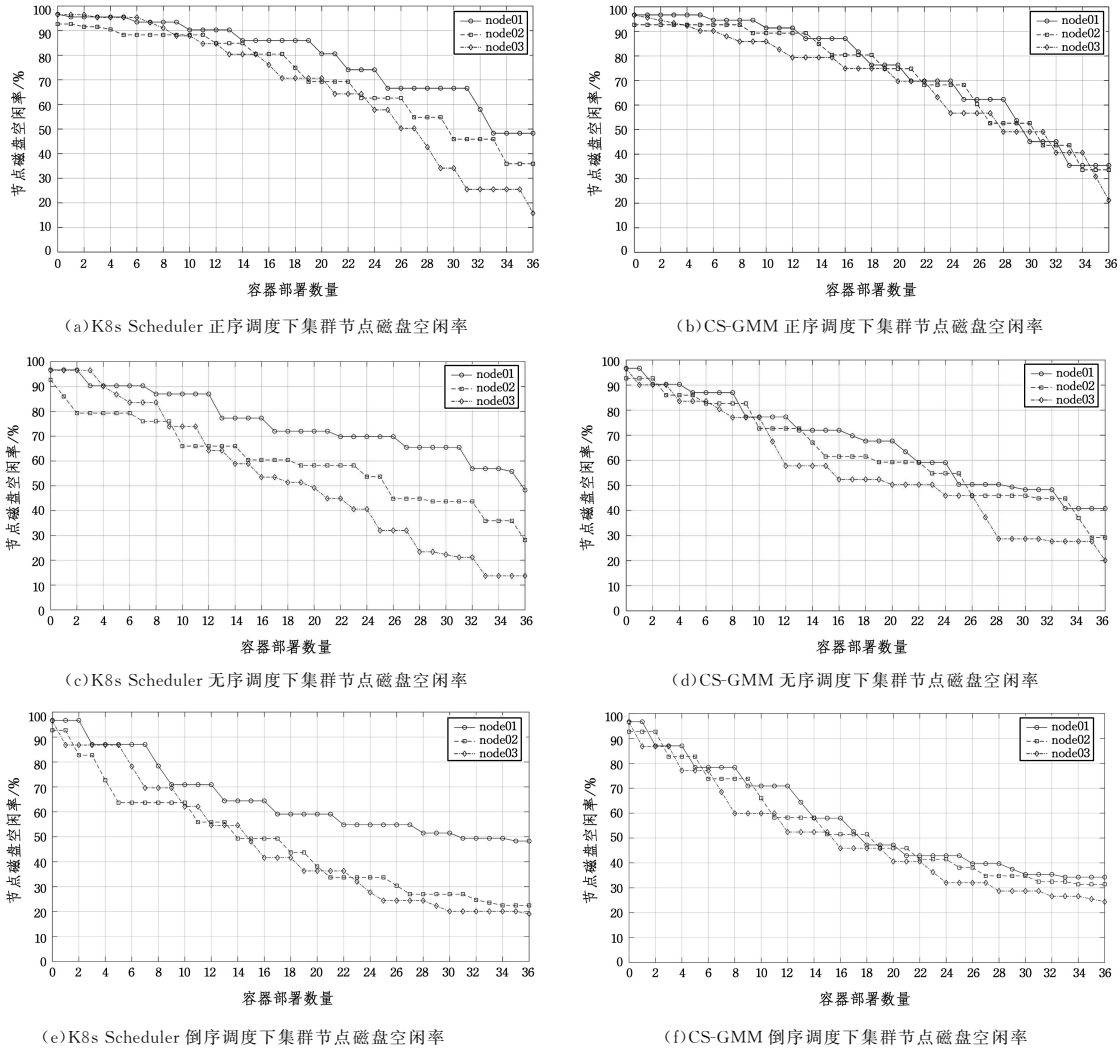
5.2.1　集群节点资源利用率效果评估

本文实验采用 CS-GMM 调度策略和 K8s Scheduler 调度策略对集群节点的资源空闲率进行对比。从容器池中选取容器依次进行调度,每个容器都待上个容器调度成功,且集群资源稳定后再进行调度。容器通过 Stress 压力测试软件模拟真实的业务操作行为中产生的磁盘负载,进行 3 组实验,分别为容器池中容器的正序、无序、倒序调度实验。正序调度即按照容器池中容器的磁盘空间需求从小到大的顺序进行调度,无序调度即从容器池中随机选择容器进行调度,倒序调度即按照容器池中容器的磁盘空间需求从大到小的顺序进行调度。通过上述 3 组实验来全面评估调度策略在各种条件下的性能表现。

图 3 是集群中 3 个 Node 节点分别在 K8s Scheduler 调度策略和 CS-GMM 调度策略下集群节点磁盘空闲率的对比情况。如图 3(a)所示,在正序调度实验中,K8s Scheduler 调度策略在处理前 15 个容器时,3 个节点的磁盘负载保持了较好的均衡;然而,在调度第 15 至第 25 个容器期间,节点间的磁盘空闲率差距开始逐步拉大;第 25 个容器调度完毕后,各节点磁盘空闲率的差异已非常显著。如图 3(c)所示,在更贴近真实复杂调度场景的无序调度实验中,K8s Scheduler 调度策略将所有容器调度完成后,观察到 3 个节点的磁盘空闲率存在显著差异。具体来看,node01 的磁盘空闲率为 48.3%,显著高于 node02 的 28.1%,而 node02 的磁盘空闲率又远高于 node03 的 13.7%。磁盘空闲率的最大值与最小值之间相差高达 34.6%,这表明集群在磁盘资源负载方面存在严重的失衡现象。如图 3(e)所示,在倒序调度实验中,K8s Scheduler 调度策略使得 node02 和 node03 节点的磁盘空闲率保持相对接近,而 node01 的磁盘空闲率则明显高于其他两个节点。上述情况的出现,归因于 K8s Scheduler 调度策略在应对容器的多样化需求和执行定制化调度时存在局限性,未能有效优化集群内特定资源的利用效率,使多个容器可能会被调度到同一节点,进而引发了资源竞争和瓶颈问题。

如图 3(b)、图 3(d)和图 3(f)所示,CS-GMM 调度策略在上述 3 组实验的整个调度过程中,充分考虑了容器的个性化资源需求,根据容器的特定资源需求进行精准调度,动态评估集群中各节点的资源利用情况,并选择最优节点调度容器,从而极

大地满足了容器的资源需求。这种精细化的调度策略不仅有效地避免了资源竞争，还确保了各节点之间的资源负载均衡。



(a)K8s Scheduler 正序调度下集群节点磁盘空闲率　(b)CS-GMM 正序调度下集群节点磁盘空闲率

(c)K8s Scheduler 无序调度下集群节点磁盘空闲率　(d)CS-GMM 无序调度下集群节点磁盘空闲率

(e)K8s Scheduler 倒序调度下集群节点磁盘空闲率　(f)CS-GMM 倒序调度下集群节点磁盘空闲率

图 3　不同调度策略下集群节点磁盘空闲率对比

Fig. 3　Comparison of disk idleness rates of cluster nodes under different scheduling strategies

5.2.2　容器可靠性需求调度成功率评估

为确保容器的运行环境满足必要的可靠性标准，节点的可靠性等级需不低于所调度容器的可靠性需求等级，以此作为容器调度成功的判定依据。

图 4 展示了在不同实验下的容器成功调度数量，可以观察到，K8s Scheduler 调度策略未能满足所有容器的可靠性需求，3 组实验的调度成功率分别为 89%，78%，78%。这是因为 K8s Scheduler 调度策略主要基于节点的可用资源来调度容器，未将容器的属性需求纳入调度策略中，导致无法很好地满足容器对可靠性等属性的需求。相比之下，CS-GMM 调度策略首先根据容器的可靠性需求对集群中的节点进行筛选，过滤掉不符合容器可靠性需求的节点。接下来，在通过可靠性属性筛选的节点中，CS-GMM 调度策略进一步综合评估各节点的资源和属性状况，选择出最匹配容器资源和属性需求的节点进行调度。这种双重筛选优化策略，不仅确保了容器的可靠性需求得到满足，还显著提高了集群资源利用率和调度效率，实现了 100%的容器调度成功率。

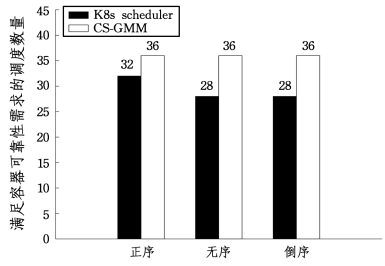


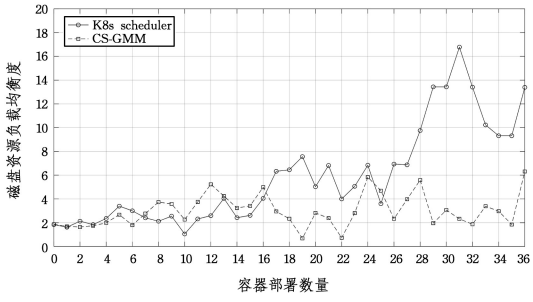
图 4　不同调度策略下满足容器可靠性需求的调度数量

Fig. 4　Number of scheduling strategies that meet container security requirements

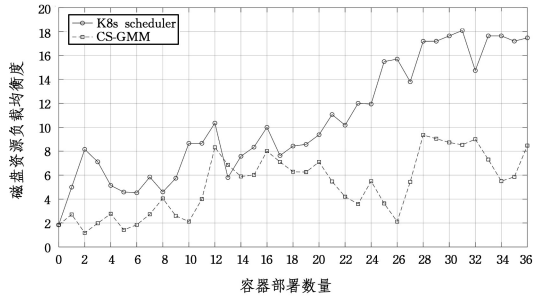
5.2.3　节点负载均衡度效果评估

为衡量集群节点磁盘资源负载均衡度，本文使用第 3 章提出的负载均衡度公式进行计算。在不同实验条件下，K8s Scheduler 调度策略和 CS-GMM 调度策略的集群磁盘资源负载均衡度对比如图 5 所示。从图中可以观察到，K8s Scheduler 调度策略的资源负载均衡度差异显著，且随着调度容器数量的增多，差异逐渐扩大，最高达到 18。相比之下，CS-GMM 调度策略的均衡度在大部分时间都保持在 8 以下，表现出较

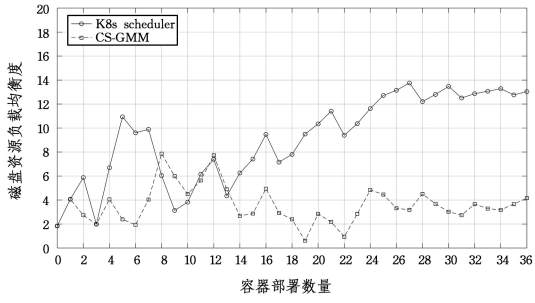
好的资源负载均衡性,有效分配了集群中的磁盘资源,避免了因资源利用不均而造成的调度瓶颈,从而提高了集群的整体性能。



(a)正序调度下节点磁盘资源负载均衡度对比



(b)无序调度下节点磁盘资源负载均衡度对比



(c)倒序调度下节点磁盘资源负载均衡度对比

图 5　不同调度策略下节点磁盘资源负载均衡度对比

Fig. 5　Comparison of load balancing of disk resources under different scheduling strategies

结束语　本文提出了一种基于高斯混合模型(GMM)的容器定制化调度策略 CS-GMM,以应对容器调度中的多样化需求和资源负载失衡问题。通过构建面向容器多样化需求和集群资源负载均衡度的模型,能够有效定制容器需求和量化集群资源和属性的状态,并通过 CS-GMM 实现对容器的精准聚类和定制化调度。通过实验进一步验证了 CS-GMM 策略在多种容器调度场景下的优越性能,其能够根据实际调度环境的变化,动态调整调度策略,为容器集群管理提供一个高效、灵活的解决方案。

本文提出的调度策略主要针对的是低耦合关系的容器调度问题,即容器间不存在复杂的依赖关系。然而,在解决高耦合性的容器组调度问题时,还存在一定的不足。因此,在下一步的研究中,将进一步评估 CS-GMM 在更加复杂的情况下的调度性能以及考虑容器间存在服务依赖关系的调度问题。

参 考 文 献

[1] CARRIÓN C. Kubernetes scheduling:Taxonomy,ongoing issues and challenges[J]. ACM Computing Surveys,2022,55(7):1-37.

[2] WU Y W,ZHANG Y,WANG T,et al. The Development of Container Technology from the Perspective of Docker Containers:A Systematic Literature Review Perspective[J]. Journal of Software,2023,34(12):5527-5551.

[3] HASSAN M,CUSTODE L L,YILDIRIM K S,et al. FedEdge:Federated Learning with Docker and Kubernetes for Scalable and Efficient Edge Computing[C]// Proceedings of the 2023 International Conference on Embedded Wireless Systems and Networks. New York:ACM,2023:339-344.

[4] NICOLAESCU A C,MASTORAKIS S,PSARAS I. Store edge networked data(SEND):A data and performance driven edge storage framework[C]// IEEE INFOCOM 2021—IEEE Conference on Computer Communications. IEEE,2021:1-10.

[5] SINGHAL S,ALI S,AWASTHY M,et al. Rock-hyrax:An energy efficient job scheduling using cluster of resources in cloud computing environment[J]. Sustainable Computing:Informatics and Systems,2024,42:100985.

[6] KATAL A,CHOUDHURY T,DAHIYA S. Energy optimized container placement for cloud data centers:a meta-heuristic approach[J]. The Journal of Supercomputing,2024,80(1):98-140.

[7] CHEN X,XIAO S. Multi-objective and parallel particle swarm optimization algorithm for container-based microservice scheduling[J]. Sensors,2021,21(18):6212.

[8] LIU B,LI J,LIN W,et al. K-PSO:An improved PSO-based container scheduling algorithm for big data applications[J]. International Journal of Network Management,2021,31(2):e2092.

[9] BOUVEYRON C,BRUNET-SAUMARD C. Model-based clustering of high-dimensional data:A review[J]. Computational Statistics & Data Analysis,2014,71:52-78.

[10] WANG Z,WANG P H,WANG B C,et al. GPU Shared Scheduling System Under Deep Learning Container Cloud Platform[J]. Computer Science,2023,50(6):86-91.

[11] SINGH N,HAMID Y,JUNEJA S,et al. Load balancing and service discovery using Docker Swarm for microservice based big data applications[J]. Journal of Cloud Computing,2023,12(1):4.

[12] MUNISWAMY S,VIGNESH R. Joint optimization of load balancing and resource allocation in cloud environment using optimal container management strategy[J]. Concurrency and Computation:Practice and Experience,2021,124:253-262.

[13] AL RESHAN M S,SYED D,ISLAM N,et al. A fast converging and globally optimized approach for load balancing in cloud computing[J]. IEEE Access,2023,11:11390-11404.

[14] ZHU L,HUANG K,FU K,et al. A priority-aware scheduling framework for heterogeneous workloads in container-based cloud[J]. Applied Intelligence,2023,53(12):15222-15245.

[15] XIAO Z,LIU K,HU M,et al. DeepCTS:A Deep Reinforcement Learning Approach for AI Container Task Scheduling[C]// Proceedings of the 2024 3rd Asia Conference on Algorithms,Computing and Machine Learning. ACM,2024:342-347.

[16] MAO Y,FU Y,ZHENG W,et al. Speculative container schedu-

ling for deep learning applications in a kubernetes cluster[J]. IEEE Systems Journal, 2021, 16(3): 3770-3781.

[17] XIE Y S, HUANG X H, CHEN N J. Self-balanced Scheduling Strategy for Container Cluster Based on Improved DQN Algorithm[J]. Computer Science, 2023, 50(4): 233-240.

[18] CHEN Y, HE S, JIN X, et al. Resource utilization and cost optimization oriented container placement for edge computing in industrial internet[J]. The Journal of Supercomputing, 2023, 79(4): 3821-3849.

[19] MOHAMMADZADEH A, MASDARI M, GHAREHCHOPOGH F S. Energy and cost-aware workflow scheduling in cloud computing data centers using a multi-objective optimization algorithm[J]. Journal of Network and Systems Management, 2021, 29(3): 31.

[20] LIU R, YANG P, LYU H, et al. Multi-objective multi-factorial evolutionary algorithm for container placement[J]. IEEE Transactions on Cloud Computing, 2021, 11(2): 1430-1445.

[21] ZHANG W, CHEN L, LUO J, et al. A two-stage container management in the cloud for optimizing the load balancing and migration cost[J]. Future Generation Computer Systems, 2022, 135: 303-314.

[22] SHEGANAKU G, SCHULTE S, WAIBEL P, et al. Cost-efficient auto-scaling of container-based elastic processes[J]. Future Generation Computer Systems, 2023, 138: 296-312.

[23] AGGARWAL A, DIMRI P, AGARWAL A, et al. Self adaptive fruit fly algorithm for multiple workflow scheduling in cloud computing environment[J]. Kybernetes, 2021, 50(6): 1704-1730.

[24] SONG Y, LIANG W F, ZHAO J, et al. Cloud Resource Scheduling Performance of Water Wave Optimization Algorithm Improved by Artificial Bee Colony Algorithm[J]. Journal of University of Jinan (Science and Technology), 2023, 37(4): 472-477.

[25] YAN J, HUANG Y, GUPTA A, et al. Energy-aware systems for real-time job scheduling in cloud data centers: A deep reinforcement learning approach[J]. Computers and Electrical Engineering, 2022, 99: 107688.

[26] DENG L, WANG Z, SUN H, et al. A deep reinforcement learning-based optimization method for long-running applications container deployment[J]. International Journal of Computers Communications & Control, 2023, 18(4): 108-125.

[27] TANG Z, LOU J, JIA W. Layer dependency-aware learning scheduling algorithms for containers in mobile edge computing [J]. IEEE Transactions on Mobile Computing, 2022, 22(6): 3444-3459.

[28] AHMED M, SERAJ R, ISLAM S M S. The k-means algorithm: A comprehensive survey and performance evaluation[J]. Electronics, 2020, 9(8): 1295.

[29] DO C B, BATZOGLOU S. What is the expectation maximization algorithm? [J]. Nature Biotechnology, 2008, 26(8): 897-899.

[30] WAN H, WANG H, SCOTNEY B, et al. A novel gaussian mixture model for classification[C]// 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC). IEEE, 2019: 3298-3303.

[31] LAI W K, WANG Y C, WEI S C. Delay-aware container scheduling in kubernetes[J]. IEEE Internet of Things Journal, 2023, 10(13): 11813-11824.

ZHOU Kai, born in 1999, postgraduate. His main research interests include cloud computing and embedded system resource virtualization.

WANG Kai, born in 1980, Ph.D, professor, Ph.D supervisor. His main research interests include network security governance and communication network security.

(责任编辑:喻藜)