计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
2026,35(1):88−101 [doi: 10.15888/j.cnki.csa.010049] [CSTR: 32024.14.csa.010049]

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

基于改进元学习的深度强化学习集群调度优化①

贾晗铭, 王丽芳, 檀龙伟, 李沛聪, 黄昱帆

(中北大学 计算机科学与技术学院, 太原 030051)
通信作者: 王丽芳, E-mail: 727690392@qq.com

摘　要: 深度强化学习 (deep reinforcement learning, DRL) 在计算机集群调度任务中展现出了巨大潜力. 然而, 现有的基于深度强化学习的集群调度方法缺乏足够的泛化性, 导致其无法有效应对高度动态且变化频繁的集群环境. 为了应对这一挑战, 提出了一种改进元学习优化深度强化学习集群调度方法 MRLScheduler. 该方法的核心在于对元学习的两项改进: 首先, 引入了基于扩散模型的数据生成模块, 该模块在元学习的初始化阶段生成多样化的合成数据, 用于扩充和优化多任务数据集. 然后, 引入了基于扩散模型的经验回放模块, 该模块在元学习跨任务训练中利用历史任务数据生成合成经验, 用于对历史经验的重用. 最后, 将改进后的元学习集成到深度强化学习的集群调度算法中, 对处于高度动态且变化频繁的集群环境中的智能体进行策略微调, 从而改善智能体的泛化能力. 实验结果表明, MRLScheduler 优于其他基线算法, 有效地提升了深度强化学习集群调度算法的泛化能力.

关键词: 深度强化学习; 计算机集群调度; 泛化性; 扩散模型; 元学习

引用格式: 贾晗铭,王丽芳,檀龙伟,李沛聪,黄昱帆.基于改进元学习的深度强化学习集群调度优化.计算机系统应用,2026,35(1):88–101. http://www.c-s-a.org.cn/1003-3254/10049.html

Deep Reinforcement Learning Cluster Scheduling Optimization Based on Improved Meta-learning

JIA Han-Ming, WANG Li-Fang, TAN Long-Wei, LI Pei-Cong, HUANG Yu-Fan

(School of Computer Science and Technology, North University of China, Taiyuan 030051, China)

Abstract: Deep reinforcement learning (DRL) has shown significant promise in computer cluster scheduling tasks. However, existing DRL-based cluster scheduling methods often lack sufficient generalization, hindering their effectiveness in highly dynamic and frequently changing cluster environments. To address this challenge, this study proposes an improved meta-learning optimization method for deep reinforcement learning cluster scheduling, termed MRLScheduler. The essence of this methodology lies in two improvements to meta-learning: First, a data generation module based on diffusion models generates diverse synthetic data during the initialization phase of meta-learning to expand and optimize multi-task datasets. Second, an experience replay module based on diffusion models leverages historical task data to generate synthetic experiences during cross-task training in meta-learning, enabling the reuse of historical experiences. Finally, the improved meta-learning is integrated into the deep reinforcement learning cluster scheduling algorithm to fine-tune the strategies of agents in highly dynamic and frequently changing cluster environments, thus improving their generalization ability. The experimental results indicate that MRLScheduler outperforms other baseline algorithms, effectively enhancing the generalization ability of deep reinforcement learning-based cluster scheduling algorithms.

Key words: deep reinforcement learning (DRL); computer cluster scheduling; generalization; diffusion model; meta-learning

① 基金项目: 山西省科技创新计划 (20210222); 山西省重点研发计划 (202202010101008, 202102010101011); 山西省省筹资金资助回国留学人员科研项目 (2024-118)
收稿时间: 2025-06-09; 修改时间: 2025-07-07; 采用时间: 2025-08-15; csa 在线出版时间: 2025-11-26
CNKI 网络首发时间: 2025-11-27

随着大模型 (如 GPT-4、BERT 等) 的出现, 计算需求呈现出爆炸式增长. 大模型的训练和推理需要依赖庞大的分布式系统和异构计算资源, 如 GPU、TPU 及专用加速器[1]. 因此, 如何有效管理和调度这些计算资源, 以应对不断增长的计算需求和多样化的任务类型, 已成为亟待解决的关键问题[2]. 特别是不同硬件的异构性和任务需求的动态变化, 使得资源调度面临的挑战愈加复杂. 所以设计出适应不同场景的灵活、高效的任务调度策略, 不仅有助于提升系统性能、优化资源利用率, 也是应对复杂计算环境的关键[3].

深度强化学习 (deep reinforcement learning, DRL)[4] 作为一种结合深度学习[5]和强化学习优势的智能优化方法, 展现出了在资源调度问题中的应用潜力[6]. DRL 通过让智能体在与环境的交互中不断学习和优化行为, 实现了从经验中学习的能力, 特别适用于应对目标冲突和资源紧张等调度场景[7].

尽管 DRL 在应对目标冲突和资源紧张等调度场景中展现了巨大优势, 但其在高度动态且变化频繁的集群环境中, DRL 仍面临挑战[8]. 具体而言, 高度动态且变化频繁的集群环境表现为资源状态实时异构、任务负载非平稳分布, 且数据具有高方差特性. 因此, 增强 DRL 在高度动态且变化频繁的集群环境中的泛化性和适应性, 已成为当前研究的重点[9].

元学习[10]作为一种新兴的机器学习方法, 通过在多个不同任务上训练模型, 使其能够快速适应新任务和环境变化. 这种能力使元学习适合动态调度环境, 使得智能体在应对新环境时无需重新训练, 从而提升调度策略的泛化性和适应性[11].

然而, 尽管元学习在增强 DRL 智能体的泛化性和适应性方面展现出巨大潜力, 但将其实际部署于复杂集群环境时仍面临诸多挑战. 具体而言, 数据集的质量和数量问题, 以及对历史经验的高效利用成为制约其效能的关键因素, 这在一定程度上限制了元学习在复杂多变环境中的实际应用效果[12].

为了解决上述不足, 本文提出了一种改进元学习优化深度强化学习集群调度方法 MRLScheduler (meta-learning reinforcement learning scheduler). 方法通过引入基于扩散模型的数据生成模块和经验回放模块, 有效提升了元学习在跨任务学习中的数据质量和数量, 并提高了对历史经验的利用效率. 改进后的元学习用来对 DRL 智能体的调度决策进行精细的策略微调, 使 DRL 调度方法能够在复杂动态环境中表现出更强的泛化性和适应性.

本文的主要贡献包括 3 方面.

(1) 引入了基于扩散模型的数据生成模块, 该模块会在元学习的初始化阶段通过分析和整合不同任务的输入特征, 并且提取共性信息以及减少类内变化, 从而生成多样化的合成数据, 解决了元学习在集群调度领域中受数据质量和数量不足影响的问题.

(2) 为提高元学习对历史经验的利用效率, 方法引入了基于扩散模型的经验回放模块. 该模块利用历史任务数据生成合成经验, 使元学习在不断变化的环境中更高效地适应新任务, 同时优化决策质量. 通过这一机制, 使得智能体在面对复杂任务组合时, 元学习可以提供更高效的策略调整.

(3) 对 MRLScheduler 在多种轨迹数据集上与其他对比方法进行了全面的实验, 验证了 MRLScheduler 比之前的方法更适用于未见过的任务.

1 相关工作

1.1 扩散模型

近些年, 扩散模型[13]因其在数据样本生成和增强训练稳定性方面的卓越性能而备受关注. 其原因在于逐步添加和去除噪声的方式: 前向噪声添加和逆向去噪还原. 在前向加噪阶段, 模型向数据逐步添加噪声, 最终使数据达到接近纯噪声的状态; 在逆向生成阶段, 模型通过逐步去噪的逆向过程, 将数据还原成高质量的样本.

为优化生成过程中的噪声预测, 扩散模型在每个时间步长上使用去噪损失函数, 如式 (1) 所示:

$$L_{\text{denoise}} := \mathbb{E}_{k\sim U(1,K),x_0(\tau)\sim q,\epsilon\sim N(0,I)}[\| \epsilon-\epsilon_\theta(x_k(\tau),y(\tau),k)\|^2] \tag{1}$$

该损失函数通过预测噪声并将其最小化来优化生成过程, 其中由深度神经网络参数化的 ϵ_θ, 被训练并用来预测添加到数据集样本 $x_0(\tau)$ 中的噪声 $\epsilon\sim N(0, I)$, 以产生 $x_k(\tau)$.

在去噪过程中, 它描述了如何在逆向生成过程中逐步去除噪声, 从而还原出清晰的合成数据样本.

$$x_{k-1}(\tau)\leftarrow\frac{1}{\sqrt{\alpha_k}}\left(x_k(\tau)-\frac{\beta_k}{\sqrt{1-\bar{\alpha}_k}}\epsilon_\theta(x_k(\tau),y(\tau),k)\right)+\sqrt{\beta_k}\sigma \tag{2}$$

其中, 噪声 $\epsilon_\theta(x_k(\tau), y(\tau), k)$ 被深度神经网络参数化, 并且通过训练来预测当前时间步 k 的噪声. 模型利用预测的噪声值逐步从带噪声的样本 $x_k(\tau)$ 中去除噪声, 最终恢复出高质量的生成样本 $x_{k-1}(\tau)$.

因此, 扩散模型在生成模型领域中具有广阔的应用前景, 为高质量数据生成提供了可靠的技术支持[14].

1.2 元学习

元学习是一种通过跨任务经验来提升模型在新任务上学习效率的框架[15]. 其核心目标是在多个任务的学习中积累经验, 使模型能够快速适应新任务.

在元学习的研究中[16], 传统单一初始化方法 MAML 以其简单高效的特点受到广泛关注. 然而, 单一初始化方法在面对复杂任务时存在局限, 难以找到适用于多样化任务的最优参数. 因此, 研究人员提出了多初始化方法, 以更好地应对任务的多样化需求[17].

XB-MAML[18]是一种多初始化方法, 通过学习可扩展的基础参数, 使这些初始化可以线性组合形成更适合给定任务的初始化点. 与 MUSML 和 TSA-MAML 等[19]方法的预定义初始化数量不同, XB-MAML 能够根据任务分布动态扩展初始化的数量, 更好地覆盖多样化的任务分布.

如图 1 所示, XB-MAML 凭借动态扩展和组合能力, 使其能够更好地帮助 DRL 集群调度方法去适应新任务, 从而达到提高泛化性的目的. 然而, XB-MAML 在动态环境中的实际应用效果, 会受到其数据集的质量、数量以及能否高效利用历史经验的影响.

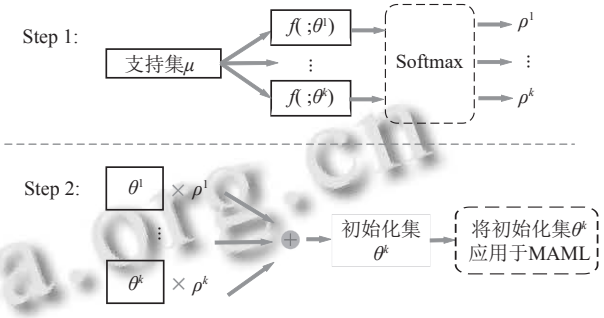


图 1 XB-MAML 的任务特征获取流程

2 方法

针对深度强化学习集群调度方法在应对高度动态且变化频繁的集群环境时泛化能力不足的问题, 本文提出了一种改进元学习优化深度强化学习集群调度的方法 MRLScheduler, 图 2 展示了 MRLScheduler 的整体框架.

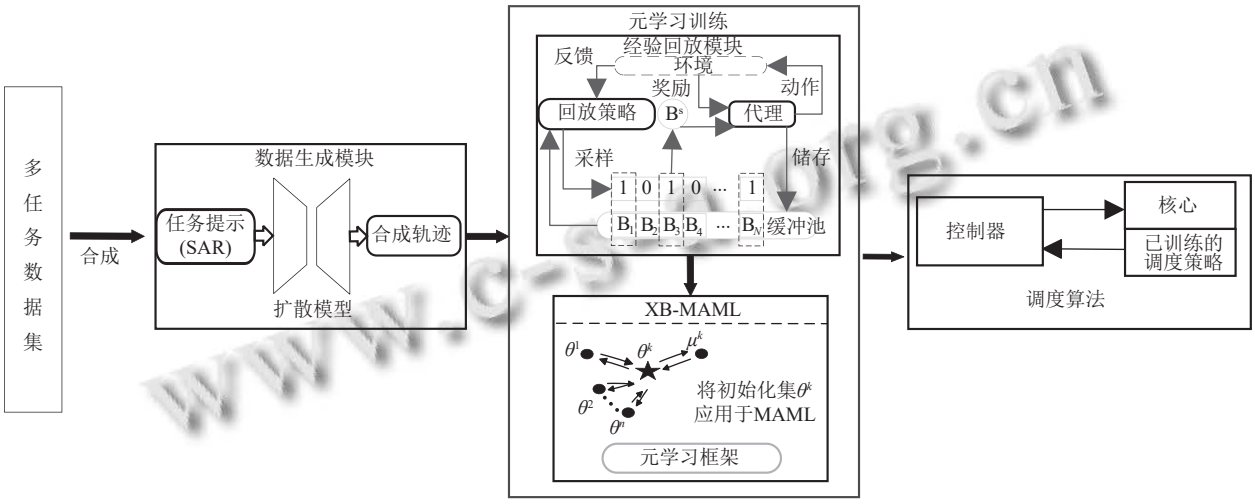


图 2 整体框架图

首先, 数据生成模块会对多任务数据集进行扩充和优化, 生成一个更加丰富和多样化的训练数据集. 然后, 优化后的数据集被用于改进元学习的训练. 在训练阶段中, 经验回放模块通过已学习的历史经验以及合成经验创建经验缓冲池以供元学习对历史经验进行重用. 最后, 在调度模块的训练中, 元学习会针对当前环境以及任务微调 DRL 智能体做出的调度决策, 从而提高 DRL 智能体在动态环境中的泛化能力.

2.1 改进的元学习

为克服元学习在数据样本及历史经验利用方面的局限性, 本文通过引入基于扩散模型的数据生成模块和经验回放模块对元学习进行改进, 充分利用扩散模

型在数据分布模拟和生成方面的优势, 旨在从数据层面优化元学习的训练过程, 提升其性能.

2.1.1 基于扩散模型的数据生成模块

为提高元学习对调度策略微调的精确性, 本文使用基于扩散模型的数据生成模块, 用于增强原始数据集. 数据生成模块的体系结构如图 3 所示. 模块主要包括: GPT2 Transformer 组件、多层感知器 (MLP) 和层归一化 (Layer Norm).

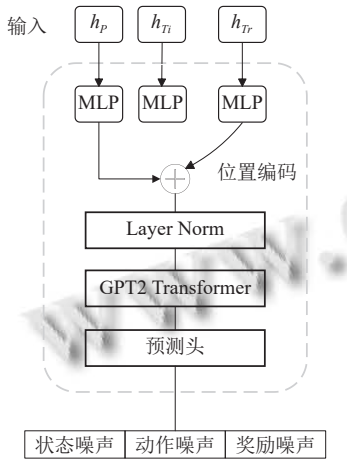


图 3 数据生成模块的体系结构

如图 3 所示, 模块通过独立的 MLP 模块将不同的原始输入 x 嵌入到相同维度 d 中的综合表示 h. 其中, 不同类型的原始输入如提示(h_P)、扩散时间步长(h_{Ti})以及转移数据(h^s_{Tr})的输入分别通过 $f_P(x^{\text{prompt}})$, $f_{Ti}(x^{\text{timestep}})$ 以及 $f_{Tr}(x^{\text{transitions}})$ 进行嵌入. 然后, 这些嵌入的结果被组合为扩散模型的输入令牌 h^s_{tokens}, 并经过以下预处理:

$$h^s_{\text{tokens}} = LN(h_{Ti} \times [h_P, h_{Ti}, h^s_{Tr}] + E^{\text{pos}}) \tag{3}$$

其中, E^{pos} 为位置嵌入, LN 为稳定训练的层归一化. GPT2 是一个仅解码的 Transformer, 它包含自注意力机制来捕获输入序列中不同位置之间的依赖关系. 它输出更新后的表示为:

$$h^s_{\text{out}} = Transformer(h^s_{\text{tokens}}) \tag{4}$$

其中, h^s_{tokens} 是被组合为扩散模型的输入令牌, h^s_{out} 为由 Transformer 处理后的输出. 最后, 给定输出表示, 使用一个由全连接层组成的预测头来随机预测扩散时间步长 k 处相应的噪声.

为了更清晰地展示数据生成模块, 图 4 展示了数据生成模块从多任务数据集中生成状态-动作-奖励 (state-action-reward, SAR) 序列的具体流程.

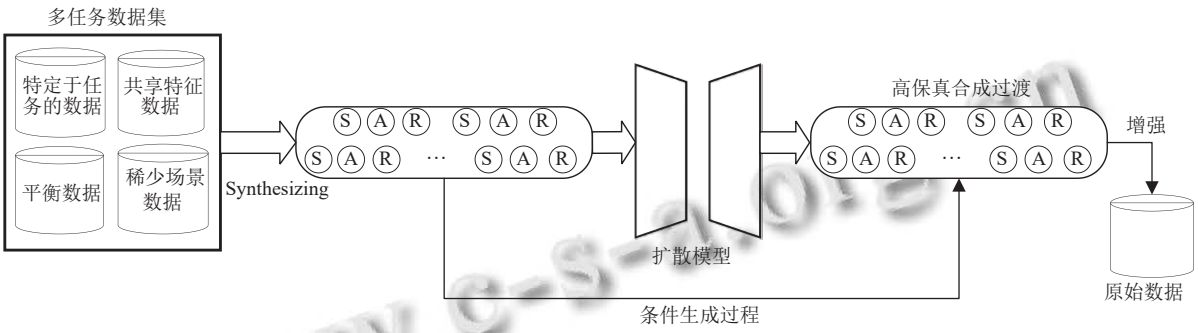


图 4 模块从多任务数据集中合成状态-动作-奖励序列

如图 4 所示, 数据生成模块会从多任务数据集中合成 SAR 序列. 这些合成的 SAR 序列被输入到扩散模型中进行处理. 扩散模型通过拟合这些 SAR 序列的数据分布, 生成新的数据样本, 既保留了任务相关的特性, 又通过增加数据的多样性和平衡性来扩展训练数据的覆盖范围. 生成的合成数据样本会被反馈回原始数据集 D 中, 以提升元学习的训练效果, 解决数据集不均衡和数据质量参差不齐的问题.

2.1.2 基于扩散模型的经验回放模块

为提高元学习对历史经验的利用效率, 本文引入了基于扩散模型的经验回放模块, 旨在通过扩展基础参数和优化策略, 以支持当前任务的学习, 从而提高其在复杂任务组合中的适应能力和效率.

经验回放模块利用扩散模型来模拟和扩展过去的训练经验, 这些生成经验不仅扩大了已经训练的经验规模, 还改善了任务经验集中特征不足的问题, 帮助模型更好地掌握通用任务表示. 通过这种方式, 模型可以更有效地记忆和再利用历史任务的知识.

如图 5 所示, 经验回放模块首先处理回合数据, 通过扩散模型的前向加噪过程, 逐步对历史任务中的数

据添加噪声. 从现有经验中进行上采样以生成更多的生成经验, 然后这些上采样后的合成经验被用来训练元学习. 这一过程不仅模拟了数据的自然退化过程, 还为后续生成合成经验提供了基础. 这一过程可以用式 (5) 表示:

$$x_t = \alpha_t x_0 + \sigma_t \epsilon \tag{5}$$

其中, ϵ 是加噪的高斯噪声, α_t和σ_t为时间步长 t 对应的噪声强度参数. 随着时间 t 的推移, 数据逐渐被噪声污染.

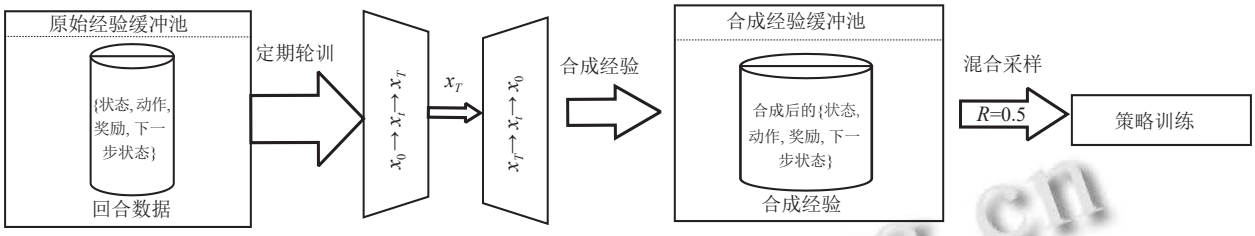


图 5 经验回放模型

在逆向去噪过程中, 扩散模型利用学到的去噪网络 D_θ 来逐步恢复原始数据. 对于每一步 t, 去噪网络根据当前状态 x_t 预测去噪后的数据为:

$$x_{t-1} = D_\theta(x_t, t) \tag{6}$$

这个生成过程不仅保留了历史任务中的关键特征, 还能有效地增加经验的多样性, 帮助模型更好地适应未知的任务.

经验回放模块会将生成的合成经验与历史经验一起放入创建的经验缓冲池中. 之后元学习在训练过程中会从经验缓冲池中随机抽取经验样本进行训练. 由于合成经验样本的加入, 元学习能够更好地捕捉和学习到任务之间的潜在关联.

2.2 理论可行性分析

MRLScheduler 的理论可行性可从收敛性和时间复杂度两方面展开分析.

收敛性分析聚焦于元学习在扩散模型生成数据支持下的策略收敛能力. 其核心前提是扩散模型生成的合成 SAR 序列与真实数据具有近似一致性, 即两者的分布误差 ε 可忽略 (实验中余弦调度的最低 FID 值验证了这一前提). 同时, 元学习的损失函数满足 Lipschitz 连续性, 即存在常数 $L>0$, 对任意参数θ_1, θ_2, 有$\| L(\theta_1) - L(\theta_2) \| \leqslant L \| \theta_1 - \theta_2 \|$, 以确保梯度更新的稳定性. 元学习的核心目标是最小化跨任务损失期望:

$$J(\theta) = \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[L_{\mathcal{T}}(\theta_{\mathcal{T}}^*) \right] \tag{7}$$

其中, $\theta_{\mathcal{T}}^*$为任务$\mathcal{T}$的最优参数, 由于引入扩散模型生成的合成数据, 目标函数扩展为真实数据损失与合成数据损失的加权和:

$$J(\theta) = \alpha J_{\text{real}}(\theta) + (1-\alpha) J_{\text{synth}}(\theta) \tag{8}$$

其中, $\alpha \in [0,1]$控制合成数据的权重. 元学习的参数更新遵循梯度下降规则:

$$\theta_{t+1}^{(m)} = \theta_t^{(m)} - \beta \nabla L_t^{(m)} \tag{9}$$

其中, β为学习率, $L_t^{(m)}$为第 m 个初始化参数在 t 步的损失. 基于梯度下降收敛理论, 参数序列$\{\theta_t\}$将收敛至最优解$\theta^\star$, 即:

$$\lim_{t \to \infty} \left\| \theta_t - \theta^\star \right\| = 0 \tag{10}$$

综上, 当扩散模型生成数据的分布误差足够小时, 改进元学习的策略参数更新可收敛至最优调度策略, 且收敛速度与扩散模型的生成质量正相关.

时间复杂度主要由扩散模型与元学习的操作成本共同决定. 扩散模型的核心操作为前向加噪与逆向去噪 (K 步), 每步涉及 GPT2 Transformer 的自注意力计算, 复杂度为 $O(d^2)$, d 为嵌入维度. 设样本量为 N, 因此时间复杂度为 $O(K \cdot N \cdot d^2)$, 数据生成模块与经验回放模块均涉及此操作, 故扩散部分为主要计算瓶颈. 元学习的跨任务训练包含 B 个任务批次的参数微调, 每批次涉及 T 次元更新, 梯度计算复杂度 $O(d)$, 总复杂度为 $O(B \cdot T \cdot d)$. 因此, 算法的计算瓶颈主要存在于扩散模型的高维嵌入变换过程. 但实验表明其显著提升泛化性并且通过优化扩散步数 K 与嵌入维度 d 的配置 (如 K=800, d=128), 在保证生成质量的同时有效控制计算开销.

2.3 改进元学习对集群调度的优化

鉴于深度强化学习的调度算法在动态环境中泛化能力不足, MRLScheduler 集成了改进的元学习来提高算法的泛化能力.

在集成改进元学习的调度优化框架中, DRL 用于初始的策略训练和优化, 提供基础调度策略. 而改进元

学习则利用 DRL 提供的策略框架, 进一步对不同任务进行快速适应和策略微调. 通过这种协同作用, 调度算法能够在面对动态集群任务时更快、更高效地调整调度策略.

图 6 展示了该框架的主要组件及其交互方式.

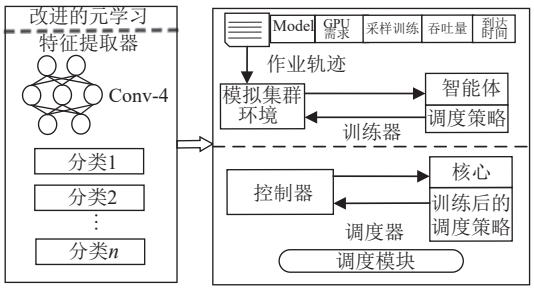


图 6 集成元学习的调度优化框架

特征提取器从数据中提取关键信息, 这些信息被用来组成初始化参数集$\{\theta^{(m)}\}_{m=1}^{M}$, M 为初始化参数的数量. 每个初始化参数$\theta^{(m)}$通过支持样本进行优化, 产生对应的损失值$\{\mathcal{L}_i^{(m)}\}_{m=1}^{M}$(其中 i 表示任务的索引, m 为对应的初始化编号), 从而确保在不同任务环境下获得适应性较强的初始参数$\theta^{(m)}$. 这些参数信息随后被传递给元学习模块.

改进元学习利用这些损失值和初始化参数, 通过 Softmax 机制对每个初始化参数$\theta^{(m)}$的损失值进行加权, 生成的系数$\{\sigma_i^{(m)}\}_{m=1}^{M}$用于线性组合各初始化参数, 形成更新后的参数$\theta_i^{\star}$. 通过这种方式, 改进元学习能够动态调整调度策略的权重, 以适应不同的任务需求. 其计算公式如下:

$$\theta_i^{\star}=\sum_{m=1}^{M}\sigma_i^{(m)}\theta^{(m)} \tag{11}$$

其中, $\theta^{(m)}$表示第 m 次初始化的参数, $\sigma_i^{(m)}$是通过 Softmax 计算得到的权重系数. 同时, 为了避免初始化参数过度趋同, 改进元学习引入了正则化损失$\mathcal{L}_{\text{reg}}$, 确保参数之间的正交性, 保持多样性以增强调度策略的泛化能力. 其损失函数的形式为:

$$\mathcal{L}_{\text{total},i}^{(m)}=\mathcal{L}(Q_i;\phi_i^{\star})+\mathcal{L}_{\text{reg}}^{(m)} \tag{12}$$

其中, $\mathcal{L}(Q_i;\phi_i^{\star})$为查询样本的损失, $\mathcal{L}_{\text{reg}}^{(m)}$是用于保证初始化参数正交性的正则化项. 改进元学习通过元更新机制在生成更新的初始化参数$\theta_i^{\star}$之后, 使用梯度下降法优化每个初始化参数$\theta^{(m)}$, 具体公式如下:

$$\theta^{(m)}\leftarrow\theta^{(m)}-\frac{\beta}{B}\sum_{i=1}^{B}\nabla_{\theta^{(m)}}\mathcal{L}_{\text{total},i}^{(m)} \tag{13}$$

其中, β 表示学习率, B 是任务的批次大小. 元更新通过逐步优化每个初始化参数, 使其能够在更广泛的任务分布中保持良好的适应性, 在实际的集群环境中不断提高调度效率.

在本研究中, 模型训练分 3 个阶段, 首先是元学习预训练阶段, 借助扩散模型数据生成模块, 对多数据集进行深度剖析, 提取作业的处理器请求数、运行时间等关键特征, 运用前向加噪与逆向去噪过程生成大量多样化的合成数据. 紧接着, 依托元学习开展跨任务训练, 在支持集任务上, 快速微调初始化参数$\theta^{(m)}$, 结合正则化损失$\mathcal{L}_{\text{reg}}$维持参数的多样性, 确保模型能灵活适应不同任务. 进入经验混合训练阶段, 经验回放模块按 3:7 比例混合合成经验与真实经验, 智能体基于 PPO 算法以 256 批次大小更新策略网络. 最后是动态环境微调阶段, 在目标集群环境中, 元学习通过 Softmax 加权机制动态生成策略, 直至平均有界延迟率波动<5% 终止训练.

3 实验

为了验证本文提出的改进元学习对深度强化学习的调度优化方法 (MRLScheduler) 的有效性, 设计并进行了多项实验. 这些实验针对不同的工作负载、优化目标, 以及算法在动态环境中的适应能力和稳定性进行评估, 分析算法在实际调度场景中的性能表现[20].

3.1 实验设置

MRLScheduler 的训练在个人工作站及模拟集群环境中进行, 硬件配置为 NVIDIA GeForce RTX 3080 Ti、12 核 Xeon(R) Silver 4214R CPU 及 90 GB 内存, 可通过软件模拟多个异构计算节点的集群环境 (支持 CPU/TPU/GPU 混合资源建模). 软件基于 PyTorch 2.0 框架开发, 集成 Diffusers 库实现扩散模型组件, 并通过自定义的 OpenAI Gym 环境模拟集群动态负载. 数据预处理采用 Pandas/NumPy 库, 训练过程利用多线程并行计算优化数据加载效率.

为了全面评估 MRLScheduler 在不同特征工作负载下的性能表现, 实验选择了多种具有代表性和多样化特征的作业跟踪数据集, 并基于综合性评价指标进行分析.

(1) 实验数据集

SDSC-SP2: 包含 128 个作业, 平均到达间隔为 1 055 s, 平均运行时间为 6 687 s, 平均请求处理器数为

11. 该数据集来源于 San Diego Supercomputer Center 的 SP2 系统, 代表了典型的科学计算工作负载, 具有较长的运行时间和较大的处理器需求.

HPC2N: 包含 240 个作业, 平均到达间隔为 538 s, 平均运行时间为 17 024 s, 平均请求处理器数为 6. 此数据集来源于 High Performance Computing Center North, 反映了学术研究中的高性能计算任务, 具有较高的资源需求和较长的作业运行时间.

PIK-IPLEX-2009: 包含 2 560 个作业, 平均到达间隔为 140 s, 平均运行时间为 30 889 s, 平均请求处理器数为 12. 该数据集来自 Potsdam Institute for Climate Impact Research (PIK) 的 IPLEX 系统, 主要用于气候模拟和环境研究, 作业数量多且复杂.

ANL Intrepid: 包含 163 840 个作业, 平均到达间隔为 301 s, 平均运行时间为 5 176 s, 平均请求处理器数为 5 063. 此数据集来源于 Argonne National Laboratory 的 Intrepid 系统, 是一个大规模并行计算环境, 主要用于大型科学计算和模拟, 数据集规模庞大, 作业数量众多.

Lublin-1 和 Lublin-2: 这两个数据集为合成工作负载, 分别包含 256 个作业, 具有不同的平均到达间隔和运行时间. Lublin 数据集由 Lublin 大学研究团队生成, 旨在模拟实际高性能计算环境中的工作负载, 通过调整参数生成不同特性的作业, 用于测试调度算法的适应性和泛化能力.

Google Cluster Data V2: 数据集来源于 Google 数据中心的真实作业跟踪, 记录了连续 29 天内数十万级任务的提交时间、CPU/内存请求与实际使用量、执行时长等关键信息. 与 HPC 场景下以长作业为主的大规模并行负载不同, Google Cluster Data V2 体现了云计算环境中高并发、小粒度、资源需求多样且负载波动频繁的典型特征, 特别适用于评估调度算法在任务到达速率骤变、资源碎片控制和快速响应能力方面的表现.

(2) 评价指标

实验使用了作业执行效率这一综合性指标, 涵盖了系统资源利用率和平均有界延迟率这两个维度.

系统资源利用率 (resource utilization, RU): 衡量计算资源的使用效率, 通过实际使用的处理器时间与可用处理器时间的比值进行评估. 在评估调度算法的性能时, 较高的资源利用率意味着算法能够有效地管理和分配计算资源, 减少资源浪费, 提高系统的整体效率.

平均有界延迟率 (average bounded delay ratio, ABDR): 衡量作业的执行效率, 通过计算作业的周转时间 (即作业从提交到完成的全部时间, 包括等待时间和执行时间) 与作业执行时间的比值来评估调度算法的效果. 这个比值反映了作业在系统中因调度而产生的额外延迟. 较低的平均有界延迟率意味着作业在系统中受到的延迟较少, 表明调度算法能够有效地减少等待时间, 保持作业的执行效率.

3.2 MRLScheduler 实验评估

本节通过对比 MRLScheduler 与传统深度强化学习算法 RLScheduler 在训练过程中的收敛速度和调度效果, 深入分析改进元学习的性能. 此外, 还将 MRLScheduler 与 RLScheduler 以及现有启发式调度算法 (FCFS、SJF[21]、WFP3[22]、UNICEP (简写为 UNI)[23]、F1[24]) 进行对比, 评估其在不同作业负载下的调度表现.

3.2.1 改进元学习的性能评估

在实验中, 选用了不同的作业跟踪数据集, 分别是 Lublin-1、Lublin-2 和 ANL Intrepid. 这些数据集具有不同的作业负载特征, 为调度算法提供了多样化的测试环境. 图 7 展示了 MRLScheduler 和 RLScheduler 在这 3 个数据集上的训练曲线对比.

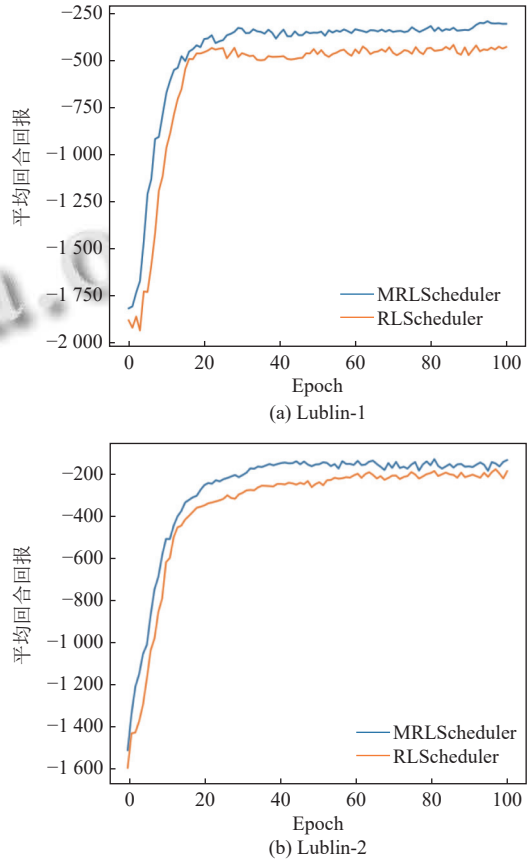
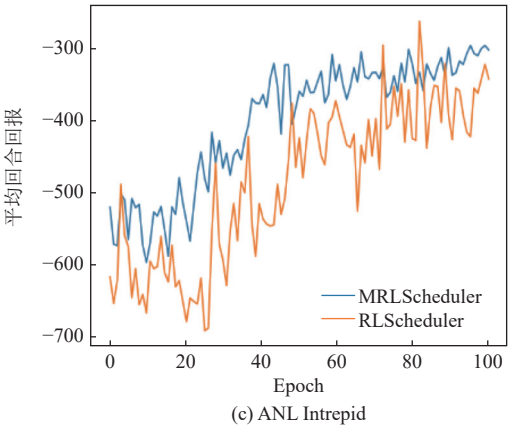


图 7 MRLScheduler 与 RLScheduler 的训练曲线对比



(c) ANL Intrepid

图 7 MRLScheduler 与 RLScheduler 的训练曲线对比 (续)

通过图 7 的 3 幅训练曲线图可以看出, MRLScheduler 在训练的早期阶段具有明显更快的收敛速度, 尤其是在 Lublin-1 和 Lublin-2 数据集上, 其训练曲线迅速趋于稳定. 这表明改进的元学习能够更迅速地适应环境变化, 并找到最优策略. 此外, 在 ANL Intrepid 数据集上, MRLScheduler 在面对复杂的工作负载时, 也展现了较高的泛化能力和快速调整的性能.

相比之下, RLScheduler 的训练曲线则表现出较大的波动性, 尤其是在面对 ANL Intrepid 时. 这是由于传统深度强化学习方法需要更长的时间来探索和优化策略, 且在处理复杂任务时, 需要频繁调整模型参数.

3.2.2 不同作业跟踪下的调度算法性能

为了进一步验证 MRLScheduler 在实际作业调度中的优势, 本文比较了其与 RLScheduler 以及传统启发式调度算法在不同随机作业序列下的调度效果.

实验从 SDSC-SP2、HPC2N、PIK-IPLEX-2009 和 ANL Intrepid 轨迹数据集中随机抽取 10 个作业序列, 在此调度情况下, 测试几种任务调度方法的调度效果, 实验使用了平均值来表示调度策略的效果. 图 8 展示了 MRLScheduler、RLScheduler 与其他传统启发式调度算法在处理不同随机作业序列下的性能表现.

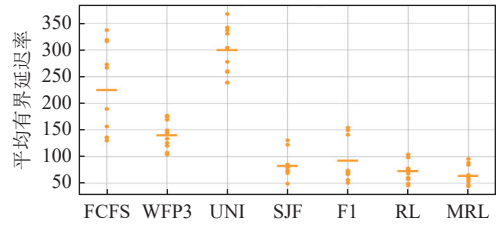


图 8 随机作业序列下不同算法的性能比较

如图 8 所示, 在随机作业序列下, MRLScheduler 的结果点集中在性能较好的区域, 说明其在大多数情况下都能保持较低的平均有界延迟率. 相比之下, RLScheduler 和其他传统算法的结果点分布更为分散, 尤其是在作业减速率较高的区域, 显示出更大的波动性和不稳定性. 这表明其他算法在不同任务序列中的性能不稳定, 难以有效应对作业序列的随机性.

通过以上对比分析, 可以发现 MRLScheduler 在各种作业跟踪数据上均能快速收敛, 展现出很好的训练效率和稳定性. 在随机作业序列下, 它不仅保持了较低的平均有界延迟率, 而且在不同任务序列中表现一致, 体现了良好的鲁棒性和适应性.

3.3 调度算法的综合性能

为了评估 MRLScheduler 在资源分配与作业调度上的综合表现, 实验选取多个典型数据集 (Lublin-1、Lublin-2、SDSC-SP2 和 HPC2N), 涵盖不同特性的任务负载. 并且设计了不启用回填和启用回填两种调度模式, 分别模拟资源负载较重和动态负载优化的情境.

在不启用回填模式下, 系统严格按照任务的提交顺序来调度资源, 若当前任务无法获得所需资源, 则继续等待. 此情境能够更真实地反映不同算法在高负载条件下的表现, 适合于资源紧张的环境. 而在启用回填模式中, 系统在排队的前序任务暂时无法获得资源时, 会优先调度符合资源要求的后续任务, 从而最大化资源利用率并减少任务等待时间. 这种动态资源管理方式适用于高效资源配置环境, 能够测试各算法在调度性能上的最大潜力.

3.3.1 资源利用率对比

实验通过图 9 和图 10 分别展示了在不启用回填和启用回填模式下, 不同调度算法在 4 个数据集上的资源利用率表现. 实验数据基于多次实验的汇总结果, 主要以中位数为分析依据.

从图 9 可以看出, 在不启用回填模式下, MRLScheduler 在 4 个数据集中整体资源利用率表现出色. 具体来说, 在 SDSC-SP2 中, MRLScheduler 的资源利用率可以达到 0.69, 超越了其他调度算法, 取得了最优效果, 展现了其在高负载条件下有效提升资源利用率的能力. 在类似情境下, MRLScheduler 在 Lublin-2 中的资源分配效果也表现优异, 资源利用率聚集在 0.57 左右, 超过其他算法. 尽管 SJF 和 RL 的资源利用率接近于 0.57, 但都略低于 MRLScheduler, 而 FCFS 的资源利用率最低, 仅在 0.40 左右, 这进一步突显了 MRLScheduler 在资源有限条件下的优势.

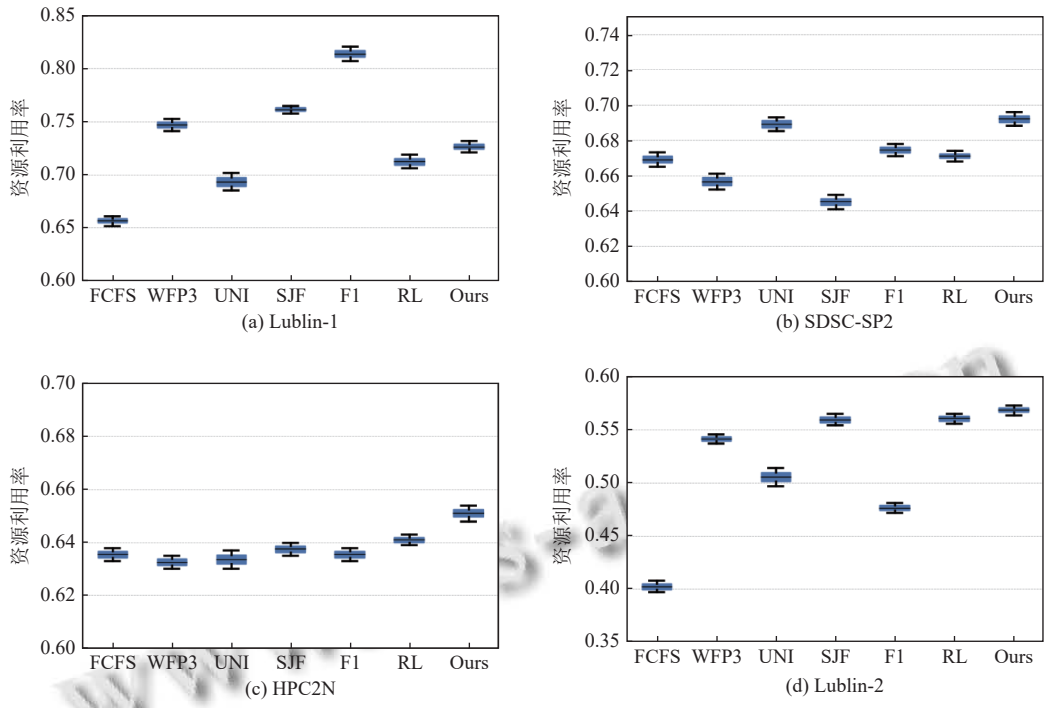


图 9 不同作业跟踪下各算法在不启用回填时的资源利用率对比

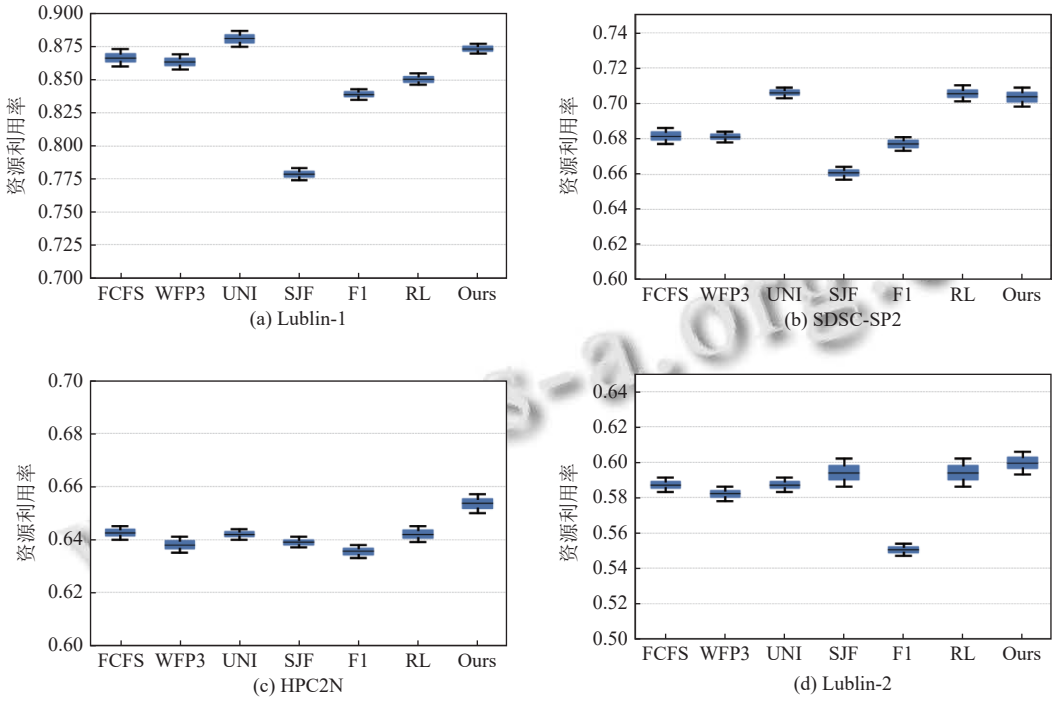


图 10 不同作业跟踪下各算法在启用回填时的资源利用率对比

在 HPC2N 中, MRLScheduler 表现出良好的竞争力和稳定的资源分配能力. 此外, 在 Lublin-1 中, 虽然 MRLScheduler 的资源利用率相较其他数据集表现并不突出, 但依然体现了稳定的资源分配能力. 相比之下, F1 算法在该数据集中取得了较高的资源利用率, 但其表现波动较大, 缺乏一致性.

如图 10, MRLScheduler 在多个数据集上表现出显著的资源利用率优势. 尤其在 Lublin-2 中, MRLScheduler 的资源利用率集中在 0.607 附近, 超过了其他调度算法, 例如 RLScheduler 的 0.59 和 SJF 的 0.58.

同样的, 在 SDSC-SP2 中, MRLScheduler 的资源利用率约为 0.704, 仅次于 RLScheduler 的 0.707, 进一步展示了在启用回填模式下的高效性和稳定性. 而在 Lublin-1 中, MRLScheduler 的资源利用率分布在 0.874 左右, 与表现最佳的 UNI (0.883) 接近, 反映出其在动态资源管理中的优异表现.

综上所述, 实验数据表明, 无论是在不启用回填还是启用回填条件下, MRLScheduler 均表现出较为突出的资源利用率优势, 使其能够在不同负载环境中实现稳定且高效的资源分配. 在大多数数据集中, MRLScheduler 相比传统调度算法在资源紧张和动态资源管理两种环境中均展现出高度适应性和资源优化能力, 有效提升了系统的资源利用效率.

3.3.2 平均有界延迟率对比

实验通过图 11 和图 12 分别展示了在不启用回填和启用回填模式下, 不同调度算法在 4 个数据集上的平均有界延迟率表现.

从图 11 可以看出, MRLScheduler 在大多数数据集上的平均有界延迟率明显低于其他调度算法, 展现出了较好的调度性能. 具体来说, MRLScheduler 在 Lublin-1 中取得了最低的平均有界延迟率 (253.31), 优于其他算法, 如 F1 (258.37) 和 RLScheduler (255.67). 这一结果表明, MRLScheduler 在高负载下能够有效降低任务的等待时间, 显示出较强的资源调度效率. 并且在 SDSC-SP2 中, MRLScheduler 的平均有界延迟率集中在 466.45, 同样优于其他对比算法.

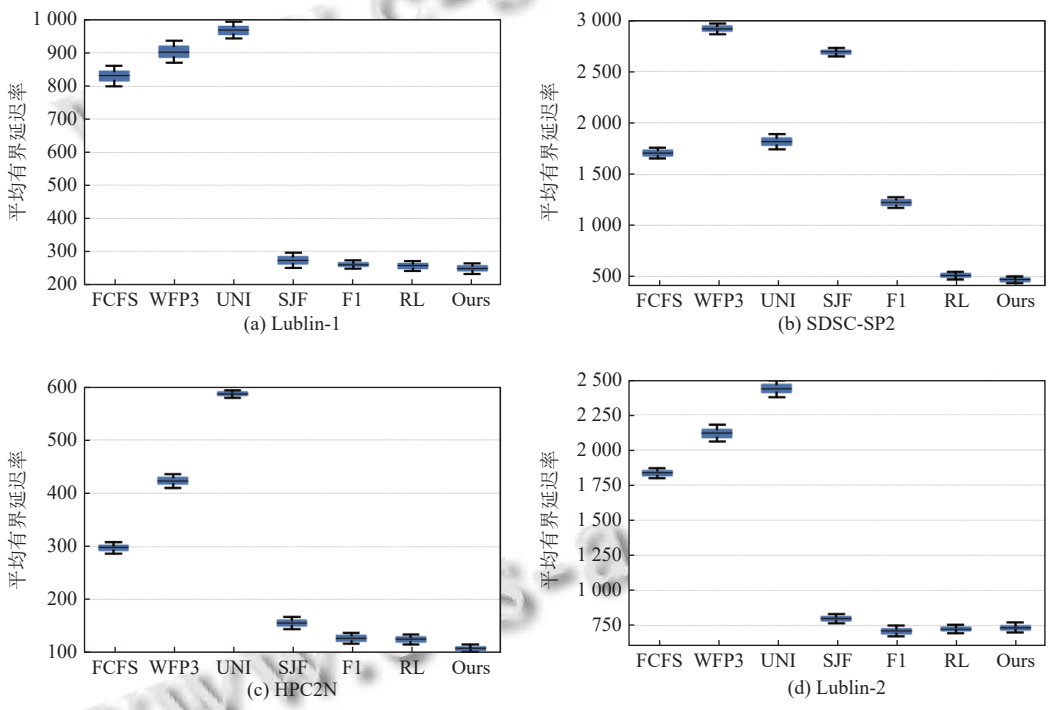


图 11 不同作业跟踪下各算法在不带回填时的平均有界延迟率对比

此外, MRLScheduler 在 HPC2N 中平均有界延迟率聚集在 112.54, 优于其他基线算法. 同样地, 在 Lublin-2 中, MRLScheduler 的平均有界延迟率达到 748, 与 F1 和 RL 算法的表现相近, 表明在大规模作业的调度情境下, MRLScheduler 能够提供稳定的效率优势, 并在不启用回填的情况下表现出一定的抗负载能力.

从图 12 中可以看出, 在启用回填情况下, 回填机制有效提升了所有算法的作业减速效率, 而 MRLScheduler 的优势更为显著. 例如, 在 Lublin-1 中, MRLScheduler 的平均有界延迟率进一步降低至 54.52, 优于其他所有对比算法, 其中 FCFS 的作业减速为 235.82, WFP3 为 133.87, 这显示出 MRLScheduler 在启用回填条件下的强大优化能力. 此外, RLScheduler 算法在该数据集中达到 58.64 的平均有界延迟率, 但仍然落后于 MRLScheduler 的表现, 进一步证明了 MRLScheduler 在高效资源配置的条件下能够进一步优化调度效率.

同样的, 在 SDSC-SP2 中, MRLScheduler 的平均有界延迟率集中在 394.79, 表现略优于 RL 算法 (397.82),

远低于传统算法 FCFS (1 595.12) 和 WFP3 (1 083.12). 这一结果表明, 即使在较高负载下, MRLScheduler 也能有效利用回填机制来减少作业等待时间, 体现了其在动态资源调度中的稳健性能. 并且, 在 HPC2N 数据集中, MRLScheduler 的平均有界延迟率为 67, 优于基线算法 F1 的 71.95.

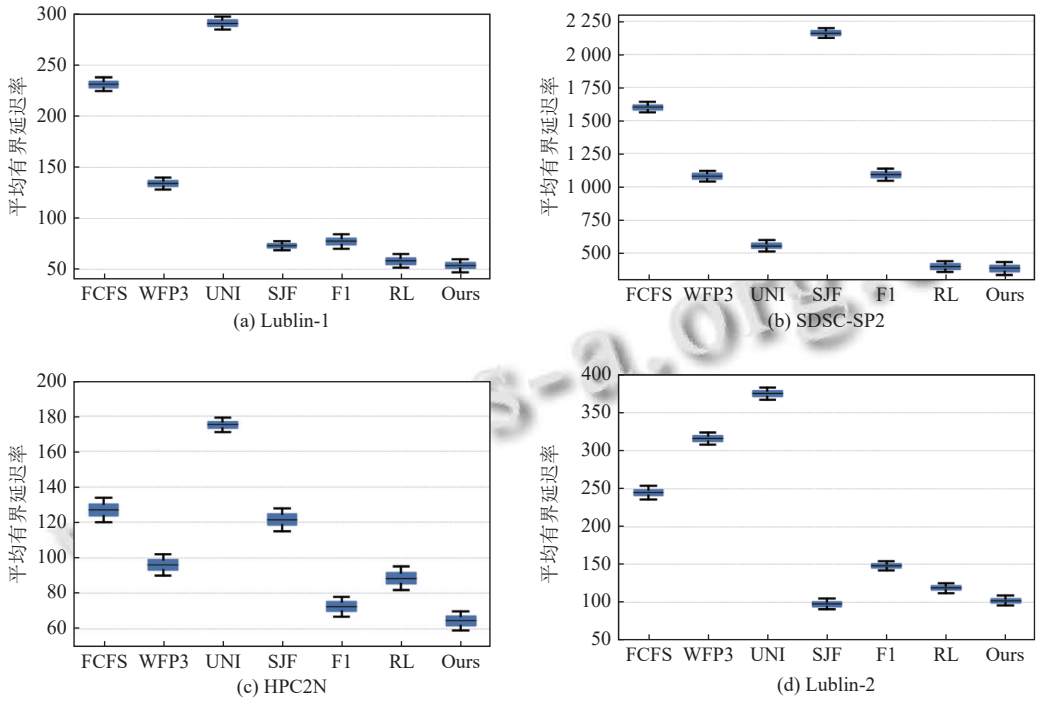


图 12 不同作业跟踪下各算法在带回填时的平均有界延迟率对比

此外, 在 Lublin-2 中, MRLScheduler 的平均有界延迟率降至 105.26, 明显低于 FCFS 的 247.16 和 RLScheduler 的 118.79, 接近于最佳算法 SJF 的 91.99. 这展现了其在动态资源配置条件下的显著效率优势. 通过回填机制, MRLScheduler 能够显著降低作业的等待时间, 从而提升整体系统的任务响应速度.

综上所述, MRLScheduler 在两种模式下均表现出显著的作业减速优势, 特别是在启用回填的高效资源配置情境下, 其作业减速性能最为突出, 能够最大限度地优化作业执行效率. 在大多数数据集中, MRLScheduler 相比传统算法在两种环境中均展现出优越的适应性和调度效率, 这证明了该算法在实际计算机集群调度任务中的应用潜力.

3.4 消融实验

本节通过逐步移除 MRLScheduler 中的关键模块, 验证其对任务泛化能力和策略优化效果的独立贡献. 实验设置 4 组对比模型.

MRLScheduler: 完整模型, 包含所有模块.

MRLScheduler w/o DG: 移除数据生成模块, 仅使用原始数据集训练元学习.

MRLScheduler w/o ER: 移除经验回放模块, 不使用历史经验及合成经验.

MRLScheduler w/o ML: 替换改进元学习为传统元学习 (如 MAML).

消融实验基于 Lublin-1、SDSC-SP2、HPC2N 及 Lublin-2 这 4 个数据集, 以平均有界延迟率和资源利用率作为核心评价指标, 实验结果如表 1 所示.

所有实验在相同超参数 (学习率 β=0.001, 批次大小 B=64, 正则化系数 λ=0.1) 下进行, 并固定随机种子以确保公平性.

如表 1 所示, 替换改进的元学习 (MRLScheduler w/o ML) 后, 在动态场景 (Lublin-2) 中, 完整模型的 ABDR 为 105.67, RU 为 0.607, 优于移除改进元学习的模型. 表明改进元学习通过动态参数组合机制有效适应了任务分布的动态变化, 优化了策略权重分配.

移除数据生成模块 (MRLScheduler w/o DG) 后, 模型在动态场景 (Lublin-2) 中的 ABDR 由完整模型的 105.67 升至 122.4, RU 由 0.607 降至 0.584. 在科学计

算场景 (SDSC-SP2) 中, ABDR 从 394.79 升至 430.6, RU 由 0.704 降至 0.684. 结果表明, 数据生成模块通过扩散模型生成多样化的 SAR 序列, 显著缓解了原始数据集的不均衡性.

表 1 MRLScheduler 的消融实验结果

模型	Lublin-1		SDSC-SP2		HPC2N		Lublin-2	
	ABDR	RU	ABDR	RU	ABDR	RU	ABDR	RU
MRLScheduler w/o ML	68.18	0.839	**392.91**	0.689	89.77	0.631	125.67	0.585
MRLScheduler w/o DG	62.4	0.845	430.6	0.684	82.1	**0.658**	122.4	0.584
MRLScheduler w/o ER	65	0.856	415.2	0.692	75.3	0.642	112.8	0.592
MRLScheduler	**54.52**	**0.874**	394.79	**0.704**	**67**	0.654	**105.67**	**0.607**

注: 加粗数据表示最优

移除经验回放模块 (MRLScheduler w/o ER) 后, 模型在高资源需求场景 (HPC2N) 中的 ABDR 由 67 升至 75.3, RU 由 0.654 降至 0.642. 稳态负载场景 (Lublin-1) 中, ABDR 从 54.52 升至 65, RU 由 0.874 降至 0.856. 经验回放模块的缺失导致模型无法高效复用历史经验, 策略优化效率降低. 该模块通过合成经验扩展经验缓冲池规模, 从而加速策略收敛.

为验证 MRLScheduler 的鲁棒性, 实验测试了在负载突变场景下 (作业到达率从 100 tasks/s 突增至 500 tasks/s), 记录模型恢复至稳定状态所需的训练周期数. 表 2 展示了各模型在该场景下的恢复能力与资源利用率波动.

表 2 各模型负载突变场景下的恢复能力与资源利用率

模型	恢复周期数	RU最大改变量
MRLScheduler	9	0.07
MRLScheduler w/o ML	16	0.10
RLScheduler	21	0.14

如表 2 所示, 在作业到达率从 100 tasks/s 突增至 500 tasks/s 的负载突变场景中, MRLScheduler 表现出优越的鲁棒性, 恢复周期数最短 (9 周期), 资源利用率波动仅为 0.07. 相比之下, 移除改进元学习后恢复周期数增至 16 周期, ΔRU 升至 0.1, 而 RLScheduler 需 21 周期恢复且 ΔRU 高达 0.14. 这一结果验证了改进元学习的动态参数调整机制能够快速重组策略权重, 结合经验回放模块生成的合成经验提供任务间关联信息, 显著缩短策略优化时间.

实验进一步验证模型在异构资源环境中的任务分配效率. 实验设置资源类型分布为: GPU (80%)、TPU (15%)、CPU (5%). 表 3 展示了异构资源环境下的任务分配效率.

如表 3 所示, 在异构资源环境中, MRLScheduler 的 GPU 利用率为 89.2%, TPU 利用率为 83.6%, 显著优于 RLScheduler. 改进元学习通过动态参数组合优先分配高负载任务至 GPU/TPU. 此外, 数据生成模块通过合成多样化数据增强了资源分配策略的泛化性, 避免了局部过载问题.

表 3 异构资源环境下的任务分配效率 (%)

模型	GPU利用率	TPU利用率
MRLScheduler	89.2	83.6
MRLScheduler w/o ML	82.5	78.1
RLScheduler	76.8	72.5

3.5 噪声策略与参数敏感性分析

本节分析了扩散模型中噪声策略及关键参数对调度性能的影响. 实验在 Lublin-2 和 SDSC-SP2 上进行, 重点评估平均有界延迟率和资源利用率等核心指标.

扩散模型的性能高度依赖噪声调度策略的选择. 本研究对比了 3 种典型策略.

线性调度: $\beta_t = \beta_{\min} + (\beta_{\max} - \beta_{\min}) \cdot \frac{t}{T}$

余弦调度: $\beta_t = \cos\left(\frac{t}{T} \cdot \frac{\pi}{2}\right)$

指数调度: $\beta_t = \beta_{\min} \cdot \left(\frac{\beta_{\max}}{\beta_{\min}}\right)^{\frac{t}{T}}$

其中, $\beta_{\min}$=0.0001, $\beta_{\max}$=0.02, T=800 为扩散总步数. 实验额外引入了 FID 评估合成数据分布与真实分布的相似性 (值越低表示质量越高), 其通过计算特征空间的分布距离, 量化保留关键调度特征的能力. 同时采用恢复步数衡量负载突变后的稳定性.

如表 4 所示, 余弦调度的 ABDR 较指数调度降低 2.3%, RU 提升了 1.5%. 在负载突变场景下, 余弦调度恢复步数仅需 9 步, 较线性调度减少 31%.

表 4 噪声调度策略性能对比

噪声策略	ABDR	RU	FID	恢复步数
线性调度	114.8	0.592	28.2	13
余弦调度	105.7	0.607	12.8	9
指数调度	108.2	0.601	17.6	12

这可能是由于线性调度因恒定噪声衰减率破坏长尾任务特征. 指数调度在噪声添加初期衰减过快, 导致

资源突变的瞬态特征未被充分保留, 而余弦调度通过其周期性的噪声衰减模式更有效地平衡了特征保留与噪声平滑.

扩散步数 T 和嵌入维度 d 是影响模型容量与效率的核心参数. 实验评估了不同参数组合的 ABDR 表现.

如表 5 所示, 在扩散步数 T 为 800, 嵌入维度 d 为 128 时 ABDR 最低为 394.9. 当 T>1000 且 d>192 时, ABDR 上升 3.4%, 这可能是因为模型过度拟合训练数据中的噪声模式.

表 5 不同参数组合对 ABDR 的影响

T	64	128	192	256
500	412.3	396.7	401.2	407.9
800	396.6	394.9	395.1	397.8
1 000	396.2	397.5	396.8	400.3
1 200	401.7	399.2	403.5	408.1

3.6 高频调度场景实时性分析

为验证方法在实际场景下的适用性, 实验模拟高频作业流环境: 基于 Lublin-2, 将到达间隔调整为 300 tasks/s, 对比传统启发式算法及 RLScheduler 的实时性. 实验依据工业调度系统标准设定阈值: 平均决策延迟≤20 ms、P99 延迟≤50 ms、积压率≤5%. 其中, P99 延迟是衡量系统尾部性能的关键指标, 表示在高频作业调度场景下, 99% 的调度决策的响应时间都低于该阈值. 该指标相比平均延迟更能揭示系统在最差情况下的性能表现, 对于保障调度服务的稳定性和一致性至关重要.

如表 6 所示, MRLScheduler 平均决策延迟为 9.7 ms, P99 延迟 24.2 ms, 显著优于 RLScheduler 的 39.6 ms. 积压率 4.1% 较 RLScheduler 降低 43%, 且远低于 FCFS 与 SJF. 这一优势源于改进元学习的轻量化策略微调机制: 预训练阶段 (扩散模型数据生成+元学习跨任务优化) 使在线调度仅需单次前向传播, 避免了传统 DRL 的复杂策略搜索开销. 同时, 合成数据增强的策略泛化性减少了动态环境下的决策迭代次数, 使积压率稳定低于阈值 5%. 传统算法虽延迟更低, 但固定策略难以应对负载波动, 导致积压率超标.

表 6 高频场景下各算法实时性表现

算法	平均决策延迟 (ms)	P99延迟 (ms)	积压率 (%)
FCFS	6.8±0.5	15.1±1.3	28.7±2.4
SJF	8.2±0.6	19.3±1.1	14.9±1.9
RLScheduler	18.5±1.2	39.6±2.8	7.2±0.7
MRLScheduler	9.7±0.3	24.2±1.5	4.1±0.3

3.7 其他场景下的性能分析

为了进一步验证算法在不同应用场景中的泛化性, 本节设计了云计算场景下的实验. 实验选用了云计算数据集 Google Cluster Data V2.

实验将 MRLScheduler 与 RLScheduler 以及传统启发式算法 FCFS、SJF 进行对比. 除继续使用平均有界延迟率和资源利用率这两个评价指标外, 实验还引入了任务平均响应时间和资源碎片率两个指标, 其中任务平均响应时间用以反映算法对小粒度任务的快速响应能力, 越小越好; 资源碎片率用以衡量资源利用的精细化程度, 越低代表资源管理越高效.

如表 7 所示, MRLScheduler 各项指标均优于其他基线算法. 具体而言, 与 FCFS 相比, MRLScheduler 的任务平均响应时间降低了约 31.1%, 资源碎片率减少了 38.7%, 表明 MRLScheduler 不仅能够更有效地管理并发的小粒度任务, 还可以显著提高资源利用精度.

表 7 云计算场景下各算法的性能

算法	ABDR	RU	任务平均响应时间 (ms)	资源碎片率
FCFS	231.45	0.51	298.7	0.31
SJF	140.18	0.56	325.4	0.27
RLScheduler	108.75	0.60	240.9	0.23
MRLScheduler	95.12	0.67	205.6	0.19

同时, 与 RLScheduler 相比, MRLScheduler 的平均有界延迟率降低了 12.5%, 资源利用率提升了 11.7%, 进一步验证了本文提出的改进元学习模块与扩散模型数据增强策略在云计算环境中的有效性.

综上所述, 实验验证了 MRLScheduler 在云计算场景中的出色性能.

4 总结

本文针对深度强化学习集群调度方法在应对高度动态的集群环境时泛化性不足的问题, 提出了一种改进元学习优化深度强化学习集群调度的方法, 旨在提高深度强化学习集群调度方法的泛化性. 实验结果表明, 该方法能够通过预测环境变化, 并动态调整调度策略, 在多变的环境中展现出较高的泛化性.

参考文献

1 Patil R, Gudivada V. A review of current trends, techniques, and challenges in large language models (LLMs). Applied Sciences, 2024, 14(5): 2074. [doi: 10.3390/app14052074]

2 Narayanan D, Shoeybi M, Casper J, *et al*. Efficient large-

scale language model training on GPU clusters using megatron-LM. Proceedings of the 2021 International Conference for High Performance Computing, Networking, Storage and Analysis. St. Louis: ACM, 2021. 58. [doi: 10.1145/3458817.3476209]

3 蒋筱斌, 熊轶翔, 张珩, 等. 基于 Kubernetes 的 RISC-V 异构集群云任务调度系统. 计算机系统应用, 2022, 31(9): 3–14. [doi: 10.15888/j.cnki.csa.008844]

4 陈妙云, 王雷, 盛捷. 基于值分布的多智能体分布式深度强化学习算法. 计算机系统应用, 2022, 31(1): 145–151. [doi: 10.15888/j.cnki.csa.008237]

5 齐玉峰, 贺晓. 基于深度学习的动态优先级任务调度算法. 计算机系统应用, 2023, 32(7): 195–201. [doi: 10.15888/j.cnki.csa.009169]

6 Luong NC, Hoang DT, Gong SM, *et al*. Applications of deep reinforcement learning in communications and networking: A survey. IEEE Communications Surveys & Tutorials, 2019, 21(4): 3133–3174. [doi: 10.1109/COMST.2019.2916583]

7 Botteghi N, Poel M, Brune C. Unsupervised representation learning in deep reinforcement learning: A review. IEEE Control Systems, 2025, 45(2): 26–68. [doi: 10.1109/MCS.2025.3534477]

8 Yao ZY, Ding ZH. Learning distributed and fair policies for network load balancing as Markov potential game. Proceedings of the 36th Conference on Neural Information Processing Systems. New Orleans: NeurIPS, 2022. 28815–28828.

9 Jayanetti A, Halgamuge S, Buyya R. Multi-agent deep reinforcement learning framework for renewable energy-aware workflow scheduling on distributed cloud data centers. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(4): 604–615. [doi: 10.1109/TPDS.2024.3360448]

10 Hospedales T, Antoniou A, Micaelli P, *et al*. Meta-learning in neural networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022, 44(9): 5149–5169. [doi: 10.1109/TPAMI.2021.3079209]

11 Vettoruzzo A, Bouguelia MR, Vanschoren J, *et al*. Advances and challenges in meta-learning: A technical review. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2024, 46(7): 4763–4779. [doi: 10.1109/TPAMI.2024.3357847]

12 Feurer M, Eggensperger K, Falkner S, *et al*. Auto-sklearn 2.0: Hands-free automl via meta-learning. The Journal of Machine Learning Research, 2022, 23(1): 261.

13 Yang L, Zhang ZL, Song Y, *et al*. Diffusion models: A comprehensive survey of methods and applications. ACM Computing Surveys, 2023, 56(4): 105. [doi: 10.1145/3626235]

14 Chen H, Xiang Q, Hu JX, *et al*. Comprehensive exploration of diffusion models in image generation: A survey. Artificial Intelligence Review, 2025, 58(4): 99. [doi: 10.1007/s10462-025-11110-3]

15 Ye HJ, Ming L, Zhan DC, *et al*. Few-shot learning with a strong teacher. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2024, 46(3): 1425–1440. [doi: 10.1109/TPAMI.2022.3160362]

16 Finn C, Abbeel P, Levine S. Model-agnostic meta-learning for fast adaptation of deep networks. Proceedings of the 34th International Conference on Machine Learning. Sydney: PMLR, 2017. 1126–1135.

17 Guo ZJ, Li XD, Han L, *et al*. Robust inference for federated meta-learning. Journal of the American Statistical Association, 2025. [doi: 10.1080/01621459.2024.2443246]

18 Lee JJ, Yoon SW. XB-MAML: Learning expandable basis parameters for effective meta-learning with wide task coverage. Proceedings of the 27th International Conference on Artificial Intelligence and Statistics. Valencia: PMLR, 2024. 3196–3204.

19 Jiang WS, Kwok J, Zhang Y. Subspace learning for effective meta-learning. Proceedings of the 39th International Conference on Machine Learning. Baltimore: PMLR, 2022. 10177–10194.

20 李耀芳, 刘国瑞, 洪姣. 数据中心差异化时延满足率优化调度. 计算机系统应用, 2023, 32(4): 77–85. [doi: 10.15888/j.cnki.csa.009046]

21 Pinedo ML. Scheduling: Theory, Algorithms, and Systems. New York: Springer, 2012. [doi: 10.1007/978-3-031-05921-6]

22 Blazewicz J, Ecker K, Pesch E, *et al*. Handbook on Scheduling. Cham: Springer International Publishing, 2019. [doi: 10.1007/978-3-319-99849-7]

23 Carastan-Santos D, De Camargo RY. Obtaining dynamic scheduling policies with simulation and machine learning. Proceedings of the 2017 International Conference for High Performance Computing, Networking, Storage and Analysis. 2017. 1–13. [doi: 10.1145/3126908.3126955]

24 Geurtsen M, Didden JBHC, Adan J, *et al*. Production, maintenance and resource scheduling: A review. European Journal of Operational Research, 2023, 305(2): 501–529. [doi: 10.1016/j.ejor.2022.03.045]

(校对责编: 张重毅)