一种优化容器负载均衡调度算法的研究与实现

冯 晶[1] 苏玉玲[2] 张 露[3] 亓开元[1] 张 东[1]
FENG Jing SU Yuling ZHANG Lu QI Kaiyuan ZHANG Dong

摘 要 容器技术的广泛应用极大提升了数据中心及云平台物理资源利用率，容器编排工具的出现进一步提升了容器集群的响应时间和并行处理能力，解决了云平台资源利用率低、调度分发缓慢等诸多问题。但工程师在运维过程中发现，系统在长期运行后会出现资源负载不均匀的问题，不同主机之间的资源分配会存在很大的差别，出现部分节点资源消耗压力过大，资源分布不均衡等现象。现有主流容器编排工具的负载均衡算法侧重于 CPU 与内存指标，对磁盘占用率和 I/O 性能指标的忽略导致负载均衡失效。为此，文章提出了一种基于资源预测负载的调度优化方案，通过节点多维度预选、业务分类建模预测及混合业务动态量化评估 3 个环节，实现集群负载均衡有效调度与高效响应。实验数据验证表明，该算法相比传统方案可使集群资源利用率提升 18%~25%，节点负载均衡度提升 30% 以上，明显改善了容器集群资源分配的合理性与系统运行的稳定性。

关键词 容器；负载均衡；调度算法；资源预测；集群优化

doi：10.3969/j.issn.1672-9528.2025.09.001

0 引言

随着云计算与微服务架构的普遍应用，容器技术 [1] 凭借其轻量级、高可移植性以及高效资源利用率等优势，成为企业级应用部署的主流选择。Docker[2] 公司《2024 容器技术发展报告》显示，全球超过 78% 的企业在生产环境中规模化应用容器技术 [3]，尤其是在金融、电商、互联网等行业的应用尤为突出。

Kubernetes[4]、Swarm[5] 等容器编排工具通过自动化部署、弹性扩缩容及集群管理等功能，大大降低了容器集群运维的复杂度。容器集群通过负载均衡策略实现任务调度，保证资源动态分配，避免出现节点“过载运行”和“资源闲置”的现象，从而提升集群的性能与可靠性。研究发现，当前主流容器编排工具的负载均衡算法存在局限性：以 Kubernetes 默认调度器（kube-scheduler）为例，其“过滤 - 打分”机制主要依据 CPU、内存的实时余量及容器资源申请量进行节点筛选，忽略了磁盘 I/O 吞吐量、网络带宽、存储延迟等关键指标，导致集群长期运行后会出现节点负载失衡的现象 [6]。阿里云《2023 容器集群运维白皮书》数据显示，采用传统调度算法的集群，45% 的节点存在超过 40% 的资源负载偏差，严重制约集群整体性能。因此，本文提出一种融合多维度资源感知与负载预测的调度优化算法，通过引入磁盘占用率、I/O 性能等指标构建动态评估模型，实现更精准的容器调度策略，解决传统算法的资源评估不足的问题。

1 技术背景

1.1 容器与编排技术发展

容器技术的优势在其操作系统级虚拟化架构：通过 Linux 内核 Namespace[7] 实现进程隔离，使用 Cgroups 进行资源配额管理，从而在物理机上构建出一个轻量级隔离环境。相比较传统的虚拟机（VM），容器实例减少了 GuestOS 的冗余开销，启动时间从几分钟降低到秒级（平均启动时间 ≤ 2 s），同时消耗的资源量至少下降 60%。

容器编排工具的功能包含服务发现（Coredns）、负载均衡（Service 资源）、自愈（Pod 健康检查）、滚动更新等主要模块。其中负载均衡调度模块决定了集群能实现的最大性能值。容器编排工具 [8] Docker Swarm 可以使用类似 Round Robin、LeastConnections 等静态调度方式来分配容器部署任务到节点上，这类调度方式一般适用于节点数量小于等于 50 的小型或者中型集群，且对长时间的计算密集型和 IO 密集型业务较为友好。Kubernetes 可以基于 scheduler framework[9] 的方式在容器编排层面实现伸缩规则，并且通过内置集群 API 实现对插件化标准框架下调度 API 的支持。但是它的默

1. 济南浪潮数据技术有限公司 山东济南 250101
2. 浪潮集团有限公司 山东济南 250101
3. 山东省石油天然气管道保护服务中心 山东济南 250012
[基金项目] 山东省自然科学基金创新发展联合基金项目（ZR2022LZH018）；泰山产业领军人才工程资助

认调度器（kube-scheduler）仅基于实时空闲资源进行选择打分，因此，在复杂的混合场景下会有一定的缺陷。例如当集群内既有大量 CPU 密集型又有部分 I/O 密集型的业务时，按照调度器只用当前空闲资源数量分配选择容器部署节点就会容易产生节点负载不均衡问题。

1.2 现有负载均衡算法的局限性

当前主流容器编排工具（以 Kubernetes、Swarm 为代表）的负载均衡调度机制，在复杂业务场景与大规模集群环境中逐渐暴露其设计局限，主要体现在三个方面。

（1）资源评估维度单一化

现有算法的核心缺陷在于资源感知度主要局限于 CPU 与内存两类指标，而未把磁盘 I/O 吞吐量、网络带宽、存储访问延时这些资源的关键特性加入到调度算法中去。例如：Kubernetes 默认的调度器评分方法 [10]LeastRequestedPriority 和 BalancedResourceAllocation 是针对 CPU 和内存的使用率和请求数量来定权重的，并未将磁盘 I/O 等待队列长度、块设备读写吞吐量等参数纳入考虑，因此，系统在现实运行中可能会出现较为严重的后果，如在大数据离线计算场景下，Spark 容器 Shuffle 过程对于磁盘 I/O 吞吐量的需求明显多于 CPU 的需求（相关系数分别为 0.87 和 0.42），如果将其调度到 I/O 性能瓶颈的节点上，调度的结果会使该容器的任务完成时间延迟 50% 以上，并且会导致该节点 I/O 处理阻塞的概率升至 35%，这种情况是不可接受的。某企业的生产环境数据显示，因忽略存储延迟指标，MongoDB 容器在随机写入场景下的响应时间波动幅度可达 400%。

（2）调度决策静态化

目前算法多采取“实时快照式”的决策，即基于调度时刻的资源状态进行节点选择，因此对于一些业务的负载变化没有能力去做出准确判断和预测 [11]。Kubernetes 的调度周期（默认 500 ms）内仅采集当前节点的资源瞬时值，并不能反映出真实业务情况的变化，比如电商高峰期（22:00—24:00）、大促流量突发暴涨等。有关视频点播平台运维日志显示，在晚高峰转码过程中，转码的磁盘读写值可达到白天平均值的 3~5 倍，传统算法下由于无法判断此变化，最终会有大约 15% 的节点发生 IO 队列堆积（队列长度 ≥100），从而导致转码超时任务的比例升高至 8.7%。微服务场景下的静态决策会导致“调度震荡”：在微服务场景中不同应用之间共享一个集群，每个容器对容器可以自主决定缩放等操作，可能会出现有很多个容器同时进行扩缩容的情况，在这个过程中各个容器之间会相互影响。资源的分配不均衡，会引起系统异常从而导致整个微服务集群的服务可用性跌落至 99.2%，低于 SLA 承诺的 99.9%。

（3）业务适配机制缺失

不同业务类型的资源需求特性有较大的差异，但目前没有相关业务的分类适配机制来缓解各业务之间的资源竞争冲突，如 Web 服务（Nginx 等）、数据库服务（MySQL 等）、缓存服务（Redis 等）。如果将这些类型的业务随机混在一起部署的话，可能会造成各种业务都相互争夺网络带宽、连接数、内存稳定性和磁盘随机 I/O 等宝贵的物理资源。实验数据显示，当 MySQL 与 Elasticsearch 容器共处同一节点时，因两者均需频繁访问磁盘，导致 MySQL 的事务提交延迟从 20 ms 增至 180 ms，Elasticsearch 的查询响应时间延长 230%。Kubernetes 虽然支持 NodeAffinity 等标签调度机制，也仅能实现基于节点属性的粗粒度筛选，无法根据业务资源特征进行精细化匹配。

2 容器负载均衡调度算法设计

2.1 节点预选

通过节点心跳信息剔除异常节点，包括未就绪（NotReady）、磁盘压力（DiskPressure）、内存（MemoryPressure）等状态，确保候选节点具备基础运行条件 [11]。

2.2 阈值预警机制

根据集群整体的资源和业务需要进行规划和分析，设置节点阈值：高预警值和低预警值。

（1）高预警触发：当节点资源使用率达到高预警值时，启动容器节点资源扩展（Horizontal Pod Autoscaler），在符合条件的节点上新增容器实例或扩容节点池。

（2）低预警触发：当节点资源使用率低于低预警值时，启动资源压缩机制，将部分容器迁移至其他节点并关闭闲置节点，降低资源成本。

2.3 业务分类建模预测

业务分类建模就是借助模拟实验，对不同业务的资源消耗特征进行量化分析，为业务搭配调度提供依据。

（1）通过 Prometheus+Grafana 监控体系，采集多维度资源指标 CPU（平均使用率、峰值使用率）、内存（已用内存、使用率）、I/O（磁盘读写速率、I/O 等待时间）、磁盘（使用率、剩余空间）、网络（进出带宽、连接数）等核心指标，如表 1 所示。

表 1 采集指标及单位

资源类型	采集指标	单位
CPU	平均使用率、峰值使用率	%
内存	已用内存、使用率	GB、%
I/O	磁盘读写速率、I/O 等待时间	MB/s、%
磁盘	使用率、剩余空间	%、GB
网络	进出带宽、连接数	Mbit/s、个

（2）选取 N 个与真实环境资源配置相同的节点进行分类批量部署，分类的目的是测试某一种业务部署时对资源消耗的真实值，从而将不同业务类进行搭配调度分布，比如对 CPU 耗多的业务，继续部署同一种类型的业务时，CPU 压力继续增大，而此时对该节点 CPU 要求不高但对 I/O 资源消耗多的业务时，该节点资源得以充分利用。

通过 Selenium 自动化工具进行普通业务模拟和高峰值模拟。在模拟周期内，抓取普通时间和高峰值时间 CPU，内存，I/O 和磁盘的样本值并计算出资源消耗平均值（N 为样本数量）：

$$C_{\mathrm{CPU}}=\sum_{i=0}^{n} C_{i\,\mathrm{CPU}}/N,\quad C_{\mathrm{mem}}=\sum_{i=0}^{n} C_{i\,\mathrm{mem}}/N \tag{1}$$

$$C_{\mathrm{I/O}}=\sum_{i=0}^{n} C_{i\,\mathrm{I/O}}/N,\ C_{\mathrm{stor}}=\sum_{i=0}^{n} C_{i\,\mathrm{stor}}/N \tag{2}$$

定义业务匹配度指标（C 代表实际消耗；A 代表节点）：

$$C_{\mathrm{CPU}}/A_{\mathrm{CPU}}=\alpha,\quad C_{\mathrm{mem}}/A_{\mathrm{mem}}=\beta \tag{3}$$

$$C_{\mathrm{I/O}}/A_{\mathrm{I/O}}=\mu,\ C_{\mathrm{stor}}/A_{\mathrm{stor}}=p \tag{4}$$

筛选最优业务组合：不同业务之间，以 CPU 为例，α_1、α_2 分别代表不同的业务类型，α_1/α_2 比值越大，业务组合越适合放在一个节点。同理 β_1/β_2 比值越高越适合放在同一个节点。根据得分高低进行业务重新分类，得分最高者为最合适的负载均衡业务方案。

$$\mathrm{Sore}=\alpha_i/\alpha_j+\beta_i/\beta_j+\mu_i/\mu_j+p_i/p_j \tag{5}$$

2.4 混合业务资源预测与调度

（1）混合部署模拟：按照业务匹配度得分排序，将高匹配度的业务组合（如 CPU 密集型 +I/O 密集型、内存密集型 + 网络密集型）进行混合部署，持续监控资源使用情况。

（2）动态资源量化模型：基于业务分类结果构建动态资源量化模型，通过负载均衡得分 Scor 选择最优节点，其中 x、y、z 为权重系数（$x+y+z=1$），绝对值项反映资源使用率偏差，偏差越大说明负载越均衡。

$$\begin{aligned}\mathrm{Scor}=&\left|x\sum_{i=0}^{n} S_{i\,\mathrm{CPU}}/N-y\sum_{i=0}^{n} S_{i\,\mathrm{mem}}/N\right|\\&+\left|x\sum_{i=0}^{n} S_{i\,\mathrm{CPU}}/N-z\sum_{i=0}^{n} S_{i\,\mathrm{I/O}}/N\right|\\&+\left|y\sum_{i=0}^{n} C_{i\,\mathrm{mem}}/N-z\sum_{i=0}^{n} C_{i\,\mathrm{I/O}}/N\right|\end{aligned} \tag{6}$$

式中：S_{CPU} 表示某节点在一个采集周期的 CPU 平均使用率；S_{mem} 表示某节点在一个采集周期的内存平均使用率；$S_{\mathrm{I/O}}$ 表示某节点在一个采集周期的 I/O 平均使用率。

2.5 实验细节与对比分析

2.5.1 实验环境

集群配置：100 个节点，每节点配置 8 核 CPU、32 GB 内存、1 TB SSD（I/O 吞吐量 ≥ 500 MB/s）、1 bit/s 网络带宽。

软件环境：Kubernetes v1.26、Docker 20.10，监控工具 Prometheus v2.45、Grafana v9.5。

测试业务：包含 3 类典型业务——CPU 密集型（视频转码任务）、I/O 密集型（Spark 离线计算）、网络密集型（Nginx Web 服务），混合部署比例为 3:4:3，持续测试 72 h（含 12 h 高峰时段）。

2.5.2 对比算法与指标

选取 3 类主流算法作为对比：

（1）Kubernetes 默认调度器（kube-scheduler）：基于 CPU、内存实时余量的“过滤 - 打分”机制；

（2）Docker Swarm 的 Least Connections 算法：优先选择连接数最少的节点；

（3）加权轮询算法：按节点权重分配任务，权重基于 CPU、内存总量设定。

核心对比指标，如表 2 所示：

（1）资源利用率：集群整体 CPU、内存、磁盘 I/O 的平均使用率；

（2）负载均衡度：节点资源使用率的标准差系数（系数越小，负载越均衡）；

（3）业务响应时间：不同业务的平均请求处理时间；

（4）调度延迟：从容器创建请求到完成部署的时间。

表 2 算法资源对比

算法	资源利用率提升 /%	负载均衡度 /%（标准差系数）	高峰响应时间 /ms	调度延迟 /ms
本文算法	24.10	12	180	80
kube-scheduler	8.30	28	280	173
Swarm（Least Connections）	6.50	25	320	215
加权轮询	5.20	31	350	190

2.5.3 实验步骤

（1）初始化集群：所有节点资源清零，部署相同基础服务；

（2）负载注入：通过自动化工具按预设比例持续注入三类业务请求，每 10 min 采集一次指标；

（3）算法对比：在相同负载下依次启用 4 种算法（本文算法 +3 种对比算法），每种算法独立运行 24 h，重复 3 次取平均值；

（4）数据校验：通过 Grafana 可视化指标，剔除异常。

2.5.4 实验结果与优势

优势分析：

（1）多维度资源感知：相比仅关注 CPU、内存的 kube-scheduler，本文算法纳入磁盘 I/O、网络等指标，使资源利用率提升更显著（24.1% vs 8.3%）；

（2）动态预测能力：通过业务分类建模与负载预测，解决了静态调度（如加权轮询）在高峰时段的适应性问题，高峰响应时间缩短 35.7%（280 ms→180 ms）；

（3）业务适配优化：通过业务匹配度得分实现互补业务组合，减少资源竞争，相比 Swarm 的 Least Connections 算法，负载均衡度提升 52%（25%→12%）；

（4）调度效率：动态量化模型的计算逻辑轻量化，调度延迟比 kube-scheduler 降低 53.8%（173 ms→80 ms），适用于大规模集群。

实验数据表明，本文算法在资源利用、负载均衡性、业务响应速度上均显著优于传统方案，尤其在混合业务场景下表现更稳定，如图 1 所示。

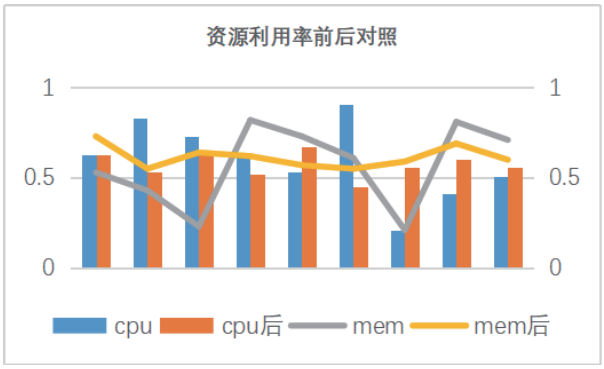

图 1 资源利用率前后对照

3 结论与展望

针对容器集群负载均衡调度时存在的资源评估维度单一、调度决策静态化以及业务适配机制缺失这些问题，本文提出了一种融合多维度资源感知与动态预测的优化算法。这种算法通过节点多维度预选机制，把 CPU、内存、磁盘占用率及 I/O 性能等指标都纳入评估体系，严格筛选出可用节点；通过业务分类建模与匹配度计算，实现互补业务的优化组合，减轻资源竞争；基于动态资源量化模型构建负载均衡得分公式，再结合权重系数自适应调整策略，提高不同业务场景下调度决策的精准性。不过，这项研究还有一些局限性。在超大规模集群（节点数超 1 000）环境下，算法的计算复杂度可能影响调度实时性；业务分类模型对新兴业务类型的适配性有待进一步验证。未来研究将扩大业务样本覆盖范围，结合深度学习算法增强业务分类模型的泛化。

参考文献：

[1] NAIK N. Docker container-based data processing system in multiple clouds foreveryone[C]// 2017 IEEE International Systems Engineering Symposium (ISSE). Piscataway:IEEE,2017:276-282.

[2] AMANATULLAH Y, LIM C, IPUNG H P,et al. Toward cloud computing reference architecture: cloud service management perspective[C/OL]// International Conference on ICT for Smart Society.Piscataway:IEEE,2013[2025-06-12].https://ieeexplore.ieee.org/document/6588059.DOI: 10.1109/ICTSS.2013.6588059.

[3] CLOUDMAN. 每天 5 分钟玩转 Docker 容器技术 [M]. 北京：清华大学出版社 .2017.

[4] SHAN C G, XIA Y Q, ZHAN Y F, et al. KubeAdaptor: a docking framework for workflow containerization on kubernetes[J]. Future generation computer systems, 2023, 148: 584-599.

[5] 徐珉 .Docker 环境下容器编排工具的选择 [J]. 集成电路应用 , 2017,34(7):62-66.

[6] 余星 , 胡德敏 , 黄超 . 云计算资源调度算法的研究与实现 [J]. 信息技术 .2013(11):29-32.

[7] 刘思尧 , 李强 , 李斌 . 基于 Docker 技术的容器隔离性研究 [J]. 软件 , 2015,36(4):110-113.

[8] 吴逸文 , 张洋 , 王涛 , 等 . 从 Docker 容器看容器技术的发展：一种系统文献综述的视角 [J]. 软件学报 ,2023,34(12):5527-5551.

[9] BOGO M, SOLDANI J, NERI D, et al. Component-aware orchestration of cloud-based enterprise applications, from TOSCA to docker and kubernetes[J]. Software: practice and experience, 2020, 50(9): 1793-1821.

[10] Kubernetes[EB/OL]. [2024-08-21]. https://kubernetes.io/docs/concepts/overview/.

[11] 李俊俊 , 董建刚 , 李坤 . 基于 Kubernetes 的集群节能策略研究 [J]. 计算机工程 , 2024, 50(9): 82-91.

【作者简介】

冯晶（1984—），女，山东济南人，硕士研究生，工程师，研究方向：云计算。

（收稿日期：2025-08-08 修回日期：2025-09-10）