Computer Science and Application 计算机科学与应用, 2024, 14(12), 132-141
Published Online December 2024 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2024.1412247

Hans汉斯

基于LSTM和遗传算法的容器资源智能调度策略研究

梁　拓[1]，王利众[1]，许　震[2]

[1]中央民族大学信息工程学院，北京
[2]北京青云科技集团股份有限公司项目服务部门，北京

收稿日期：2024年11月11日；录用日期：2024年12月10日；发布日期：2024年12月18日

摘　要

本研究旨在优化Kubernetes集群中容器化服务的资源调度，提高资源利用效率和系统性能。针对现有Kubernetes自动伸缩机制在应对复杂负载时的不足，提出一种基于LSTM和遗传算法的智能资源调度策略。首先利用LSTM模型预测未来的资源需求，以应对不同的负载波动；然后利用遗传算法优化资源配额，确保在高效利用资源的同时满足系统性能要求。实验在实际Kubernetes集群环境中进行，设置高负载、突发请求和低负载等负载场景，对比LSTM+遗传算法与Kubernetes默认HPA/VPA方案在响应时间、吞吐量和资源利用率方面的性能。结果表明，LSTM+遗传算法在大多数负载场景下均优于HPA/VPA策略，提高了平均资源利用率，降低了系统响应时间。

关键词

容器资源调度，LSTM，遗传算法，Kubernetes，自动伸缩

Research on Intelligent Scheduling Strategy of Container Resources Based on LSTM and Genetic Algorithm

Tuo Liang[1], Lizhong Wang[1], Zhen Xu[2]

[1]School of Information Engineering, Minzu University of China, Beijing
[2]Project Services Department, Beijing Qingyun Technology Group Co., Ltd., Beijing

Received: Nov. 11th, 2024; accepted: Dec. 10th, 2024; published: Dec. 18th, 2024

Abstract

This study aims to optimize the resource scheduling of containerized services in Kubernetes clus-

文章引用: 梁拓, 王利众, 许震. 基于 LSTM 和遗传算法的容器资源智能调度策略研究[J]. 计算机科学与应用, 2024, 14(12): 132-141. DOI: 10.12677/csa.2024.1412247

ters and improve resource utilization efficiency and system performance. In view of the shortcomings of the existing Kubernetes auto-scaling mechanism in dealing with complex loads, an intelligent resource scheduling strategy based on LSTM and genetic algorithm is proposed. First, the LSTM model is used to predict future resource requirements to cope with different load fluctuations; then the genetic algorithm is used to optimize resource quotas to ensure that system performance requirements are met while efficiently utilizing resources. The experiment is carried out in an actual Kubernetes cluster environment, setting load scenarios such as high load, burst requests, and low load, and comparing the performance of LSTM+ genetic algorithm and Kubernetes default HPA/VPA solution in terms of response time, throughput, and resource utilization. The results show that LSTM+ genetic algorithm outperforms HPA/VPA strategy in most load scenarios, improves average resource utilization, and reduces system response time.

Keywords

Container Resource Scheduling, LSTM, Genetic Algorithm, Kubernetes, Automatic Scaling

1. 引言

随着云计算和容器化技术的快速发展，Kubernetes 已经成为容器编排领域一个很重要的标准，其强大的调度功能和灵活的扩缩机制使开发者能够高效地管理和调度分布式应用。然而，面对高度动态的负载需求和复杂的应用场景，Kubernetes 现有的资源调度和自动扩缩容策略仍然存在一些局限性，例如基于规则的水平 Pod 自动扩缩器(HPA)和垂直 Pod 自动扩缩器(VPA)仅依赖于简单的资源使用阈值，难以及时响应复杂且快速变化的负载需求，可能导致资源浪费或短缺的情况发生。

为了更好地优化系统性能和资源利用效率，机器学习在云计算资源管理中的应用近年来受到了广泛关注，尤其是长短期记忆网络(LSTM)作为一种擅长处理时间序列数据的深度学习模型，由于能够有效捕捉时间序列中的长期依赖关系，在资源需求预测中得到了广泛的应用。通过 LSTM 模型，系统可以根据历史资源使用数据预测未来的负载需求，从而实现更精准的资源调度和自动扩缩策略。

然而，虽然 LSTM 在时间序列预测方面表现良好，但将其预测结果转化为更加高效的资源调度策略仍然需要进一步优化。遗传算法作为一种智能优化算法，具有很强的全局搜索能力，可以在复杂的优化问题中找到最优解。将 LSTM 预测与遗传算法相结合，可以进一步提高资源调度策略的智能化。遗传算法可以根据 LSTM 预测的未来资源需求，动态调整容器的资源配额，实现资源的最优配置。

因此，本研究的目标是提出一种基于 LSTM 和遗传算法的容器化服务性能优化与资源动态调度策略。该策略通过 LSTM 预测容器化服务的资源需求，并通过遗传算法优化资源调度和自动扩缩机制，旨在提高容器集群的资源利用效率和系统性能。我们将通过实验来验证该方案在不同负载场景下的有效性，并与 Kubernetes 现有的 HPA、VPA 方案进行对比分析，展现其在处理复杂负载的优越性。

2. 相关工作

2.1. 容器资源调度与管理

容器编排平台的核心功能之一就是高效的资源调度和管理。作为业界主流的容器编排平台，Kubernetes 提供了一系列的资源调度机制来满足不同场景的需求[1]。Kubernetes 调度器主要基于预定义

的规则和配置参数，以“适应性”和“亲和性”来决定容器的调度。适应性指调度器会根据资源需求和可用性，尝试将容器调度到最合适的节点上；亲和性指根据指定的节点选择策略，选择特定类型的节点进行容器部署。

然而，这种静态的调度方式存在明显的不足，当面对复杂、动态变化的负载时，Kubernetes 的调度机制往往难以实时响应应用需求，无法灵活调整资源分配。例如，默认的 Kubernetes 调度器无法根据应用的历史负载智能地分配资源，容易造成资源浪费或性能瓶颈。针对这一问题，研究者提出了多种改进方案，包括使用负载均衡算法、动态优先级策略等[2]。但这些改进仍然以规则驱动为主，缺乏对系统状态进行充分学习和实时预测的能力。

2.2. Kubernetes 自动伸缩机制及局限

Kubernetes 提供了两种自动扩缩机制，即水平 Pod 自动扩缩(HPA)和垂直 Pod 自动扩缩(VPA)，以应对应用负载的动态变化。

HPA 根据应用的 CPU 或内存利用率自动调整 Pod 副本数，以满足不同负载下系统的性能要求[3]。VPA 根据应用的实际资源使用情况动态调整容器的资源配额(如 CPU 和内存)，确保应用在资源不足或浪费时能够及时调整[4]。但 HPA 和 VPA 的调整策略主要由阈值驱动，需要手动配置各种阈值和策略参数。

这些现有的自动扩缩机制在实践中也存在许多局限性。首先，HPA 和 VPA 都依赖于当前的资源使用情况进行调整，而不是预测未来的资源需求。因此，当负载突然增加时，自动扩缩往往无法及时响应，从而可能导致性能下降或资源分配不足。其次，传统的自动扩缩机制在复杂场景下表现不佳。尤其在面临多维度资源竞争和突发流量时，HPA 和 VPA 很难同时兼顾各维度的资源优化需求[2]。

2.3. 机器学习在资源管理中的应用

随着机器学习技术的快速发展，许多研究开始将其应用于云计算资源管理，特别是在负载预测和资源调度领域[5]。机器学习模型可以通过学习历史数据自动发现系统资源使用的规律，帮助系统实现更准确的资源调度和优化。

在云计算环境中，常用的机器学习算法包括线性回归、决策树、支持向量机(SVM)和神经网络。它们应用于不同的资源管理任务，如 CPU 和内存使用率预测、负载平衡和任务调度[6]。与传统的规则驱动方法相比，机器学习算法可以通过数据训练实现复杂环境中的动态资源分配。例如，支持向量回归(SVR)用于预测云平台上的 CPU 和内存使用情况，以帮助优化资源调度。

近年来，深度学习技术，尤其是循环神经网络(RNN)及其变体 LSTM，由于其强大的处理时间序列数据的能力，已成为资源需求预测领域的研究热点。LSTM 模型能够有效捕捉时间序列数据中的长期依赖关系，适用于复杂的资源需求预测场景，因此基于 LSTM 的负荷预测模型逐渐成为动态资源调度研究的重要方向[7]。

2.4. LSTM 在时间序列预测中的应用

长短期记忆网络(LSTM)是循环神经网络(RNN)的一种特殊结构，旨在解决时间序列数据中的长期依赖问题[8]。在传统的 RNN 中，随着时间步长的增加，梯度消失或爆炸，使得网络难以捕捉长期依赖关系[7]。LSTM 通过引入记忆单元(Cell State)和门控机制(输入门、遗忘门、输出门)，可以有效地保留长期依赖信息，从而解决了 RNN 在处理长期依赖关系方面的局限性。

LSTM 广泛应用于各种时间序列预测任务[8]，包括股票价格预测、天气预报、交通流量预测等。LSTM 在系统性能优化和资源管理等领域也展现出了巨大的潜力。例如，LSTM 已被用于预测数据中心服务器的能耗、网络流量以及 CPU 和内存使用情况[9]。通过对历史资源使用数据进行建模，LSTM 可以预测未

来的资源需求，为系统资源调度提供决策支持。

2.5. 遗传算法及其在资源分配和优化问题中的应用

遗传算法(GA)是一种基于自然选择和遗传机制的全局优化算法，由 John Holland 于 20 世纪 70 年代首次提出[10]。它模拟生物进化过程，通过选择、交叉、变异等操作在解空间中搜索最优解，适用于解决复杂的优化问题。遗传算法在优化问题中具有很强的全局搜索能力，尤其适用于高维、非线性、多峰值的优化场景。

在资源分配和优化问题中，遗传算法通常用于解决在有限资源下如何最大化系统性能的问题[11]。它通过定义适应度函数(如资源利用率或响应时间)来评估不同资源分配方案的优劣，经过多轮进化操作逐渐逼近最优资源分配方案。特别是在容器化服务中，遗传算法可以根据负载需求动态调整 Pod 的 CPU 和内存配额，平衡系统性能和资源消耗，避免资源过度分配和分配不足。

3. 系统架构与方法

3.1. 总体架构设计

3.1.1. 系统组成

1. 资源监控模块：该模块负责实时监控 Kubernetes 集群中各个 Pod 的资源使用情况。它使用 Prometheus 和 cAdvisor 等工具收集 CPU、内存、网络带宽、磁盘 I/O 等关键指标的数据，并存储在时序数据库中。

2. LSTM 预测模型：通过对历史资源监控数据的学习，LSTM 模型可以预测未来的 CPU、内存需求，为后续的资源调度提供参考。

3. 遗传算法优化的资源调度器：根据 LSTM 模型的预测结果，遗传算法对容器的资源配额进行动态优化。遗传算法通过多次进化搜索，找到最优的资源分配方案，避免过度或不足分配。

4. 自动伸缩策略：根据遗传算法优化后的资源配额方案，自动执行 Kubernetes 的水平和垂直伸缩操作(HPA 和 VPA)，确保系统在不同负载条件下都能保持高效运行。

3.1.2. 工作流程

1. 数据收集：资源监控模块使用 Prometheus，从 Kubernetes 集群中获取资源使用数据。

2. 资源预测：收集到的历史数据通过 LSTM 预测模型进行处理，预测未来的资源需求。

3. 遗传算法优化：根据 LSTM 预测的未来负载，使用遗传算法对 Pod 的资源配额进行进一步优化，生成新的调度策略。

4. 调度与伸缩决策：根据优化后的调度策略，自动调整 Kubernetes 中的 HPA 和 VPA 参数，动态改变 Pod 的数量和资源配额，确保在不同时段和负载下保持高效的资源利用率。

3.2. 资源监控与数据收集

3.2.1. Kubernetes 集群资源监控工具

1. Prometheus：Prometheus 是一个开源的监控和报警工具，广泛用于 Kubernetes 集群中。它可以定时抓取来自 cAdvisor 等监控工具的资源使用数据，并将数据以时间序列的方式存储在内部数据库中。

2. cAdvisor：cAdvisor 是 Kubernetes 集成的容器级监控工具，能够监控容器的资源消耗情况，如 CPU、内存、网络和磁盘 I/O。

3.2.2. 数据收集

1. 数据指标选择：包括 CPU 使用率、内存占用、网络带宽、磁盘 I/O 等四类资源使用情况。

2. 数据采样频率：采集间隔设定为 30 秒，确保足够的时效性和准确性。

3. 数据存储：监控数据被存储在 Prometheus 的时序数据库中，作为模型训练的输入数据。

4. 基于 LSTM+遗传算法的智能资源调度策略

在本节中，LSTM 模型用于预测系统未来的资源需求，遗传算法负责根据预测结果动态优化资源分配与调度策略。通过结合两者的优势，可以实现资源利用效率的最大化和系统性能的动态调节。

4.1. LSTM 预测模型设计

LSTM 是一种专门处理时间序列数据的神经网络，适合预测容器化服务中负载变化的规律。为了准确预测容器的资源需求，需要对监控数据进行充分的预处理，并设计适当的 LSTM 模型结构。

4.1.1. 数据预处理

1. 数据清洗：由于监控数据可能存在缺失值、异常值等问题，所以首先要进行数据清洗。这里使用 Z-score 异常检测法检测异常点，公式为

$$Z = \frac{X - \mu}{\sigma} \tag{1}$$

其中，X 是数据集中某个数据点的值，μ 是数据集的均值，σ 是数据集的标准差，如果|Z| > 3，则该数据点可能为异常值。

然后使用线性插值法填补缺失数据，并替换明显的异常点，公式为：

$$y = y_1 + \frac{(x - x_1)(y_2 - y_1)}{x_2 - x_1} \tag{2}$$

其中，y 是需要替换的异常点计算后的替换值，(x_1, y_1)和(x_2, y_2)为异常点前后的两个数据点。

2. 归一化：LSTM 网络对数据的量级敏感，因此需要对所有的输入特征进行归一化处理。我们使用了 Min-Max 归一化方法，将所有特征映射到[0, 1]的范围内，以便加快模型的收敛速度。

归一化公式如公式(1)：

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{3}$$

其中，x 为原始特征值，$x_{\min}$ 和 $x_{\max}$ 分别为特征值的最小值和最大值。

3. 特征选择：根据分析结果，选取了对资源预测较为重要的特征，具体包括：CPU 使用率(%)，内存使用量(MB)，负载均衡请求速率(req/s)。

4.1.2. LSTM 模型结构

LSTM 是一种特殊的递归神经网络(RNN)，具有记忆单元和门控机制，能够有效捕捉时间序列中的长时间依赖关系。其核心结构包括遗忘门、输入门和输出门，控制信息在时间步之间的传递和更新。

本文中的 LSTM 模型的结构为：

1. 输入层：接收时间序列数据，输入维度为(batch_size, time_steps, features)，其中 time_steps 设为 10，代表过去 10 分钟的数据，features 包括 CPU 使用率、内存使用率。

2. 隐藏层：包含两层 LSTM 层，每层 64 个神经元，激活函数为 ReLU。

3. 输出层：全连接层将 LSTM 的输出转换为未来 5 分钟的资源需求预测，输出维度为 CPU 和内存使用率的预测值。

4.1.3. 训练数据集与模型参数设置

1. 数据集：从集群中采集了 7 天的资源使用数据，包括 CPU 使用率、内存使用率。数据集按 7:3 的比例划分为训练集和验证集。

2. 输入特征：CPU 使用率、内存使用率，时间步长为 10 分钟，即模型输入为过去 10 分钟的资源使用情况。

3. 输出：预测未来 5 分钟的 CPU 和内存使用率。

4. LSTM 模型参数：两层 LSTM，每层 64 个神经元，激活函数为 ReLU；优化器为 Adam；损失函数为均方误差(MSE)；批大小(Batch Size)为 64，训练轮数(Epochs)为 50。

4.1.4. 模型训练评估

在测试数据上使用均方误差(MSE)和平均绝对误差(MAE)作为评估指标。

$$\mathrm{MSE}=\frac{1}{n}\sum_{i=1}^{n}\left(y_i-\hat{y}_i\right)^2 \tag{4}$$

在公式(1)中，n 是测试样本的数量，y_i 和 $\hat{y}_i$ 为真实值和预测值。

$$\mathrm{MAE}=\frac{1}{n}\sum_{i=1}^{n}\left|y_i-\hat{y}_i\right| \tag{5}$$

在公式(1)中，n 是测试样本的数量，y_i 和 $\hat{y}_i$ 为真实值和预测值。

LSTM 模型在未来 5 分钟内对 CPU 和内存使用预测与每分钟内平均实际资源使用情况的对比如下表 1：

Table 1. LSTM predicted resource usage and actual usage
表 1. LSTM 预测资源使用率与实际使用率

时间(分钟)	实际 CPU 使用率(%)	预测 CPU 使用率(%)	实际内存使用率(%)	预测内存使用率(%)
1	71.83%	69.31%	67.98%	65.13%
2	73.61%	71.89%	67.98%	67.54%
3	75.94%	74.58%	70.98%	68.94%
4	78.32%	76.12%	73.21%	71.32%
5	80.67%	79.34%	74.87%	72.65%

LSTM 模型能够较为准确地预测未来的资源需求，CPU 和内存使用率预测误差均控制在 5%以内，符合实际调度系统的需求。

4.2. 遗传算法优化资源分配

遗传算法在本研究中用于优化 Kubernetes 集群中的资源分配，基于 LSTM 预测的未来资源需求对 Pod 的资源配额和数量进行调整。

4.2.1. 初始种群设置

遗传算法在本研究中用于优化 Kubernetes 集群中的资源分配，基于 LSTM 预测的未来资源需求对 Pod 的资源配额和数量进行调整。

1. 随机生成初始种群：基于 LSTM 预测未来几分钟的资源需求，包括未来的 CPU 和内存使用率，生成初始的 Pod 资源配额作为遗传算法的初始种群。个体示例：{CPU 配额 = [2.0, 1.0, 1.5]，内存配额 = [2048 MB, 512 MB, 1024 MB]}。

2. 种群规模：设定初始种群的规模为 50 个，每个个体表示一种资源分配策略，包括 CPU 和内存的配额。

4.2.2. 适应度函数

适应度函数用于评估每个个体(每个资源的分配策略)的优劣，目标是最大化资源利用率并最小化系统响应时间。

1. 目标函数：适应度函数的目标是最大化 CPU 和内存的利用率，同时最小化系统的响应时间。适应度函数表达式如下：

$$F=\alpha\times U_{\text{cpu}}+\beta\times U_{\text{mem}}-\gamma\times R \tag{6}$$

其中，U_{cpu} 是 CPU 利用率，U_{mem} 是内存利用率，R 是系统响应时间。α，β，γ 是权重系数，分别控制 CPU 利用率、内存利用率和响应时间在适应度中的权重，确保在优化过程中权衡资源利用率和响应时间。

2. 优化目标：最大化资源利用率并且最小化系统响应时间。

4.2.3. 遗传操作

遗传算法的核心包括选择、交叉和变异三种操作，通过这些操作逐代演化，优化资源调度策略。

1. 选择：据个体的适应度值选择优秀的个体进入下一代。这里使用轮盘赌选择法，过程如下：每个个体根据其适应度值分配选择概率，适应度高的个体有更大机会被选中；将种群中每个个体的适应度值进行归一化处理，以确保适应度值的和为 1；随机产生一个[0, 1]范围内的数字，选择对应的个体；该过程重复进行，直到生成新的种群。

2. 交叉：将两个个体(父代)的部分特征进行交换，产生新的后代。交叉概率设为 0.81，随机选择两个父代个体，将其 Pod 配额或数量部分进行交换。

3. 变异：对个体的某些参数(如 CPU 配额)进行随机微小调整，增加种群的多样性，防止早期收敛。变异概率设为 0.23，随机选择一个个体，并对其某些特征进行轻微改变。

4. 终止条件：遗传算法的终止条件通常是当种群中的最优个体在连续 20 代中无明显变化时，算法终止，即种群的最优解已经收敛。

4.2.4. 结果评估

在遗传算法优化过程中，平均资源利用率是评估种群中每个个体(每种资源分配策略)性能的关键指标之一。为了计算种群中的平均资源利用率，首先需要定义每个个体的资源利用率，并对种群中的所有个体求平均。

每个个体的资源利用率计算公式如下：

$$U_i=\alpha\times U_{\text{cpu},i}+\beta\times U_{\text{mem},i} \tag{7}$$

其中，$U_{\text{cpu},i}$ 是个体 i 的 CPU 利用率，计算为：

$$U_{\text{cpu},i}=\frac{U_{c1}}{U_{c2}} \tag{8}$$

其中，U_{c1} 是实际 CPU 使用量，U_{c2} 是分配的 CPU 配额。

$U_{\text{mem},i}$ 是个体 i 的内存利用率，计算为：

$$U_{\text{mem},i}=\frac{U_{m1}}{U_{m2}} \tag{9}$$

其中，U_{m1} 是实际内存使用量，U_{m2} 是分配的内存配额。

α，β 是 CPU 和内存利用率的权重系数，用于反映系统中 CPU 和内存对整体性能的重要性，通常 $\alpha + \beta = 1$。

由以上公式可以得出个体数量为 N 的种群的平均资源利用率公式为：

$$U_{\text{avg}} = \frac{1}{N}\sum_{i=1}^{N}\left(\alpha \times U_{\text{cpu},i} + \beta \times U_{\text{mem},i}\right) \tag{10}$$

经过 100 代的迭代，遗传算法逐渐收敛，初始种群中的平均资源利用率约为 58.74%，优化后的资源利用率提高至约为 73.12%。具体适应度结果如下表 2：

Table 2. Genetic algorithm optimization resource quota results
表 2. 遗传算法优化资源配额结果

迭代次数	平均适应度	最优适应度
10	0.6176	0.6457
20	0.6524	0.6819
50	0.7141	0.7528
100	0.7436	0.7809

5. 集群实验验证与结果分析

为了验证基于 LSTM 和遗传算法的智能资源调度策略在容器化服务中的有效性，本研究通过实际 Kubernetes 集群搭建实验环境，设计了一系列对比实验来评估其在不同负载场景下的性能表现。以下是实验的详细步骤和结果分析。

5.1. 实验环境与负载场景

实验环境基于 Kubernetes 集群，实验平台运行在 4 个节点的 Kubernetes 集群上，每个节点配备 4 核 CPU 和 16 GB 内存。该集群用于部署运行不同服务的容器，并进行负载管理。

5.2. 负载场景

1. 较高负载场景：模拟较高并发场景，每秒请求数(RPS)逐步上升到最大值，持续一段时间后逐步下降。使用 JMeter 生成流量负载，配置 200~1000 个并发用户，逐步增加 RPS，持续高峰测试 10 分钟，RPS 峰值达到 600 请求/秒。

2. 突发请求场景：模拟短时间内大量突发请求的情况，RPS 在几秒内急剧上升到高峰，然后快速下降恢复到正常水平。使用 JMeter 进行突发负载模拟，配置为短时间内生成 500~1500 并发用户，RPS 峰值达到 1000 请求/秒，持续突发 3 分钟。

3. 低负载场景：模拟较长时间的低负载场景，每秒请求数保持在较低水平。使用 JMeter 进行低负载测试，RPS 设置在 50~200 请求/秒，持续 20 分钟以测试系统在资源使用较少时的优化效果。

5.3. 对比实验设计

为了全面评估基于 LSTM 和遗传算法的智能资源调度方案的性能，本文设计了与 Kubernetes 默认的 HPA 和 VPA 方案的对比实验。

1. 实验对象

LSTM+遗传算法，基于本研究提出的调度策略，通过 LSTM 预测未来资源需求，并结合遗传算法进

一步优化资源配额。

Kubernetes HPA/VPA：使用 Kubernetes 的默认水平和垂直 Pod 自动伸缩策略，根据实时资源使用情况调整系统资源配额。

2. 评估指标

响应时间：测量系统在不同负载下的平均响应时间，单位为毫秒(ms)。

资源利用率：CPU 和内存的利用率(实际 CPU 或内存使用量/分配的 CPU 或内存资源)，以及平均资源利用率(设定 $\alpha=\beta=0.5$)，单位为百分比(%)，反映资源的分配与实际使用是否合理。

吞吐量：系统在给定时间内处理的请求数，单位为请求/秒(RPS)。

5.4. 实验结果与分析

以下表 3~5 是系统在不同负载场景下的实验数据，分别对比了基于 LSTM+遗传算法的调度方案与 Kubernetes 默认 HPA/VPA 方案的表现：

Table 3. Higher load scenario
表 3. 较高负载场景

调度方案	响应时间(ms)	CPU 利用率(%)	内存利用率(%)	平均资源利用率(%)	吞吐量(RPS)
LSTM+遗传算法	121.52	82.47	75.32	78.89	580.43
Kubernetes 默认	180.76	73.81	66.45	70.13	526.38

Table 4. Burst request scenario
表 4. 突发请求场景

调度方案	响应时间(ms)	CPU 利用率(%)	内存利用率(%)	平均资源利用率(%)	吞吐量(RPS)
LSTM+遗传算法	183.21	85.23	80.34	82.79	635.12
Kubernetes 默认	279.43	73.34	70.16	71.75	578.24

Table 5. Lower load scenario
表 5. 较低负载场景

调度方案	响应时间(ms)	CPU 利用率(%)	内存利用率(%)	平均资源利用率(%)	吞吐量(RPS)
LSTM+遗传算法	90.17	76.89	73.27	75.08	185.43
Kubernetes 默认	134.78	68.32	62.57	65.44	172.54

实验结果表明，基于 LSTM 和遗传算法的智能资源调度方案在不同负载场景下对比 Kubernetes 默认的 HPA/VPA 策略均表现出优势。LSTM 模型的预测功能使得系统能够提前识别高负载和突发请求，遗传算法则对资源配额进一步优化，使得调度策略在资源利用率、响应时间和吞吐量都优于 Kubernetes 默认的 HPA/VPA 策略。

6. 结论与未来工作

6.1. 研究总结

本研究提出的基于 LSTM 和遗传算法的智能资源调度策略，利用 LSTM 预测未来的资源需求，并结合遗传算法优化资源配额，提升容器化服务的性能和资源利用率。实验结果表明，该策略在不同负载场景下均优于 Kubernetes 默认的 HPA/VPA 调度机制，并且提升了系统响应速度和吞吐量。研究验证了该

智能资源调度策略在 Kubernetes 集群中的应用价值，为容器化服务的动态资源管理提供了一种新的解决方案。

6.2. 未来工作

未来可进一步优化 LSTM 模型和遗传算法的精度和效率，提高资源预测和调度的准确性。同时考虑引入强化学习等其他智能优化算法，以更好地应对复杂的负载模式和动态的资源需求，提高算法的适应性和鲁棒性，从而实现更高效、智能的资源调度策略。

参考文献

[1] Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J. (2016) Borg, Omega, and Kubernetes. *Communications of the ACM*, **59**, 50-57. https://doi.org/10.1145/2890784

[2] Xu, Z., Gong, Y., Zhou, Y., Bao, Q. and Qian, W. (2024). Enhancing Kubernetes Automated Scheduling with Deep Learning and Reinforcement Techniques for Large-Scale Cloud Computing Optimization. 9*th International Symposium on Advances in Electrical, Electronics, and Computer Engineering* (*ISAEECE* 2024), Volume 13291, 132916E. https://doi.org/10.1117/12.3034052

[3] Nguyen, T., Yeom, Y., Kim, T., Park, D. and Kim, S. (2020) Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration. *Sensors*, **20**, Article No. 4621. https://doi.org/10.3390/s20164621

[4] Niazi, M., Abbas, S., Soliman, A., Alyas, T., Asif, S. and Faiz, T. (2023) Vertical Pod Autoscaling in Kubernetes for Elastic Container Collaborative Framework. *Computers, Materials & Continua*, **74**, 591-606. https://doi.org/10.32604/cmc.2023.032474

[5] Bi, J., Li, S., Yuan, H. and Zhou, M. (2021) Integrated Deep Learning Method for Workload and Resource Prediction in Cloud Systems. *Neurocomputing*, **424**, 35-48. https://doi.org/10.1016/j.neucom.2020.11.011

[6] Huang, C., Wang, Y., Guan, C., Chen, H. and Jian, J. (2013) Applications of Machine Learning to Resource Management in Cloud Computing. *International Journal of Modeling and Optimization*, **2**, 148-152. https://doi.org/10.7763/ijmo.2013.v3.256

[7] Graves, A. (2012) Long Short-Term Memory. In: Graves, A., Ed., *Supervised Sequence Labelling with Recurrent Neural Networks*, Springer, 37-45. https://doi.org/10.1007/978-3-642-24797-2_4

[8] Lindemann, B., Müller, T., Vietz, H., Jazdi, N. and Weyrich, M. (2021) A Survey on Long Short-Term Memory Networks for Time Series Prediction. *Procedia CIRP*, **99**, 650-655. https://doi.org/10.1016/j.procir.2021.03.088

[9] Thonglek, K., Ichikawa, K., Takahashi, K., Iida, H. and Nakasan, C. (2019). Improving Resource Utilization in Data Centers Using an LSTM-Based Prediction Model. 2019 *IEEE International Conference on Cluster Computing* (*CLUSTER*), Albuquerque, 23-26 September 2019, 1-8. https://doi.org/10.1109/cluster.2019.8891022

[10] Holland, J.H. (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press.

[11] Gawanmeh, A., Parvin, S. and Alwadi, A. (2017) A Genetic Algorithmic Method for Scheduling Optimization in Cloud Computing Services. *Arabian Journal for Science and Engineering*, **43**, 6709-6718. https://doi.org/10.1007/s13369-017-2812-8