DOI:10.20033/j.1003-7241.(2026)04-0140-06

融合深度学习与云计算的智能调度系统优化策略

宿洪智, 郑少明, 董 鹏, 刘 丹, 牟澎涛

(国家电网有限公司华北分部,北京 100053)

摘 要:随着云计算与深度学习技术的快速发展,传统智能调度系统在动态环境适应性、资源利用率及调度效率等方面的不足日益凸显。针对上述问题,提出了一种融合深度学习与云计算的智能调度系统性能优化策略。首先,构建了智能调度系统模型,系统性分析了深度学习技术在云计算环境中的集成方式;其次,以资源管理与调度决策优化为核心,设计并实现了一种多级反馈队列调度算法,通过动态优先级调整与老化机制提升系统公平性与响应能力;在此基础上,引入深度确定性策略梯度算法对调度策略进行进一步优化,以增强系统在复杂动态负载条件下的自适应性与决策精度。通过仿真实验,将所提方法与先来先服务、短作业优先及传统多级反馈队列等调度策略进行对比分析。结果表明,基于深度学习优化的多级反馈队列调度策略在响应时间、系统吞吐量、CPU 利用率及进程等待时间等关键性能指标上均表现出显著优势。研究结果验证了深度学习与云计算融合在智能调度系统优化中的有效性与可行性,为相关领域的工程应用与后续研究提供了有价值的参考。

关键词:深度学习;云计算;智能调度系统;性能优化;多级反馈队列调度算法;策略梯度,自适应性

中图分类号: TP18;TM28 **文献标志码**: A **文章编号**: 1003-7241(2026)04-0140-06

Intelligent scheduling system optimization strategy integrating deep learning and cloud computing

SU Hongzhi, ZHENG Shaoming, DONH Peng, LIU Dan, MOU Pengtao

(North China Branch of State Grid Corporation of China, Beijing 100053, China)

Abstract: With the continuous development of cloud computing and deep learning technology, improving the performance of intelligent scheduling systems by integrating different technologies gradually become a research focus. Based on this, the article proposes a performance optimization solution for an intelligent dispatching and protection system that integrates deep learning and cloud computing technology. First, based on the advantages of the algorithm, the integration ideas of deep learning technology and cloud computing environment are systematically analyzed. Afterwards, a multi-level feedback queue scheduling algorithm is designed based on the problem of resource management and scheduling decision optimization. Finally, the deep deterministic policy gradient algorithm is used to further optimize the scheduling strategy to enhance the system´s adaptability to dynamic environments and decision-making efficiency. Test results show that compared with traditional scheduling algorithms, the proposed performance optimization method significantly improves task processing efficiency and system resource utilization, providing new ideas and technical support for research in this field.

Keywords: deep learning; cloud computing; intelligent scheduling system; performance optimization; multi-level feedback queue scheduling algorithm; policy gradient; adaptability

市场对智能调度系统的主要需求包括实时性、自适应性、高资源利用率和智能决策支持,以及系统的可扩展性和灵活性。机器学习和人工智能技术的快速发展正在重塑数据处理和复杂计算任务的处理模式,在算法优化和应用拓展方面持续取得突破。先进算法理论的融合和优化框架的创新不仅提升了数据处理的效率和智能化水平,也拓展了其在智能制造、智慧城市等领域的应用,实现了更高的运算效率和资源配置的优化。因此,探索先进算法理论在智能调度系统中的融合应用以克服传统方法的局限性并提升调度策略的智能化水平具有重要的研究价值。

为了提升智能调度系统的综合性能,研究者开展了诸多研究[1]。在现有研究成果中,文献[2]提出通过网络化策略优化调度命令增加系统的响应速度和灵活性。文献[3]指出传统电力系统调度策略在处理大规模数据和实时性要求时效率不足,在多种应用场景中存在局限。随着技术的进一步发展,文献[4]研究通过引入机器学习和人工智能技术弥补传统调度方法的不足,探索人机混合增强决策智能的应用。尽管这些高级技术提供了新的可能性,

收稿日期:2024-05-28;**录用日期**:2024-07-08

基金项目:国家电网有限公司华北分部大修项目(45992319002L)

作者简介:宿洪智(1990—),男,博士,高级工程师,研究方向:电力系统保护与控制等。

引用本文:宿洪智, 郑少明, 董鹏, 等. 融合深度学习与云计算的智能调度系统优化策略[J]. 自动化技术与应用, 2026,45(4):140-145. (SU Hongzhi, ZHENG Shaoming, DONH Peng, et al. Intelligent scheduling system optimization strategy integrating deep learning and cloud computing[J]. Techniques of Automation and Applications, 2026,45(4):140-145.)

但实际应用过程面临算法复杂性高和实施成本大的挑战，仍需深入研究更为智能化的优化策略。

随着深度学习与云计算技术的迅速发展，智能调度系统的研究已取得重大进展[5]。文献[6]基于深度强化学习开发了一种动态化任务调度方法，优化了任务分配并提高了电子政务云系统的运行效率。之后，文献[7]采用云计算环境下的资源调度与优化算法，有效提高了资源利用率及服务质量。在 AI 算力平台与云计算融合方面的研究中[8-9]，文献[10]展示了在大数据环境下边缘计算和雾计算结合深度学习的潜力。文献[11]验证了利用深度学习解决云计算中任务调度问题的有效性。文献[12-13]从机器学习算法的有效性及成本意识的调度策略角度，进一步推动了调度技术的发展。上述研究成果证实了深度学习技术与云计算融合的有效性，但在智能调度系统优化方面的性能仍需进一步分析与研究[14-17]。

总结上述分析和对已有研究成果的总结，传统调度系统由于效率低下和资源分配不均，常常难以满足快速变化的市场需求。基于此，本研究通过整合深度学习技术和云计算资源，设计了一种新型智能调度保护系统性能优化策略，以提升调度决策的实时性和准确性，优化资源配置与管理，降低运维成本，并增强系统的可扩展性与安全性。

1 多级反馈队列调度算法

智能调度系统在云计算、制造业、交通物流等领域中发挥着至关重要的角色。该系统通过优化资源分配，适应不断变化的环境和需求，以实现成本效益和效率的最大化，显著提升运营效率。

1.1 智能调度系统模型

智能调度系统通过动态地适应环境变化和需求，可以最大化效率和成本效益。智能调度系统通常包括以下步骤。

1）数据收集与分析

系统首先通过传感器和用户反馈收集实时数据，如任务类型、资源使用情况、用户需求等。数据分析则主要依赖于机器学习技术来预测未来需求和系统状态。

预测模型可描述为

$$y_t = f(x_t, \theta) \tag{1}$$

其中，y_t 是时间 t 的预测输出，x_t 是输入特征，θ 是模型参数。

2）任务与资源的匹配

根据分析结果，系统选择相应的优化算法为即将到来的任务动态匹配和分配资源。优化问题的可描述为

$$\min_x C(x), \text{ subject to } g(x) \leqslant 0 \tag{2}$$

其中，$C(x)$ 表示成本函数，x 表示决策变量，$g(x)$ 表示约束条件。

3）调度策略的执行

智能优化调度系统根据优化算法的结果执行调度策略，实时调整资源以适应需求变化。调度决策函数可描述为

$$d_t = \text{argmin}_l \sum_{i=1}^{n} \alpha_i \cdot c_i(t, d) \tag{3}$$

其中，d_t 是在时间 t 的调度决策，α_i 是权重因子，$c_i(t,d)$ 是成本或效用函数。

1.2 多级反馈队列调度策略

多级反馈队列调度策略是操作系统中一种高效的进程调度方法，该方法综合利用轮转调度和优先级调度的特点，动态地调整进程的优先级，从而优化系统的响应时间和进程执行效率。

1.2.1 基本原理

利用多级反馈队列调度策略开展任务优化调度时的步骤可描述为：

1）初始化，系统启动时，所有新进程都被放置在最高优先级的队列。

2）任务执行，从最高优先级（非空）队列开始，选择一个进程执行其时间片。

3）优先级调整，一个进程的时间片耗尽后，根据其行为（如 CPU 密集型或 I/O 密集型）将其移动到更高或更低的优先级队列。

4）循环调度，系统反复检查各队列，确保所有进程获得执行机会。

假设有 n 个队列，每个队列 Q_i 有一个固定的时间片长度 t_i。每个新进程首先被放置在 Q_1：

时间片分配过程可描述为

$$t_i = 2^{i-1} \times t_{\text{base}} \tag{4}$$

其中，t_{base} 是基础时间片长度，i 是队列的级别（Q_1 是最高优先级）。

优先级调整规则可表示为，如果进程在其时间片内完成并进入等待状态（如等待 I/O），它可能被移动回更高优先级的队列。如果进程使用完其时间片仍然需要更多 CPU 时间，则被移至下一级的队列。

$$Q_{\text{new}} = \begin{cases} \max(1, Q_{\text{old}} - 1), & \text{I/O - bound} \\ \min(n, Q_{\text{old}} + 1), & \text{CPU-bound} \end{cases} \tag{5}$$

其中，Q_{old} 和 Q_{new} 分别是进程移动前后的队列等级。

1.2.2 老化处理

老化处理是多级反馈队列调度策略中防止进程饥饿的重要机制。它通过逐渐提升长时间未能执行的进程的优先级来实现。如果进程在较低优先级队列中等待超过一定时间阈值，即

$$(t_{\text{current}} - t_{\text{last\_executed}}) > T_{\text{threshold}} \tag{6}$$

其优先级将被提高，使其有机会较早被调度，有

$$Q_{\text{new}} = \max(Q_{\text{old}} - 1, 1) \tag{7}$$

其中，t_{current} 是当前时间，$t_{\text{last\_executed}}$ 是进程最后一次执行的时间，$T_{\text{threshold}}$ 是时间阈值，Q_{old} 和 Q_{new} 分别是进程当前和新的队列等级。

通过老化处理和动态优先级调整，多级反馈队列调度策略确保系统中没有进程会因优先级太低而永久等待执行，同时为各类型进程提供了合理的响应时间。

1.2.3 任务分类与优先级划分

多级反馈队列调度策略在任务分类与优先级划分方

面采取综合动态调整方法,以优化进程响应时间和执行效率。此策略主要针对不同类型的任务(如 CPU 密集型与 I/O 密集型)实施适当的优先级调整,确保高效率与公平性的平衡。初始优先级由任务类型确定,随任务执行动态调整。实时或紧急任务获高优先级,非紧急任务则按需分配到较低优先级队列。

优先级评分 P_i 需考虑等待时间 T_{wait}, CPU 使用时间 T_{cpu}, 及任务类型 Type。

$$P_i = \alpha \times T_{\text{wait}} + \beta \times T_{\text{cpu}} + \gamma \times \text{Type} \tag{8}$$

其中, α,β,γ 是权重参数,调整以适应不同调度需求。

若 P_i 高于阈值 Threshold, 提升至高优先级队列;若 P_i 低于阈值,降低至低优先级队列。通过这种方法,多级反馈队列调度策略能够有效地管理各种类型的任务,优化系统性能。

1.2.4 资源分配

资源分配基于进程的优先级和所需时间片进行,优先级高的进程分配更多 CPU 时间,以快速响应紧急任务。时间片的分配则依据进程在当前队列中的等待时间和过去的 CPU 使用率动态调整,确保公平性与效率的平衡。

每个队列有固定的时间片 $\text{quantum}_i, i = 1,2,\cdots,n$, 基于进程的执行历史和优先级动态调整。利用老化技术和反馈机制调整进程优先级,未完成的进程时间片耗尽后,根据其行为降低或提升优先级。

$$\text{quantum}_{i+1} = \text{quantum}_i \times f(W_i, C_i) \tag{9}$$

其中, W_i 为进程在队列 Q_i 中的等待时间, C_i 为至今消耗的 CPU 时间,函数 f 根据系统策略确定,通常是增加或减少常数因子。

$$P_{\text{new}} = P_{\text{old}} + g(W_i, P_{\text{old}}, T_{\text{age}}) \tag{10}$$

P_{old} 和 P_{new} 分别为调整前后的优先级, g 是根据等待时间和老化阈值 T_{age} 调整优先级的函数。

1.2.5 性能评估指标

评估多级反馈队列调度策略的性能需要综合考虑各种指标,包括响应时间、系统吞吐量、CPU 利用率和进程等待时间等,每个指标都从不同角度反映调度策略的性能优劣。

1) 响应时间 (T_{response})

响应时间是从提交进程到首次被调度运行的时间间隔。此指标对于用户体验至关重要,特别是在交互式系统中。

$$T_{\text{response}} = T_{\text{first-run}} - T_{\text{submit}} \tag{11}$$

2) 系统吞吐量 (Θ)

系统吞吐量表示单位时间内系统完成进程的数量。高吞吐量意味着系统能够有效地处理大量进程。

$$\Theta = \frac{N_s}{\text{Unit time}} \tag{12}$$

3) CPU 利用率 (U_{CPU})

CPU 利用率是指 CPU 工作时间在总时间中的占比。高 CPU 利用率通常表示调度策略可以有效利用处理器资源。

$$U_{\text{CPU}} = \frac{\text{CPU busy time}}{\text{Total time}} \tag{13}$$

4) 进程等待时间 (W_{process})

进程等待时间是指进程提交到系统后,在就绪队列中等待调度的时间总和。优化调度策略旨在减少此等待时间,提升效率。

$$W_{\text{process}} = \sum_{i=1}^{n} (T_{\text{start},i} - T_{\text{submit},i}) \tag{14}$$

式中, $T_{\text{first-run}}$ 和 T_{submit} 分别代表进程首次运行和提交的时间点。N_s 是指在单位时间内完成的进程数。CPU busy time 和 Total time 分别指 CPU 处于工作状态和总的观察时间。$T_{\text{start},i}$ 和 $T_{\text{submit},i}$ 分别是进程 i 的启动和提交时间。

2 深度学习-云计算策略

2.1 资源调度的深度学习优化模型

在智能调度系统优化方面,核心目标为提升任务处理的效率以及系统资源的利用率。深度确定性策略梯度算法因其卓越的能力在于处理连续且高维的动作空间特别适用于这类问题。通过与环境的持续交互,该算法能有效地在复杂的动态系统中学习并迭代出最优的策略。

在所提出的基于深度学习技术和云计算环境的集成框架内,深度学习模型能够充分利用云计算环境所提供的海量数据资源,进行任务需求的预测学习。之后,基于这些预测结果,深度确定性策略梯度算法被用以实施决策优化,优化资源分配过程并提升智能调度系统的整体性能。具体优化步骤可总结为

步骤 1 初始化网络。过程包括设置策略网络和价值网络的权重 θ^{μ} 和 θ^{Q}。该步骤是策略网络学习选择有效动作和价值网络评估这些动作价值的基础。

步骤 2 收集经验。在该阶段,系统按照设定的时间间隔进行状态监测和行动评估,执行如下细化操作。

1) 基于当前状态 s,智能调度系统利用策略网络,按照确定性策略 $\mu(s \mid \theta^{\mu})$ 选择对应的调度行为 a。

2) 实施行为 a,并监测由此引发的奖励 r 和状态转移至 s',此处 s' 表示行为 a 执行后的新系统状态。

3) 这一状态-行为-奖励-新状态序列 (s,a,r,s') 被记录并存储于经验回放缓冲区内,作为后续训练过程中的学习样本。

系统的这一操作过程不仅保证了实时行为决策的有效性,而且通过经验回放机制,促进了决策模型在历史数据中学习与提炼,以期达到调度行为的最优化。

步骤 3 样本回放。系统从经验回放缓冲区中随机抽取一组样本,降低样本间相关性并防止过拟合现象的发生。随机抽样得到的经验集合为模型提供了一种丰富的学习源,其中包含了多样化的状态转换情景,有利于模型学习泛化的决策策略。

步骤 4 更新价值网络。

对于经验回放缓冲区中随机抽取的每个样本,价值网络将基于当前策略网络的输出和奖励信号来进行更新。

此过程的目标是减小估计值和目标值之间的差距,以提高预测的准确性。更新价值网络时,借助下述公式计算目标 Q 值 y_i 为

$$y_i = r_i + \gamma Q'(s_i', \mu'(s_i' \mid \theta^{\mu'}) \mid \theta^{Q'}) \tag{15}$$

式中,s_i' 是新状态,γ 是折扣因子,表明未来奖励的重要性。

之后,根据损失函数 $\Gamma(\theta^Q)$,利用梯度下降方法更新网络权重,以减小模型预测值与目标 Q 值之间的误差。损失函数定义如下,即

$$\Gamma(\theta^Q) = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i \mid \theta^Q))^2 \tag{16}$$

式中,N 是抽取的样本数量。

权重按照下述公式进行更新,即

$$\theta^Q \leftarrow \theta^Q + \alpha^Q \nabla_{\theta^Q}\Gamma(\theta^Q) \tag{17}$$

式中,α^Q 是学习率,决定了更新步长的大小。

步骤 5 更新策略网络。为了提高策略网络的效率和效果,系统首先计算策略梯度。这一计算依赖于价值网络(Critic)提供的梯度反馈,用于指导策略网络调整其参数为

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a \mid \theta^Q)\mid_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s \mid \theta^\mu)\mid_{s_i} \tag{18}$$

式中,s_i 是抽取样本的状态,$\mu(s_i \mid \theta^\mu)$ 是由当前策略产生的动作。

使用上述计算得到的梯度,通过梯度上升方法更新策略网络的权重,以最大化期望回报。更新公式为

$$\theta^\mu \leftarrow \theta^\mu + \alpha^\mu \nabla_{\theta^\mu} J \tag{19}$$

步骤 6 网络软更新。为避免在学习过程中由于参数快速变化引起的不稳定性,系统采用软更新策略逐渐调整目标网络的参数。软更新公式表达为

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{20}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{21}$$

式中,$\tau \ll 1$ 是软更新参数,表示更新的速率,确保目标网络参数的更新是渐进和平滑的。

2.2 深度学习模型的部署

部署深度学习模型到云计算环境中涉及多个关键技术和策略,以确保模型的高效运行和维护。以下是部署策略的详细说明及其对应的技术要求。

1) 容器化部署。容器化技术如 Docker 提供一种轻量级的方式来封装深度学习模型及其依赖环境,确保在不同的云平台上可以一致地运行。

2) 自动化部署。利用持续集成/持续部署(CI/CD)流程自动化模型的部署过程。这包括自动化测试、构建和部署步骤,可以加速部署周期,减少人为错误。

3) 弹性伸缩。云平台应支持自动弹性伸缩,以便根据实时需求动态调整计算资源。这不仅可以优化资源使用,减少成本,还可以在负载增加时保持服务的稳定性和可用性。

4) 性能监控与优化。部署后的模型需要持续的监控以确保其性能达标。

5) 安全性保障。确保部署在云环境中的模型具有高级别的安全性,防止数据泄露和未授权访问。

通过上述策略和技术要求,可以有效地部署和维护深度学习模型在云计算环境中的应用,确保模型的性能和安全性,同时优化资源使用和操作成本。

2.3 算法整体框架

图 1 展示了深度学习模型在性能优化方面的部署和调整过程。

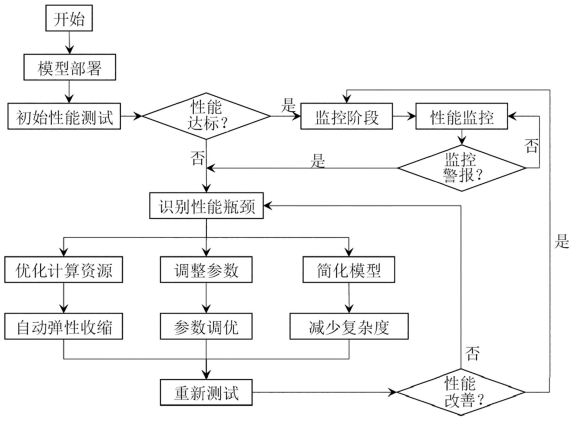


图 1 深度学习模型的部署和调整过程

Fig. 1 Deployment and tuning process of deep learning model

首先,模型部署阶段涉及将训练好的模型投入实际使用环境。之后,初始性能测试是评估模型是否满足预期性能的重要步骤。如果性能未达标准则开始识别性能瓶颈,以查明性能不足原因并制定改进策略。性能改进措施包括计算资源的优化、模型参数的调整及模型结构的简化。实施必要优化后,模型将重新进行性能测试。若性能符合要求则进入持续监控阶段,反之则需要回到识别瓶颈阶段进行进一步的优化。持续监控阶段会不断检测和应对潜在性能问题,确保模型持续运行在最佳状态。

3 测试和结果分析

以 1.2.5 节所建立的 4 个指标为分析依据,通过仿真对比了多级反馈队列调度策略以及利用深度学习方法优化后的调度性能。为了验证多级反馈队列策略的有效性,对比了基于深度学习优化的多级反馈队列(deep deterministic policy gradient optimized multilevel feedback queue, DDPGMFQ)、多级反馈队列(multilevel feedback queue, MFQ)、先来先服务(first come first served,FCFS)、和短作业优先(shortest job first,SJF)四种调度策略。主要结果如下。

图 2 对 DDPGMFQ、MFQ、FCFS、SJF 4 种调度策略的响应时间进行了比较,以评估这些策略在不同进程数量下的效率。根据图 2,响应时间作为调度系统效率的关键指标,直接影响系统的性能表现。DDPGMFQ 策略和 MFQ 策略在进程数量增加时表现出非线性的响应时间下降,显示了其优越的动态调整能力。同时,利用深度学习方法优化 MFQ 策略后,可以显著提升其性能,进一步减少了在各

个进程数量下的相应时间。FCFS 策略虽然响应时间整体下降,但在大量进程处理时效率提升不显著。SJF 策略则在处理短作业时效果较好,但整体性能仍低于 MFQ。因此,从图 2 所示的结果来看,MFQ 策略在高进程数量的环境下明显优于 FCFS 和 SJF 策略,特别是在动态和复杂的负载条件下,MFQ 的时间片和优先级调整机制显著改善响应时间和处理效率。这一点在云计算等需要处理大量或多种类型进程的应用场景中尤为重要。

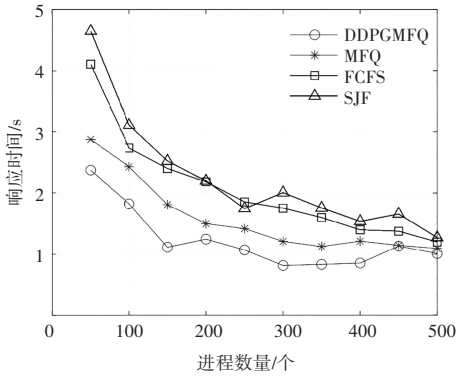


图 2 响应时间对比

Fig. 2 Response time comparison

图 3 对比了四种调度策略在处理不同进程数量时的系统吞吐量表现。测试时进程数量从 100 逐渐增加到 1 000。每增加 100 个进程执行一次性能测量以评估策略在不同系统负载下的表现。根据图 3,随着进程数量的增加,DDPGMFG 策略和 MFQ 策略吞吐量显著提升,对高负载环境具有良好适应性。这一结果得益于其时间片和优先级的动态调整,使其在处理大量进程时能有效提升处理效率。此外,基于深度学习方法的优化思路也被证明是可行且有效的,在不同城市数量下的系统吞吐量最高。相比之下,FCFS 策略虽然吞吐量随进程增加而提高,但增速相对较慢,特别是在进程数量较多时,由于缺乏灵活的调度机制,性能提升有限。SJF 策略在处理短作业时表现较好,但在高进程环境下表现差于 DDPGMFQ 和 MFQ 策略。

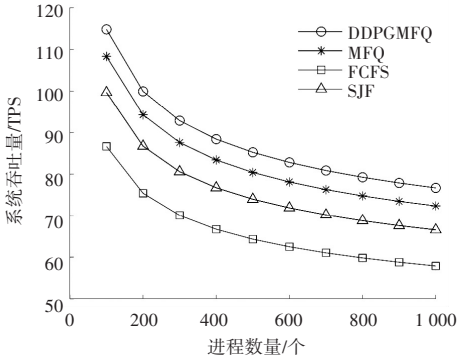


图 3 系统吞吐量对比

Fig. 3 System throughput comparison

图 4 对比了 4 种调度策略在处理不同类型进程时的 CPU 利用率。根据图 4,DDPGMFG 策略和 MFQ 策略表现出显著的优势,其 CPU 利用率最高达 95%,反映出其出色的资源管理能力。这一优势得益于 MFQ 在动态调整时间片和优先级方面的高效性。在处理 CPU 密集型任务时,该策略能够迅速适应不同的需求,优化资源分配。在深度学习算法的介入下,DDPGMFG 策略获得了最好的性能,验证了文章的优化思路的可行性与正确性。在另一方面,由于缺乏灵活的调度机制,FCFS 策略的 CPU 利用率相对较低,最高仅 85%,表明其在处理高负载情况下的性能有限。此外,SJF 策略的性能介于 MFQ 和 FCFS 之间,其最大利用率为 90%,表明其在处理短作业时较为有效,但在复杂多变的负载下仍显示出一定的局限性。

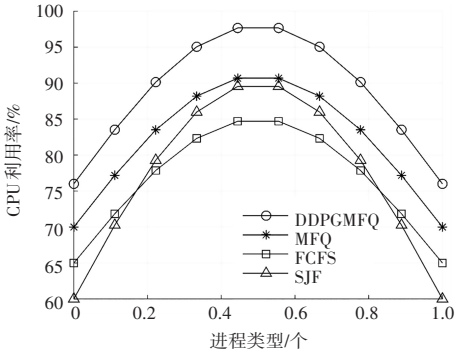


图 4 CPU 利用率对比

Fig. 4 CPU utilization comparison

图 5 对比了 4 种调度策略在处理不同进程数量下的等待时间。进程数量从 50 逐步增加到 500 的过程中,DDPGMFQ 策略和 MFQ 策略显示出在高进程数量下的显著优势,其等待时间随进程数量的增加呈现出快速下降的趋势。利用深度学习方法优化 MFQ 策略后,其对应的等待时间进一步减少,表明其性能的改进和提升。与此相比,FCFS 策略的等待时间逐渐减少,降低速度较慢,尤其是在进程数量较多时,其性能提升不明显,反映出其在高负载情况下处理效率的不足。SJF 策略的表现虽然优于 FCFS,但其等待时间的减少速度也低于 MFQ 的效率。

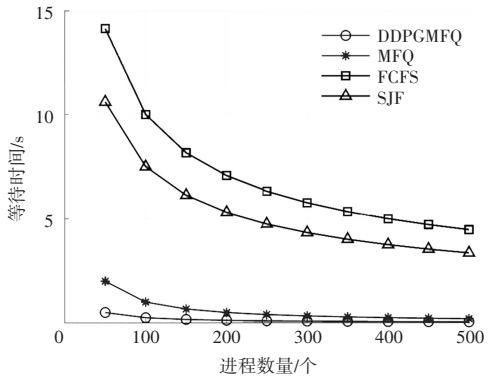


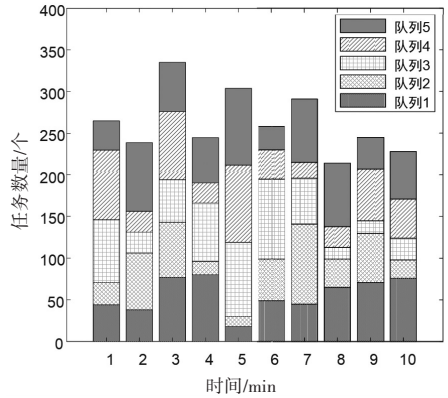
图 5 进程等待时间对比

Fig. 5 Comparison of process wait times

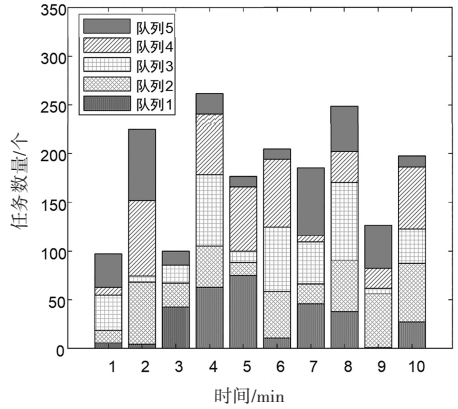
为了进一步分析文章所提利用深度学习算法优化多级反馈队列策略的可行性与有效性,对比了两种方法对应的任务调度结果,如图 6 所示。

根据图 6 可以明显观察到优化后各队列中任务的累积数量普遍降低,表明任务处理的平均响应时间有所减少。优化后的调度策略显著缩短了任务从开始到完成的

时间,有效地减少了任务在队列中的等待时间。同时,优化后的系统具有更高的处理效率,系统吞吐量得到了提升。通过对调度策略的优化,使得系统能够在面对高负载时,更加高效地分配任务到相应的处理队列。另一方面,从任务数量的减少可以间接推断出系统资源利用率得到了优化。由于深度学习算法对任务特性和系统状态的精确预测,调度系统可以更合理地分配资源,避免了资源的浪费和过度集中,使得资源利用更加均衡和充分。最后,优化后的数据显示任务在不同优先级队列中的分布更为均匀,这表明调度系统的负载均衡性有所提高。通过深度学习优化,系统能够根据当前的负载情况和任务性能要求,动态调整任务优先级,优化任务队列处理顺序,从而达到更优的负载均衡效果。



(a) 优化前的任务调度结果



(b) 优化后的任务调度结果

图 6　优化前后的任务调度结果对比

Fig. 6　Comparison of task scheduling results before and after optimization

综上所述,深度学习算法的引入为智能调度系统带来了显著的性能提升,上述结果验证了深度学习算法在智能调度领域的应用价值,展示了其在处理复杂调度问题时的巨大潜力。

4　结论

通过整合深度学习与云计算技术,文章设计一个融合深度学习算法的智能调度系统优化策略。通过深度学习模型预测资源需求与潜在系统负荷,配合实时数据动态调整资源分配,显著提升了系统响应速度与决策精度;其次,利用云计算弹性资源池的特性,动态管理资源分配,有效减少了资源闲置和过度配置,从而优化资源利用率。通过在不同的模拟环境中对系统进行测试,结果表明所设计优化后的调度策略能够在保持高吞吐量的同时,有效减少任务等待时间和系统的总运营成本。

未来的研究将聚焦于模型的简化和优化,提高系统的实时动态调度能力,并扩展模型到更多实际应用场景中,以克服当前的限制并进一步提高调度系统的智能化水平和适应能力。

参考文献

[1]李鹏, 黄文琦, 梁凌宇, 等. 人机混合增强决策智能在新型电力系统调控中的应用与展望[J]. 中国电机工程学报, 2024, 44(16): 6347-6367.
[2]张贝, 王亮, 袁少伟, 等. 电力调度智能网络化下令系统功能研究[J]. 水电站机电技术, 2024, 47(3):41-42.
[3]袁源. 智能电网技术在电力系统调度中的实践[J]. 模具制造, 2024, 24(3):206-208.
[4]李鹏, 黄文琦, 梁凌宇, 等. 人机混合增强决策智能在新型电力系统调控中的应用与展望[J]. 中国电机工程学报, 2024,44(16): 6347-6367.
[5]范霁清. 智能技术在电力调度自动化系统中的应用[J]. 集成电路应用, 2024, 41(2):90-91.
[6]龙宇杰, 修熙, 黄庆, 等. 基于深度强化学习的电子政务云动态化任务调度方法[J/OL]. 计算机应用研究, 1-8[2024-04-14]. https://doi. org/10. 19734/j. issn. 1001-3695. 2023. 10. 0527.
[7]张桂兰. 云计算环境下的资源调度与优化算法研究[J]. 信息系统工程, 2023(12):64-67.
[8]卞慧敏. AI算力平台与云计算融合探究[N]. 山西科技报, 2024-01-22(B03).
[9]洪俊. 云边端场景下基于深度学习的推荐方法研究[D]. 南京: 南京邮电大学, 2023.
[10]张亚男, 李德龙, 杨颜靖. 基于边缘计算、雾计算和深度学习的大数据智能分析[J]. 信息与电脑(理论版), 2023, 35(13):167-169.
[11]BADRI S, ALGHAZZAWI D M, HASAN S H, et al. An efficient and secure model using adaptive optimal deep learning for task scheduling in cloud computing[J]. Electronics, 2023, 12(6):1441.
[12]SRIKANTH G U, GEETHA R. Effectiveness review of the machine learning algorithms for scheduling in cloud environment[J]. Archives of Computational Methods in Engineering, 2023, 30(6):3769-3789.
[13]CHENG L, WANG Y, CHENG F, et al. A deep reinforcement learning-based preemptive approach for cost-aware cloud job scheduling[J]. IEEE Transactions on Sustainable Computing, 2024,9(3):422-432.
[14]THOMAS M, GUPTA M V, GOKUL RAJAN V, et al. Soft computing in computer network security protection system with machine learning based on level protection in thecloud environment[J]. Soft Computing, 2023:1-12.
[15]ZHANG T, LAM K Y, ZHAO J. Deep reinforcement learning based scheduling strategy for federated learning in sensor-cloud systems[J]. Future Generation Computer Systems, 2023(144):219-229.
[16]许强, 刘恩涛, 郑天龙, 等. 基于深度学习的配电网故障辨识系统研究[J]. 自动化技术与应用, 2025, 44(9):64-68.
[17]严梅. 基于深度学习算法的物联网资源信息自动采集方法[J]. 自动化技术与应用, 2025, 44(2):61-65.