张宝林, 周文勤, 高富贵. 基于容器云环境下的调度算法研究[J]. 智能计算机与应用,2025,15(4):35-44. DOI:10.20169/j.issn.2095-2163.24110504

基于容器云环境下的调度算法研究

张宝林, 周文勤, 高富贵
(天水师范学院 电子信息与电气工程学院, 甘肃 天水 741001)

摘 要: 容器调度算法作为容器编排工具的核心组成部分,负责根据一定的策略和算法,将容器分配到合适的节点上,以确保系统资源的高效利用和应用的稳定运行。但随着容器集群规模的扩大,资源碎片化和资源竞争问题日益突出,对调度算法的性能和效率提出了更高的要求。近年来,众多学者和领域内的技术人员对调度算法的优化进行了深度研究,并且也取得了一定的进展。本文基于当前主流的容器编排工具 Docker Swarm 和 Kubernetes,将调度算法的研究归为智能调度算法和基于预测的调度算法,总结分析调度算法的优化思路和方法,揭示当前研究的发展趋势和未来研究中亟待解决的问题,为相关领域的研究者提供参考和借鉴。

关键词: 容器云; 智能算法; 资源调度; Swarm; Kubernetes
中图分类号: TP391 **文献标志码**: A **文章编号**: 2095-2163(2025)04-0035-10

Research on scheduling algorithm based on container cloud environment

ZHANG Baolin, ZHOU Wenqin, GAO Fugui
(School of Electronic Information and Electronic Engineering, Tianshui Normal University, Tianshui 741001, Gansu, China)

Abstract: As the core component of the container orchestration tool, the container scheduling algorithm is responsible for allocating containers to appropriate nodes according to certain policies and algorithms to ensure the efficient utilization of system resources and the stable operation of applications, but with the expansion of the scale of container clusters, the problems of resource fragmentation and resource competition are becoming increasingly prominent, which puts forward higher requirements for the performance and efficiency of the scheduling algorithm. Based on the current mainstream container orchestration tools Docker Swarm and Kubernetes, this review divides the research on scheduling algorithms into intelligent scheduling algorithms and prediction-based scheduling algorithms, summarizes and analyzes the optimization ideas and methods of scheduling algorithms, reveals the development trend of current research and the problems that need to be solved urgently in future research, and provides reference for researchers in related fields.

Key words: container clouds; intelligent algorithms; resource scheduling; Swarm; Kubernetes

0 引 言

近些年,云计算[1-2]和容器化[3]技术发展迅速,容器云市场正在经历快速的发展和变革。随着企业对应用敏捷性、弹性和迭代效率的要求日益增长,容器技术因其提供的灵活性和弹性而受到越来越多企业的青睐。作为支持容器化应用部署、运行和管理的关键技术,容器云平台正在成为企业数字化转型的重要基础设施。艾瑞云原生系列报告[4]指出中国正处于 ICT 技术加速创新、互联网经济高速发展的阶段,移动互联网总流量和户均流量均呈现加速增长态势,预计未来数年内国内的企业级应用和云计算 SaaS 市场总规模保持稳定增长。2017~2023 年中国企业级 SaaS 市场规模如图 1 所示。容器云技术的核心用途在于构建企业级软件应用的开发、测试和生产运行环境,同时为用户提供能够应对高并发和快速访问的应用程序,随着互联网和云计算[5]行业的稳步增长,基础云服务和云应用市场也将成为众多企业首选。在此背景下,容器技术的应用范围有望进一步扩大。

基金项目: 甘肃省教育厅创新基金(2021B-197); 天水师范学院 2023 年创新基金专项(CXJ2023-11)。
作者简介: 张宝林(2000—),男,硕士研究生,主要研究方向:云计算; 高富贵(2000—),男,硕士研究生,主要研究方向:云计算。
通信作者: 周文勤(1978—),男,副教授,硕士生导师,主要研究方向:下一代互联网技术,云计算。Email:zhouwenqin@tsnu.edu.cn。
收稿日期: 2024-11-05

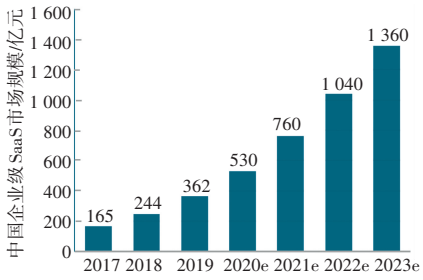

图 1 2017~2023 年中国企业级 SaaS 市场规模

Fig. 1 China´s enterprise SaaS market size from 2017 to 2023

随着容器技术的普及和应用,容器编排工具的重要性日益凸显,容器编排工具不仅可以实现用户管理和部署大规模的容器集群,自动化地进行资源调度和负载均衡,而且还可以提高系统的可靠性和稳定性。目前主流的容器编排工具有 Kubernetes[5] 和 Docker Swarm[6] 等。其中,容器调度算法是容器编排工具的核心组成部分,负责根据一定的策略和算法,将容器动态地分配到合适的节点上,以确保系统资源的高效利用和应用的稳定运行。但面对日益复杂和动态的业务需求,容器编排工具的调度算法面临着诸多挑战,如资源高效利用、负载均衡、故障恢复和弹性伸缩等。传统的调度算法,如先来先服务、最少连接法以及智能调度算法[7] 等,虽然简单易实现,但往往难以保证调度结果的最优性。目前,基于预测的容器云调度算法逐渐受到关注,通过历史数据和分析模型来预测资源需求和负载变化,从而优化容器的部署和调度。其核心在于使用机器学习、统计分析或深度学习技术来预测未来的资源使用情况和工作负载需求,结合强化学习技术,调度器可以在不断的学习和优化过程中,逐渐达到全局最优。本文将从 Docker Swarm 和 Kubernetes 两个方面开展研究,将调度算法的研究分为智能调度算法和基于预测的调度算法,总结分析其优化思路和方法。

1 容器技术

容器技术[6] 是一种软件开发的虚拟化方法,允许开发者将应用程序及其依赖项打包到一个轻量级、可移植的容器中。容器在运行时与宿主机共享内核,但与其他容器相互隔离,相比于传统的虚拟机更为高效和灵活。随着容器技术大规模的应用,传统手动管理方法无法应对大量容器和复杂应用程序的挑战,容器编排工具也得到了快速发展,其中最知名的 2 种工具是 Docker Swarm 和 Kubernetes。

1.1 Docker Swarm

Docker 公司在 2014 年 12 月初发布了 Docker 集群管理工具 Swarm,用来统一管理由多个部署有 Docker 的物理机组成的集群中的镜像和容器,采用客户端-服务器(C/S)架构。

Docker Swarm 框架如图 2 所示。Docker API 管理镜像的整个生命周期;Swarm manage CLI 是集群管理的中枢,负责整个集群的配置和维护;集群 API 定义了一套标准化的接口,主要用于任务的调度和分发;Discovery service 负责在每个节点上注册代理,并将节点的 IP 地址和端口信息上报,以便管理节点可以获取并使用这些信息。

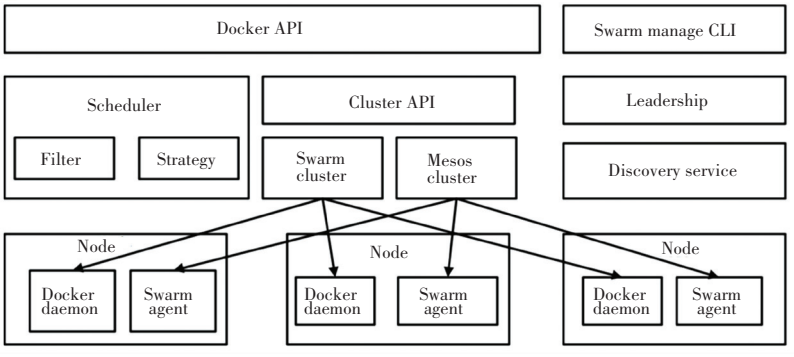

图 2 Docker Swarm 框架

Fig. 2 Docker Swarm framework

调度器(Scheduler)是 Swarm 中的一个复杂模块,在容器调度时负责选择最合适的节点来运行容器。调度器的工作流程分为 2 个主要部分:过滤(Filter)和策略(Strategy)。过滤阶段负责在创建或运行容器时确定哪些节点是可用的,用户在搭建集群时会根据节点的功能分配特定标签,过滤阶段会根据这些标签筛选出合适的节点集合,为策略阶段做准备。策略阶段则是根据预定的调度策略从过滤阶段筛选的节点中选择最佳节点。Swarm 提供了 3 种内置的调度策略:随机(Random)、分散(Spread)

和打包(Binpack)。分散策略倾向于选择资源使用较少的节点，以实现资源的均匀分布，是 Swarm 自带的默认策略。

1.2 Kubernetes

Kubernetes 作为一种开源的容器编排平台，最初由 Google[7] 开发并于 2014 年发布，现已成为 CNCF(Cloud Native Computing Foundation)的项目之一，并且是容器云领域影响力最大的开源平台之一。Kubernetes 的架构系统由 Master 节点和 Node 节点组成，Master 节点包含 kube-apiserver、etcd、kube-scheduler 和 kube-controller-manager 等核心组件，负责集群的管理和控制。Kubernetes 框架如图 3 所示，这里将展开研究分述如下。

(1)ApIServer：资源操作的唯一入口，接收用户输入的命令，提供认证、授权、API 注册和发现等机制。

(2)Scheduler：负责集群资源调度，遵循既定的调度原则，将 Pod 分配到最适合的节点上。

(3)ControllerManager：负责维护集群的状态，比如程序部署安排、故障检测、自动扩展、滚动更新等。

(4)Etcd：负责存储集群中各种资源对象的信息。

(5)Node 节点：包含 Kubelet、Kube-proxy 和容器运行时用于运行容器化的应用程序。

(6)Pod：作为最小的部署单元，由 Kubelet 在 Node 节点上创建和管理。

(7)Service：定义访问规则和负载均衡策略。

(8)Controller：监控资源状态并执行控制操作，共同实现高度可扩展、自动化的容器编排和管理功能。

(9)Kubelet：通过控制 Docker 维护容器的生命周期。

(10)Kube-proxy：负责提供集群内部的服务发现以及负载均衡。

(11)Docker：负责节点上容器的各种操作。

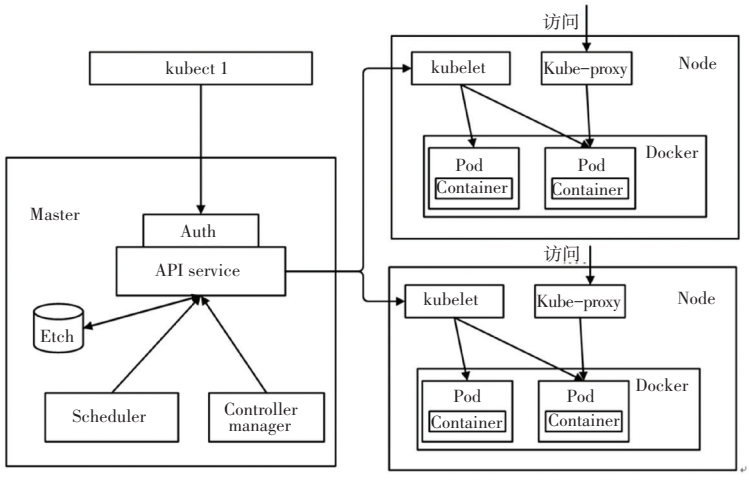

图 3　Kubernetes 框架

Fig. 3　Kubernetes framework

在 Kubernetes 集群的调度过程中，调度器通过两阶段的流程来决定 Pod 的合适宿主节点。第一阶段是过滤，其中调度器依据预设的规则，如 PodFitsHost 和 PodFitsResources 等，来筛选出符合 Pod 部署条件的节点。第二阶段是打分，此时调度器采用策略有 LeastRequestedPriority 和 BalancedResourceAllocation 等，对筛选后的节点进行评估，并结合优先级函数为每个节点打分，最终根据加权得分选择最优节点。通过这一机制不仅确保了 Pod 的调度满足特定的要求，还促进了集群资源的高效分配和负载均衡，使得 Kubernetes 的调度器能够灵活应对不同的调度场景，提升应用性能和资源利用效率。Docker Swarm 和 Kubernetes 对比见表 1。

表 1　Docker Swarm 和 Kubernetes 对比

Table 1　Comparison of Docker Swarm and Kubernetes

对比维度	Docker Swarm	Kubernetes
灵活性	低	高
扩展性	低	高
资源利用率	相对较低	高
容错率	低容错性	高容错性
适用场景	小规模集群	大规模集群

2　Docker Swarm 调度算法研究

目前 Swarm 内置的默认调度算法是扩散

(Spread),尽管 Spread 在多数情况下能够有效实现负载均衡,但也存在诸多不足:

(1)调度策略单一。面对具有特定资源需求或部署约束的应用时显得不够灵活,限制了其在复杂场景下的适用性。

(2)扩展性和可定制性较差。用户难以对默认调度算法进行深度定制或扩展,难以满足多样化的调度需求。

(3)随着集群规模的扩大,资源竞争和调度延迟问题日益凸显,可能导致服务启动时间延长和集群性能下降。

(4)Swarm 的默认调度算法缺乏全局优化视角,无法从整体上优化资源分配,容易导致资源分配不均或集群整体性能下降。在本节,将对这一领域的关键理论、方法和研究成果进行梳理总结。

2.1 基于智能算法的优化

为了解决 Swarm 默认的 3 种资源调度策略在面对高性能需求应用时的不足,鲁洪宽[8]提出了一种改进的人工蜂群算法。该策略通过实时调整节点权重,不仅提升了算法的收敛速率,还强化了其在局部搜索方面的表现。这种算法综合考量了节点资源分布、新容器的资源需求以及资源均衡分配的多维度因素,优化了单个容器的节点选择,并在多容器部署中实现了负载的均匀分布,从而提升了整个集群的资源调度效率。在 2 台物理服务器主机上创建 25 台操作系统为 Ubuntn14.04 的虚拟机作为节点进行仿真实验,其中 1 台作为主节点管理整个集群,另外 24 台为子节点。结果表明改进的调度策略能更好地实现集群的资源负载均衡。

李东光等学者[9]提出了利用蚁群优化算法对其调度进行改进,在调度任务分配前,所有工作节点首先通过一系列过滤器进行筛选,然后利用 ACO 算法替代默认的调度策略,以流水线方式分配任务。实验环境在云提供商 DigitalOcean 上形成 SwarmKit 集群上进行实验。集群由 1 个管理节点和 5 个工作节点组成,在 5 个节点上部署了 12 个 NGINX 服务器任务,结果表明,在相同集群配置下采用 ACO 算法的集群在 QPS 性能测试中实现了约 20%的性能提升。这种改进不仅促进了集群节点间资源的均衡利用,还显著提升了服务的响应速度。

林伟伟等学者[10]提出了基于遗传算法的 Docker Swarm 调度策略。该策略利用遗传算法的原理来优化容器的调度,通过将多个任务整合为单一的调度实体,每个实体的适应度评估是基于其任务的负载特性、当前节点的负载状态以及硬件的性能,以此来衡量整个集群的负载均衡性。通过模拟遗传算法的自然选择机制,能够快速筛选出接近最优的调度配置。实验环境为 4 个配备有 Ubuntu16.04 操作系统的节点部署成一个集群,其中 3 个节点为处理容器任务的从节点,1 个节点为管理集群任务调度的主节点,3 种策略各自调度 24 个容器任务。结果表明,与 Docker Swarm 现有的 Spread 策略和权重调度策略相比,这种基于遗传算法的策略在负载均衡和多任务调度效率方面都有显著提升。

在现有的集群管理策略中,通常只考虑节点的负载情况,而忽视了不同任务对资源的特定需求。这就使得在面对多任务和高复杂性负载时,往往会导致资源利用率不足,进而造成资源浪费。为了解决这一问题,李连万[11]提出了一种针对 Docker 平台的新型调度算法。该算法采用自然数编码方式,将粒子群算法中的粒子编码长度与容器任务相匹配,以实现更精细的资源分配。在算法的初始化阶段,通过随机方法确定了解空间中 NP 粒子的位置和速度,并设定了粒子群算法的参数,在迭代过程中,根据适应度函数的反馈,不断更新粒子的个体最优解和全局最优解,直至满足停止条件。为了进一步提高算法的优化能力,避免陷入局部最优解,引入了模拟退火算法,通过模拟退火过程来更新粒子的最佳值,增加了算法的探索能力和逃逸局部最优解的可能性。实验环境使用 5 台 Ubuntu16.04 版本的 Linux 系统作为 Docker 主机,通过仿真对照实验在 5 个节点上都部署 12 个 NGINX 服务器任务,结果表明,与现有策略相比,该算法能够提高集群的资源使用效率,并使得任务分配更加合理。

针对 Docker Swarm 在资源调度中只考虑节点内存大小而忽略 CPU 和网络负载的局限,卢胜林等学者[12]提出了一种权值调度算法的优化方法。该方法在 Swarm 集群中通过增加一个权值调度模块来改进容器的调度过程,该模块在 Docker Client 创建新容器时被触发,根据节点的 CPU 性能、内存使用率和网络状况等因素计算权值,并选择权值最高的节点部署容器,以此实现更合理的资源分配和提高集群性能。先后创建 18 个和 24 个内存上限为 200 M 的容器进行仿真实验,结果表明提出的权值调度算法很好地实现了 Swarm 集群中各节点的负载均衡,提高了集群的整体性能,同时也提高了集群的资源利用率。

在大数据时代背景下,Docker Swarm 的调度策

略面临着处理日益频繁的容器任务的挑战。由于 Swarm 的默认调度策略在实例化容器时依赖于先前调度活动所确定的集群状态，导致后续调度活动必须等待前一个调度完成才能启动，这种串行处理方式限制了调度效率。同时，Swarm 未能区分不同类型的容器，缺乏对容器具体需求和集群节点资源特性的优化匹配，可能导致非最优的调度结果。为了克服这些限制，林昊[13]提出了一种基于阈值的双重策略混合蛙跳算法。该算法通过动态计算节点权重，不仅提升了算法的收敛速度，还增强了局部搜索能力，综合考虑了节点资源分布、新容器资源需求和资源均衡分配。无论是单个容器部署、还是多容器部署场景，该算法都能有效地选择最佳节点并实现容器均衡分布，确保集群负载均衡。实验环境采用 8 个操作系统为 Ubuntu 的虚拟机作为集群节点，根据混合蛙跳算法的参数设置，种群规模为 20，子群的分组数为 5。结果证实，这种改进的调度策略在资源负载均衡方面表现更佳。

2.2　基于预测的算法优化

在云平台中，容器对资源的需求是多维度的，不同类型的容器可能对 CPU、内存、存储和网络等资源有不同的需求，当多个资源需求集中在某一维度的容器被调度到同一个节点时，可能会导致该维度资源的耗尽，而其他资源维度却未得到充分利用。为了解决这一问题，胡译文[14]提出了 QoS-EACO 容器资源调度策略。该策略通过考虑节点的属性信息来评估其负载状态，并构建了一个多约束条件的 QoS 模型，以此来优化蚁群算法进行资源分配。实验模拟仿真了 50 个虚拟节点和 50 个物理节点，使用的实验数据为公开数据集 PlanetLab[15]，数据记录了 2011 年 3 月 3 日至 4 月 20 日的 10 天中，一共有一万多个容器的 CPU 利用率的统计数据，结果表明 QoS-EACO 资源调度策略不仅降低服务等级协议(SLA)的违约率，同时提高资源的利用率，实现成本效益的最大化。

由于不同的容器需求会涉及到不同维度的资源，当一个节点的任意维度资源耗尽时，如果有多维资源需求的容器被启动，那么该节点将不能满足创建容器的需求，也就不能运行此容器。马晓光等学者[16]提出了一种静态平衡和动态预测相结合的容器调度算法，该算法通过计算 CPU 和内存资源的方差来评估节点的资源均衡性，方差越小表示资源分布越均匀。利用灰色模型 GM(1,1)对节点状态进行预测，以适应容器对多维资源的需求。本实验搭建 2 个 Swarm 集群，每个集群由 3 台普通 PC 组成。结果表明，优化调度算法能改善集群资源碎片过多的问题，提高资源利用率，同时能够均衡集群的动态负载。

刘梅等学者[17]针对 Swarm 在处理具有相同权值节点时采用随机分配的问题，提出了一种改进的动态调度策略。该策略通过实时收集节点数据并应用一元线性回归模型预测节点的资源使用情况，对权值相同的节点进行重新评估和排序，优先选择权值较低的节点部署容器。基于 Linux 操作系统搭建了 4 个 Swarm 集群，集群 1，2，3 采用原生的 Swarm 搭建，集群 4 采用改进后的 Swarm 搭建，分别创建等数量容器 40 个。结果表明，增加动态调度算法能够使集群中节点负载更加均衡，同时提高集群的整体资源利用率。

冯竞凯[18]提出了一种创新的负载预测框架，该框架汲取了集成学习的思想。通过对微服务资源需求的多样性进行细致分析，构建了一个动态的负载预测模型。模型特别针对负载数据的线性和非线性特征，分别采用 ARIMA 模型和改进的闪电附着过程优化算法(LAPO)来预测线性趋势和增强 LSTM 模型对非线性变化的适应性。为了进一步提升预测的准确性，还引入了 CRITIC 方法对预测结果进行加权融合，并进行误差修正。实验环境采用 5 个操作系统为 Ubuntu 的计算机作为集群节点。最终结果表明，这种融合 ARIMA 和 LAPO 优化 LSTM 的模型在负载预测的准确性上，相较于传统群体智能算法优化的 LSTM 模型有显著提升。

在容器云平台中，负载通常呈现出明显的周期性波动，而局部负载则可能表现出不稳定的特性。针对云平台资源的动态调度变化，王侯贺[19]提出了一种改进的预测算法，解决了传统 ARIMA 模型在低阶预测中准确性不足和高阶参数估计困难的问题。该模型考虑了容器与节点资源的可用性，利用灰色模型和 ARIMA 混合模型对容器负载进行预测，并据此对容器资源进行限制和水平扩展。实验使用 2 台物理机通过 VMware 创建了 3 台 Ubuntu 虚拟机，利用 Prometheus 监控集群中各节点和运行在节点中的容器资源负载，采集的频率 5 s，将采集的数据存入时序数据库 InfluxDB 中，记录一周的数据，将前 6 天的数据作为预测模型的训练集，最后 1 天的数据作为对照组。实验结果表明，该系统能够在高负载情况下确保节点中的微服务容器稳定运行，不仅提高了资源利用率，还优化了成本效益。改进 Docker Swarm 调度算法优化研究见表 2。

表 2 改进 Docker Swarm 调度算法优化研究

Table 2 Research on improved Docker Swarm scheduling algorithm

优化结果	文献	优化方法	结论	对比算法	不足
负载均衡	[9]	使用蚁群优化算法替代原始循环贪婪算法	ACO 集群 QPS 相对于 Spread 集群提高约 20%	Swarm 原算法	没有确定服务器的确定参数
	[8]	使用改进的人工蜂群算法引入节点权数来提高算法的收敛速度和局部搜索能力	使用改进的调度策略能更好地实现集群的资源负载均衡	Swarm 原算法	只考虑到选择最优的节点,并未考虑容器调度过程中的时间开销
	[13]	使用改进的基于阈值的双重策略混合蛙跳算法来提升收敛速度和局部搜索能力	改进的算法有助于提高容器调度的性能,有效地降低了集群资源的负载均衡	SFLA, TSF - SFLA, SFLA-GETD 和 DS-SFLA	未能考虑实时的剩余资源信息
	[10]	利用遗传算法筛选出全局近似最优解作为调度结果	随着任务数的增加,调度的执行效率优势越明显	Swarm 原算法,权值调度算法	模型还不够完善,需要人为设定任务的初始参数
	[17]	使用一元线性回归模型预测节点资源使用情况,然后排序优选权值小的节点	增加一个动态调度算法后,集群中每个节点的负载较为平衡	Swarm 原算法	
	[19]	设计了 ARIMA - Kalman 混合模型的 Docker Swarm 集群	节点负载更加均衡,同时提高集群的整体资源利用率		只考虑节点 CPU 与内存的使用情况
资源利用率	[14]	提出了 QoS - EACO 容器资源调度策略	在降低 SLA 违约率的同时提高资源利用率,达到成本最低	传统 ACO 算法和 MACO 算法	使用的蚁群算法没有与其他算法进行融合
	[16]	通过计算 CPU 资源和内存资源之间的方差来度量节点资源的平衡情况	优化调度算法能改善集群资源碎片过多的问题,提高资源利用率	Swarm 原算法	
	[18]	使用改进的闪电附着过程优化算法构建动态 LSTM 预测模型对负载非线性部分进行预测	在相同的迭代条件下本文对 LAPO 算法的改进与其他改进算法相比,在单峰函数和多峰函数中均表现出良好的寻优性能	Swarm 原算法	采用了集成学习的思想,并没有从预测模型的结构出发进行算法本身结构的提升
	[11]	提出了粒子群-蚁群优化算法,让 Node 和 Task 中包含更多的资源信息	相比于默认的算法资源利用率,比其他 2 种算法要高,接近 20%	Swarm 原算法	增加了容器的额外的时间开销

3 Kubernetes 调度算法研究

尽管 Kubernetes 的默认调度算法在大多数情况下能够有效地进行资源分配和负载均衡,但仍存在一些不足之处:

(1)资源维度考虑不足。默认算法主要关注 CPU 和内存资源,而忽略了网络带宽、磁盘 I/O、GPU 等其他重要资源,这可能导致资源分配不均或应用性能受限。

(2)缺乏动态调整能力。在集群负载变化时,默认算法可能无法及时优化资源分配和负载均衡,影响集群的稳定性和性能。

(3)对于具有特定需求的应用,如需要特定硬件支持或与其他 Pod 有特殊部署要求的应用,默认算法的支持不足,增加了管理的复杂性。

(4)当集群中存在大量待调度的 Pod 时,默认算法的调度效率可能会受到影响,导致调度延迟增加,影响应用的启动速度和集群的响应能力。

3.1　基于智能算法的优化

闫硕[20]提出了一种基于改进遗传-蚁群算法的初始资源调度算法。该方法通过积攒一定数量的 Pod 同时进行调度来提高资源调度后集群的 负载均衡程度。该算法改进了原始蚁群算法的启发式函数,同时设计出了一种基于奖惩机制的信息素更新方式,又根据精英保留策略与轮盘赌算法改进遗传算法的选择算子,最后根据遗传算法和蚁群算法各自的优缺点,提出了一种将遗传算法与蚁群算法进行融合的算法,将遗传算法前期快速搜索到的部分较优解转化为蚁群算法的初始信息素浓度,然后再使用蚁群算法进行最后的精细化搜索。实验将在 CloudSim 仿真云平台上进行,随机生成一定数量的待调度的 Pod 和 Node 节点。通过实验结果表明,该算法的资源调度结果在集群负载均衡度上有着更好的表现,同时在算法收敛速度上也表现更加优异。

耿棒棒等学者[21]针对 Kubernetes 默认调度算法在资源评分时只考虑 CPU 和内存,而未充分考虑 Pod 对资源的实际占用比例,提出了一种改进算法。该算法在原有资源指标基础上增加了带宽和磁盘容量,并根据这 4 类资源的占用比例对节点性能影响进行评估,以避免资源瓶颈和 Pod 非正常运行。算法将待创建的 Pod 分为 3 类,并设置不同权重,通过改进的秃鹰搜索算法(TBESK)选择最佳节点进行调度。实验采用的数据集来自文献[22],在仿真环境下, 模拟一个包含 31 个节点的 Kubernetes 集群,结果显示,在高负载情况下,TBESK 算法在负载均衡方面比默认算法提升了 24%。

于泽川[23]为解决 Kubernetes 默认调度算法在多任务处理时的局限性问题,开发了一种名为 Ku-PSO 的静态资源调度算法。该算法通过多 Pod 调度模型,引入带宽和磁盘容量评估节点负载情况,并针对 CPU、内存、带宽和磁盘这 4 类资源设置不同的 Pod 权重。利用改进的粒子群优化算法(PSO),动态调整惯性因子、个体学习因子和社会学习因子,以实现全局搜索和快速收敛。通过虚拟机的形式部署在 PC 上,集群中包括 1 个 Master 节点和 3 个用于负载 Pod 的 Node 节点,本次部署选取用于应用场景最丰富的网络服务 Web 应用,部署数量为 5 个,以 5 个为梯度,依次递增至 50 个。结果表明,Ku-PSO 在处理大量 Pod 调度时,能显著提升集群的资源利用率和负载均衡性,相较于传统 PSO 算法,消耗时间更短。

在云计算资源调度的研究领域,传统的调度算法由于其简单的调度功能和复杂的决策机制,在处理大规模集群时面临着计算负担加重和负载分配不均等问题。孙明杰[24]提出了一种新的资源调度框架,该框架基于改进的随机游走优化算法,这种方法以其较低的计算复杂性和快速的迭代能力,在全局范围内有效搜索最优解,同时减少了陷入局部最优的风险。实验环境为 6 台服务器用于搭建 Kubernetes 集群环境,定义了 10 个 CPU 密集型任务、30 个 IO 密集型任务、20 个普通任务进行调度实验。实验结果表明与智能粒子群算法和 Kubernetes 的默认调度策略相比,采用随机游走算法的调度器在负载均衡指数和负载差异指数方面均有显著下降,分别降低了约 5.56% 和 8.74%,从而实现了更加均衡的集群负载分布。

基于默认调度算法对集群节点采用了串行化的优选打分机制,导致优选阶段在整个调度过程中占用了过长的时间,姜猛[25]设计了一种基于模拟退火思想改进的自适应遗传算法(IGA),设计自适应的交叉概率和变异概率,自适应的概念则使概率能随适应度的计算值变化而改变,以跳出局部最优情况,同时也更利于优良个体的生存,在保持群体多样性的同时,保证遗传算法的收敛性。实验使用 15 台虚拟机搭建了 3 个 Kubernetes 集群,为了避免实验的偶然性,设计了 4 组实验重复组,每组重复组分别采用 5 个应用 Pod,应用镜像为基于 SpringBoot 框架开发的 Web 应用。结果表明,TLB-IGA 调度策略使用的改进遗传算法 IGA 相比于标准遗传算法 SGA 在 Kubernetes 调度问题的寻优能力上具有优越性。

刘可[26]针对 Kubernetes 默认调度策略可能导致的资源不均衡问题,提出了一种资源比例平衡(RPB)优选算法,以优化资源调度。该算法对现有节点打分机制进行了改进,通过比较待调度 Pod 的资源需求与节点的资源剩余量,选择资源匹配度更高的节点,从而减少资源碎片,并有效预防单一资源性能瓶颈,增强节点负载均衡。实验环境使用 CentOS7 的操作系统,选择了 4 个实验节点,1 个 Master 节点和 3 个 Node 节点;实验数据中一个是 500 MB 左右的压缩文件 ZipTest. rar,另一个是

1 GB 左右的视频文件 VideoTest. mp4。结果表明,采用 RPB 算法的节点在 CPU 和内存资源使用上表现出更好的均衡性,与未采用该算法的节点相比,资源使用更加合理。RPB 算法通过考虑 Pod 的具体资源需求,智能选择资源负载均衡的节点进行调度,有效避免了资源瓶颈问题。

3.2 基于预测的算法优化

王仪[27]提出了一种基于预测模型的资源均衡度调度算法,该算法基于预测模型对云平台的资源使用情况进行分析。将资源使用量区分为线性和非线性两部分,首先应用 ARIMA 时间序列模型对线性部分进行预测,通过比较 ARIMA 模型的预测结果与实际资源使用数据,得出线性部分的残差值序列,并利用 BP 神经网络来挖掘非线性部分的潜在价值。随后通过 BP 神经网络对非线性序列进行训练,调整网络参数以获得更为精确的预测值。最终将线性与非线性的预测结果进行叠加,形成综合预测模型。实验在 CentOS7 操作系统上构建 5 个主机,1 个主控制节点,1 个 Etcd 节点,1 个 Proxy 节点,2 个工作节点。数据样本集采用的是实习企业项目中智慧政务网站工作日的数据,每隔 5 s 采集一次 CPU 使用率,持续检测最近 30 min 共取得 360 个样本,把前 240 个样本作为预测模型的训练集,后 120 个样本作为测试集与预测结果进行对比分析。结果表明,资源均衡度调度算法在节点绑定阶段能够更好地适配节点,显著提升了资源均衡度指数,从而优化了资源的分配和利用效率。

张淡墨[28]提出使用 Double DQN(DDQN)强化学习算法对调度器进行改进,该算法相比于默认调度器加入了网络带宽和磁盘 I/O 两个指标,并设计了一套数据采集与接收框架使得 Master 节点可以获取各 Node 节点上的资源指标,通过深度强化学习的方法分别以缩短任务完成耗时和负载均衡为目标选择最佳动作,将 Pod 调度至最合适的节点。实验环境为 VMware 虚拟机上进行模拟试验。创建了 3 台虚拟机,其中 1 台作为 Master 节点,其余 2 台作为 Node 节点, 每轮测试均按每隔 1.5 s 的时间间隔创建一个容器,直至创建 100 个容器。结果表明,使用 DDQN 调度器后任务平均用时比原先使用 Kubernetes 默认调度器要快 2.67%。

针对 Kubernetes 默认的调度算法易产生过多资源碎片,用户的部分任务会因资源不足有较长的等待时间;并且在预选及优选环节没有考虑节点 GPU 的资源状态,在深度学习云平台的业务场景下调度不合理。王彬丞等学者[29]提出一种负载饱和调度算法。该算法通过实验确定了 GPU 和 CPU 对深度学习任务的贡献度,并据此赋予资源不同的权重,其中 GPU 资源的权重显著高于 CPU 和内存资源。通过这种方式,算法能够更有效地利用集群资源,减少任务的等待时间,并提高资源的均衡度。实验环境使用 Ubuntu 的操作系统,部署 2 个 Kubernetes 集群,1 个为实验组,1 个对照组。结果表明,该算法在资源利用率和任务调度效率方面均优于传统的调度算法。

陆威然[30]提出了一种基于博弈论-VIKOR(Game Theory-VIKOR, GTV)的 Kubernetes 资源调度算法。该算法综合考虑节点的计算机资源指标,增加网络带宽、磁盘 IO 指标,利用层次分析法、熵值法求解资源指标权重值,同时结合博弈论权重集化模型求得更客观的组合权重,将组合权重与改进的 VIKOR 算法相整合,从而计算出最优调度节点,使集群满足多指标 Pod 应用需求,并保证集群资源均衡度与资源利用率。实验环境采用 Kubernetesv1. 18. 0 版本,集群以虚拟机的形式部署在服务器上,集群部署 5 个物理节点,其中 1 个 Master 节点,4 个 Node 节点,准备 300 个测试应用 Pod,最终结果证明提出的博弈论-VIKOR 调度算法在 Kubernetes 静态资源调度上拥有更高的负载均衡度和稳定的资源利用率。

现有的调度机制只支持在应用首次部署时实现资源的配置,无法应对因为应用程序在运行过程中动态的变化,导致 Node 节点资源无法得到重复利用以及响应式资源报警机制不及时等问题。为了能够提前进行合理的资源调度以应对应用资源需求的变化,预测应用未来一段时间的资源需求情况是很有必要的。何龙等学者[31]提出了一种新颖的资源预测模型 SDLR,该模型基于 LSTM 和 RBF 网络,通过引入注意力机制,强化了对关键特征的学习,同时采用 BP 神经网络优化权重参数。这一创新使得 Kubernetes 集群能够根据模型预测动态调整资源分配,有效提升了系统的服务质量。实验环境使用 CentOS7. 8 的操作系统,并且选择 3 个节点作为实验节点,即:1 个 Master 节点和 2 个 Node 节点,设置了 2 个配置完全一样的 Kubernetes 集群,一个作为对照组,另一个作为实验组。结果表明,SDLR 模型在资源使用预测的准确性上具有显著优势,不仅提高了节点资源的利用率,还显著降低了应用的平均响应时间,增强了服务的整体质量。综上得到,改进的 Kubernetes 调度算法研究结果见表 3。

表 3　改进 Kubernetes 调度算法的研究

Table 3　Research on improved Kubernetes scheduling algorithms

优化目标	文献	优化方法	优化结果	对比算法	不足
负载均衡	[24]	改进的随机游走优化算法，在全局搜索最优分配方案，不易陷入局部最优解	相比于默认算法与粒子群优化算法，提出的随机游走优化算法使集群具有更好的负载均衡效果	Kubernetes 默认算法和 PSO 算法	调度算法并未考虑集群的能耗问题
	[20]	使用遗传算法前期快速搜索到的部分较优解转化为蚁群算法的初始信息素浓度，然后再使用蚁群算法进行最后的精细化搜索	随着待调度 Pod 数量的增加，IGAACO 算法的算法执行时间明显优于其他 2 种算法	Kubernetes 默认算法和 GA 算法	没有考虑待调度的 Pod 是否有状态且状态信息存储到本地节点上
	[21]	将 Pod 分区，通过改进的秃鹰搜索算法（TBESK）来寻找出最优节点进行调度	在集群负载较大的情况下，TBESK 算法的综合负载标准差和默认的调度算法相比提升了 24%	Kubernetes 默认算法	未充分考虑 Pod 的部署速度、成本
	[23]	改进粒子群算法中惯性因子、个体学习因子和社会学习因子	后期的收敛速度快，较传统的 PSO 算法消耗时间更短	Kubernetes 默认算法和传统 PSO 算法	没有考虑硬件的品牌和规格不同导致其相应指标实际负载的稳定性不同
	[28]	使用 Double DQN（DDQN）强化学习算法对调度器进行改进	在 2 个不同优化目标上相比默认调度器具有更好的表现	Kubernetes 默认算法和 DDQN 算法	未考虑能耗问题
资源利用率	[25]	设计了一种基于模拟退火思想改进的自适应遗传算法，结合模拟退火思想	相比于标准遗传算法在 Kubernetes 调度问题上具有更好的寻优能力	Kubernetes 默认算法和 GA 算法	执行计算逻辑所带来的工作节点负载上升
	[30]	利用层次分析法、熵值法求解指标权重值，应用于博弈论权重集化模型求得更客观的组合权重	可以减少资源碎片的产生，减少资源浪费，提高集群中的资源利用率	Kubernetes 默认调度算法、VIKOR 算法和 GTV 算法	未进一步扩展研究的适用范围
	[26]	提出了资源比例平衡算法，在优选打分时加入待调度 Pod 对不同资源消耗比例与节点所剩资源比例匹配度的分值	CPU 资源使用率和内存使用率都比较均衡	Kubernetes 默认算法	Pod 所需资源的取值不够精准。参与计算的资源不够全面
	[29]	面向深度学习的负载饱和调度算法	自定义负载饱和调度器能够将 GPU 显存利用率平均提升 6.85%，让训练任务等待时间下降 429 s	Kubernetes 默认算法	
	[31]	提出了一种分解式 ARIMA - LSTM 资源预测模型	调度策略在任务请求数量的增多过程中，其集群资源的综合利用率显著提升	WSLB 算法，Kubernetes 默认算法	稳定性在不同的集群环境中的结果存在极大的不确定性
	[27]	提出了一种基于神经网络的资源预测模型及基于资源预测模型的动态资源调度策略	有效地提高了 Node 节点的资源利用率、应用的服务质量和平均响应时间	Kubernetes 默认算法	只考虑了内存使用量和 CPU 使用率

4　结束语

本文从当前主流的容器编排工具 Docker Swarm 和 Kubernetes 出发，将调度算法的研究分为智能调度算法和基于预测的调度算法，总结分析了调度算法的优化思路和方法，并得出以下几点优化方法与思路：

（1）除了传统的 CPU 和内存资源外，加入了网络、磁盘、GPU 等指标，实现多目标优化。

（2）基于对各种资源指标的权重重新赋值，根据权值的大小对节点重新评估和排序，权值计算可以分为机器学习和智能计算。

（3）通过结合多种启发式算法，如遗传算法、粒子群优化、模拟退火等，设计出新的混合算法。

(4)基于预测的容器云资源分配,利用历史数据和预测模型,提前预测应用的资源需求,动态调整资源分配。

随着云计算技术的持续进步,未来的容器云调度算法正迈向智能化、自适应性和多目标优化的新纪元[32]。这些先进的算法将融合机器学习和人工智能技术,实现对容器资源需求动态变化的自动学习和适应[33]。近期,苏州空天信息研究院二十三室云平台技术团队研究开发基于强化学习的多目标任务调度框架和优先级感知的容器化工作负载调度框架 PA-CCWS 取得进展,通过引入强化学习技术,优化了容器云平台中的任务调度效率与资源管理[34]。在云环境的快速变化面前,调度算法必须具备更敏捷的响应能力和更强的弹性,综合考虑任务完成时间、资源利用率、成本和服务质量等多个关键因素,以做出更加精准和高效的调度决策,迅速适应资源和任务的波动。

此外,随着边缘计算和物联网技术的迅猛发展,调度算法将面临处理更多样化计算需求和复杂网络约束的新挑战。这些挑战不仅为算法设计者带来了新的难题,也为整个容器云资源调度领域提出了新的机遇。预计容器云资源调度技术将拥有更广阔的应用前景,并激发更深层次的创新潜力,推动该领域向更高效、更智能的方向发展。

参考文献

[1] 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.

[2] BUYYA R, YEO C S, VENUGOPALS, et al. Cloud computing and e-merging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility [J]. Future Generation Computer Systems, 2009, 25(6): 599-616.

[3] BERNSTEIN D. Containers and cloud: From LXC to Docker to Kubernetes[J]. IEEE Cloud Computing, 2014, 1(3): 81-84.

[4] 上海艾瑞市场咨询股份有限公司. 中国容器云市场研究报告艾瑞云原生系列报告(一)2020 年[EB/OL]. [2020-12-23]. https://stock. finance. sina. com. cn/stock/go. php/vReport _ show/kind/industry/rptid/662075480572/index. phtml.

[5] 方巍,文学志,潘吴斌,等. 云计算:概念、技术及应用研究综述[J]. 南京信息工程大学学报(自然科学版),2012,4(4):351-361.

[6] MARATHE N, GANDHI A, SHHAH J M. Docker swarm and kubernetes in cloud computing environment [C]//2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI). Piscataway, NJ: IEEE, 2019: 179-184.

[7] 苏鸿斌. 基于改进灰狼算法的云资源调度策略研究[J]. 智能计算机与应用,2024,14(2):28-34.

[8] 鲁洪宽. 基于改进的人工蜂群算法 Docker Swarm 集群调度方法设计与实现[D]. 济南:山东大学,2018.

[9] 李东光,刘智平,姜雨菲. 蚁群优化算法的 Docker 集群调度策略[J]. 西安工业大学学报, 2019, 39(3):330-335.

[10] 林伟伟,王泽涛. 基于遗传算法的 Docker 集群调度策略[J]. 华南理工大学学报(自然科学版),2018,46(3):127-133.

[11] 李连万. Docker 云平台的资源调度算法研究[D]. 南京:南京邮电大学,2019.

[12] 卢胜林,倪明,张翰博. 基于 Docker Swarm 集群的调度策略优化[J]. 信息技术,2016(7):147-151.

[13] 林昊. 基于改进混合蛙跳算法的 Swarm 集群调度策略研究[D]. 广州:广东工业大学,2020.

[14] 胡译文. 基于 Docker 的容器云资源调度策略研究[D]. 重庆:重庆邮电大学,2021.

[15] PARK K S, PAI V S. CoMon: A mostly-scalable monitoring system for PlanetLab [J]. ACM Sigops Operating Systems Review, 2006, 40(1): 65-74.

[16] 马晓光,刘钊远. 一种适用于 Docker Swarm 集群的调度策略和算法[J]. 计算机应用与软件,2017,34(5): 283-287.

[17] 刘梅,高岑,田月,等. 基于 Docker Swarm 集群的调度策略优化算法[J]. 计算机系统应用, 2018, 27(9):199-204.

[18] 冯竞凯. 容器云环境下基于负载预测的弹性伸缩研究[D]. 天津:河北工业大学,2020.

[19] 王侯贺. 基于 Docker Swarm 集群的高可用性平台研究与实现[D]. 天津:河北工业大学,2019.

[20] 闫硕. 基于 Kubernetes 的资源调度优化算法的研究与实现[D]. 北京:北京邮电大学,2023.

[21] 耿棒棒,王勇. 基于改进秃鹰搜索算法的 Kubernetes 资源调度应用[J]. 计算机系统应用,2023, 32(4):187-196.

[22] 张文辉,王子辰. 基于组合权重 TOPSIS 的 Kubernetes 调度算法[J]. 计算机系统应用,2022,31(1):195-203.

[23] 于泽川. 基于 Kubernetes 的资源调度策略研究与改进[D]. 杭州:浙江理工大学,2022.

[24] 孙明杰. 基于 Kubernetes 的云资源调度算法的研究与实现[D]. 北京:北京邮电大学,2022.

[25] 姜猛. 基于 Pod 流量数据的 Kubernetes 平台调度算法的研究[D]. 长春:长春工业大学,2023.

[26] 刘可. 基于 Kubernetes 集群的资源调度策略研究[D]. 武汉:武汉纺织大学,2022.

[27] 王仪. 基于 Kubernetes 的容器云平台资源调度策略研究[D]. 西安:西安理工大学,2021.

[28] 张淡墨. 基于强化学习的 Kubernetes 调度策略研究[D]. 北京:中国电子科技集团公司电子科学研究院, 2023.

[29] 王彬丞,王平辉,武文博,等. 面向深度学习 Kubernetes 负载饱和调度算法设计与实现[J]. 郑州大学学报(理学版),2024,56(4):21-27.

[30] 陆威然. 基于 Kubernetes 的容器云资源调度策略研究与改进[D]. 杭州:浙江理工大学,2023.

[31] 何龙,刘晓洁. 一种基于应用历史记录的 Kubernetes 调度算法[J]. 数据通信, 2019(3): 33-36.

[32] 林伟伟,齐德昱. 云计算资源调度研究综述[J]. 计算机科学, 2012, 39(10):1-6.

[33] 王占丰,张林杰,吕博,等. 基于机器学习的云计算资源调度综述[J]. 无线电通信技术, 2022, 48(2):213-222.

[34] 中国科学院空天信息创新研究院. 苏研院研究团队通过强化学习优化基于容器云技术的任务调度算法[EB/OL]. (2024-11-12). [2024-11-23] https://aircas. ac. cn/dtxw/kydt/202411/t20241112_7438594. html.