Computer Science and Application 计算机科学与应用, 2025, 15(4), 162-176
Published Online April 2025 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2025.154089

Hans汉斯

基于MO-SAC算法的工业软件组件边缘集群多目标决策部署

徐沛沅[1,2]，王　涛[1,2]，程良伦[2,3]

[1]广东工业大学自动化学院，广东　广州
[2]广东省信息物理融合系统实验室，广东　广州
[3]广东工业大学计算机学院，广东　广州

收稿日期：2025年3月11日；录用日期：2025年4月11日；发布日期：2025年4月18日

摘　要

随着工业4.0的发展，工业物联网(IIoT)中软件组件的分布式部署已成为提高生产效率和灵活性的关键手段，通过在边缘端进行数据处理和分析，能够实现更低的延迟和更高的可靠性。尽管分布式部署带来了诸多优势，但仍面临任务需求复杂，节点资源受限，响应时间要求严格、集群负载不均衡以及环境动态变化等挑战，限制了其在实际工业环境中的应用。为了在任务响应时间和受限的资源分配中寻找最优解，并保证负载的均衡与稳定，本文提出了一种工业软件组件应用程序分布式边缘部署的系统架构，并且利用改进的MO-SAC算法对工作流进行分布式部署，引入温度系数，鼓励模型探索更多更广的部署策略，避免陷入局部最优。大量仿真结果表明MO-SAC可以在有效降低任务响应时间的同时最小化资源消耗并且使集群的负载更加均衡。

关键词

工业物联网，边缘集群，分布式部署，MO-SAC算法

Multi-Objective Decision-Making Deployment on Edge Clusters of Industrial Software Components Based on MO-SAC Algorithm

Peiyuan Xu[1,2], Tao Wang[1,2], Lianglun Cheng[2,3]

[1]School of Automation, Guangdong University of Technology, Guangzhou Guangdong
[2]Guangdong Laboratory of Cyber-Physical Systems (CPS), Guangzhou Guangdong
[3]School of Computer Science and Technology, Guangdong University of Technology, Guangzhou Guangdong

文章引用: 徐沛沅, 王涛, 程良伦. 基于 MO-SAC 算法的工业软件组件边缘集群多目标决策部署[J]. 计算机科学与应用, 2025, 15(4): 162-176. DOI: 10.12677/csa.2025.154089

Received: Mar. 11th, 2025; accepted: Apr. 11th, 2025; published: Apr. 18th, 2025

Abstract

With the development of Industry 4.0, the distributed deployment of software components in the Industrial Internet of Things (IIot) has become a key means to improve productivity and flexibility, enabling lower latency and higher reliability through data processing and analysis at the edge. Despite the many advantages of distributed deployment, it still faces challenges such as complex task requirements, limited node resources, strict response time requirements, unbalanced cluster loads, and dynamic changes in the environment, which limit its application in the actual industrial environment. In order to find the optimal solution in the task response time and limited resource allocation, and ensure the balance and stability of the load, this paper proposes a system architecture for distributed edge deployment of industrial software component applications, and uses the improved MO-SAC algorithm to distribute the workflow, introduces the temperature coefficient, encourages the model to explore more and wider deployment strategies, and avoids falling into the local optimum. A large number of simulation results show that MO-SAC can effectively reduce the task response time while minimizing resource consumption and making the load of the cluster more balanced.

Keywords

Industrial Internet of Things, Edge Cluster, Distributed Deployment, MO-SAC Algorithm

1. 引言

在近年来的工业物联网领域，工业软件组件凭借其模块化设计、可重用性和灵活性，极大地促进了制造业的数字化进程。在过去以云为中心的计算架构中，组件任务工作流部署到云，利用其强大算力对数据进行分析处理。但在面对新兴应用，如 VR/AR、自动驾驶、3d 同步定位和地图(SLAM)等需要更多低延迟处理需求资源时，传统的云计算不再具有优势[1]。

边缘计算作为云计算的补充，正逐步成为实现低延迟、高效率数据处理的新范式[2]。通过将计算能力下沉靠近数据源，不仅能够缩短数据传输路径，减少响应时间，还为工业软件组件任务的实时数据分析与决策提供了有力支撑[3] [4]。所以如何在异构的边缘环境中对工作流进行高效部署调度，同时能够适应动态环境，成为了当前研究的热点和难点。

工业应用程序在部署之前需要将其划分为不同的任务组件，并在异构边缘节点上编排和部署这些任务，这给研究带来了许多挑战。本文通过采用应用程序分区、节点资源抽象化以及深度强化学习等方法的集成对目标进行优化，将计算密集型或时间敏感型任务部署到合适节点，对不同硬件架构和能力的节点资源进行抽象化建模，以通用的指标体系来表达其计算、存储及通信能力，基于 SAC 算法对部署模型进行训练，平衡资源利用与任务时延[5]。

针对这些做法，我们提出了一套解决方案，通过预分析确定任务组件资源需求，采用 MO-SAC 算法合理化资源分配，优化任务部署策略，以平衡任务处理时间和资源利用率。除此之外，本文还构建了集

成 Prometheus 的资源监控系统，实时反馈边缘集群信息，确保集群稳定运行并进行动态调整，提升系统负均衡性衡与可靠性。具体的系统架构图如图 1 所示。

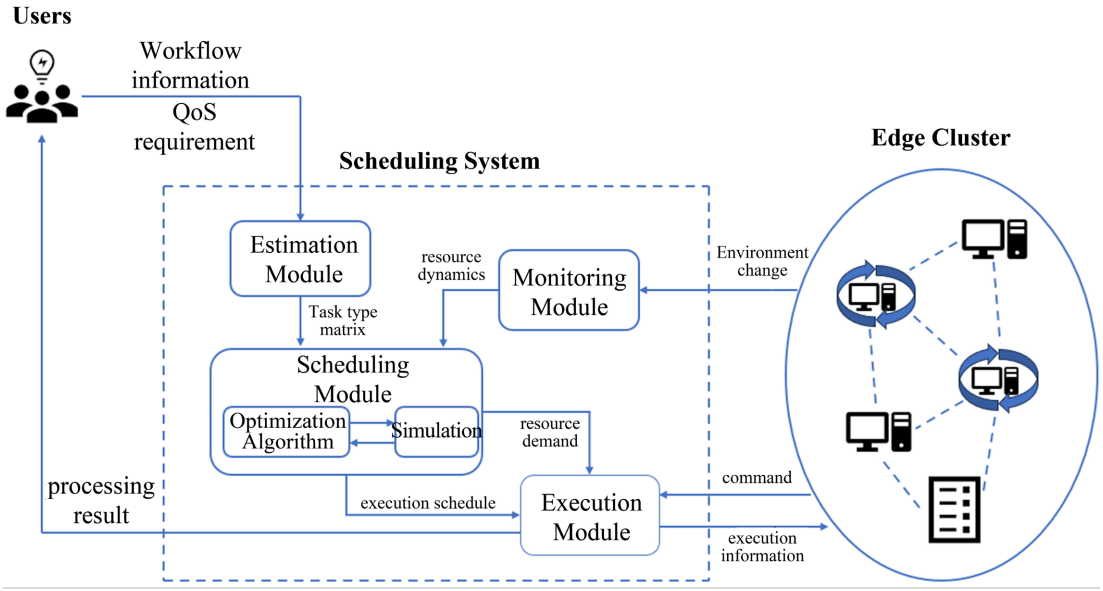

Figure 1. System framework
图 1. 系统框架

2. 相关工作

边缘计算的广泛应用为工业物联网开辟了新的可能性，但也带来了新的问题。由于边缘节点的计算和存储资源与云相比[6]相对有限，因此迫切需要有效利用边缘资源。传统调度策略常根据单一指标进行部署，当面临复杂的工业软件组件时无法做到兼顾，效率低下。如何高效部署任务组件以充分利用异构资源、处理组件间复杂的逻辑关系与实时性要求、在动态变化的环境条件下寻求任务处理时延和节点资源消耗的最优解衍生出两个任务部署的关键问题：部署模型和优化算法。

2.1. 部署模型

文献[7]提出了一种通用的在线调度模型，以最小化任务卸载到边缘设备时的任务响应时间。基于 Lyapunov 优化，文献[8]提出了一种基于通信模块和计算模块的排队模型与延迟模型用于公式推导，以最小化通信延迟和计算延迟。文献[9]提出了一种在 MEC 系统中联合协同任务感知部署的模型，目标是最小化平均协同任务完成时间，并在测试平台证明了提出的解决方案的有效性。这些方法基于理想的数学模型，虽然面向这些模型的算法可以取得良好的效果，但在节点负载与资源需求变化的动态环境中适用性依然不高。

2.2. 优化算法

文献[10]提出了一种利用强化学习解决异构分布式资源的工作流调度方案，在一定程度上减少了任务执行时间。文献[11]研究了多用户边缘计算环境中的工作流调度问题，并提出了一种基于深度 Q 网络(DQN)的多工作流调度方法，该方法能够处理边缘服务的时变性能。文献[12]提出了一种针对物联网边缘计算系统的资源分配策略，将资源分配问题抽象化为马尔可夫决策过程，利用改进的 DQN 来学习策略，能以更短的完成时间和更少的请求资源优于其他参考策略。

2.3. 总结

如上文所述，边缘计算上的任务部署从优化问题、模型再到优化算法逐步结合人工智能优化部署方案，可以看到有些工作关于资源分配的颗粒度较粗，优化目标单一，算法相对落后。针对这些不足和有待改进之处，本文提出了一套优化多目标决策的部署方案，通过预分析确定任务组件资源需求，采用 MO-SAC 算法合理化资源分配，优化部署策略，以平衡任务处理时间和资源利用。

3. 模型

3.1. 模型构建

在介绍任务模型之前，以图 2 的物块自动分拣应用程序为例，将其分解为若干个具有数据依赖和执行依赖关系的子任务：相机处理，物体检测，基于外观的实时映射、机械臂移动和放置。

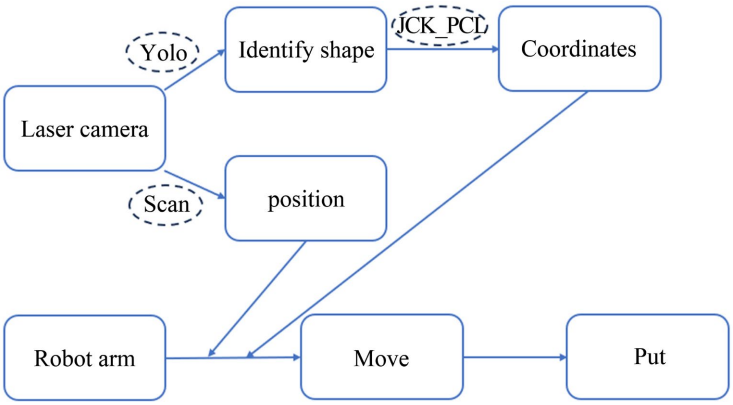

Figure 2. Automatic block sorting application based on 3D vision
图 2. 基于 3D 视觉的物块自动分拣应用程序

本文使用了两种异构边缘节点以不同配置对任务延迟进行剖析，得到如图 3 所示的任务预分析结果。

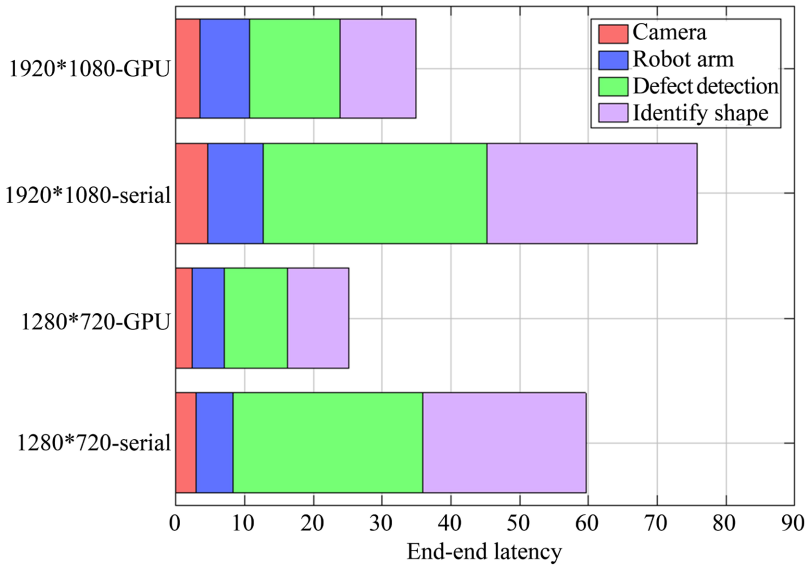

Figure 3. Delay of application tasks under different configurations
图 3. 应用程序任务在不同配置下的时延

从图中可以得出，像这种缺陷检测和形状识别等需要用到视频处理算法的任务是应用程序性能在延迟方面的主要瓶颈，随着资源配置的降低，延迟还会变得更高。

3.1.1. 任务模型

假设一个应用程序 A 可以在系统中被划分为 M 个任务 $\{T_1, T_2, \cdots, T_M\}$，系统在每个时间周期 T_{cycle} 中优化任务序列调度。每个任务 T_i 可以如式(1)这么表述：

$$T_i = \left(D_I^i, D_O^i, \tau_i, T_{type}\right) \tag{1}$$

其中 D_I^i 表示任务 T_i 从其直接前置任务接收的输入数据大小，D_O^i 表示任务 T_i 的输出数据的大小，T_{type} 表示任务预分析之后形成的任务类型，引入 T_{type} 是为了给分区之后的任务做标识，便于后续的任务类型区分，并且进一步定义了任务依赖图。

由多个依赖任务组成的应用程序通常可以用 DAG 图来表示，$G_{task} = \left(T_{task}, L_{task}\right)$，其中 $T_i \in T_{task}$，$\left(\forall 1 \le i \le N\right)$ 表示所有的分区任务，L_{task} 表示任务与任务之间的通信依赖或约束集合，每条有向边 $L_{i,j} = \left\langle T_i, T_j \right\rangle \in L_{task}$ 则代表任务 T_j 必须在任务 T_i 完成后才能执行。

3.1.2. 系统模型

假设由 K 个异构的边缘节点构成一个边缘集群 E，$e \in E$，边缘节点之间的连接可以用 $G_{con} = \left(E_{con}, L_{con}\right)$ 来表示，其中 $E_e \in E_{con}\left(e = 1, \cdots K\right)$，$E_{con}$ 表示所有边缘节点的集合，边缘节点之间通过有线链路连接，L_{con} 表示边缘节点之间的通信链路，$\left(E_e, E_f\right) \in L_{con}$ 则表示边缘节点 E_e 和 E_f 之间可以互相通信。

用 S 代表边缘节点 DAG 的混合调度策略，S 是一个元组。假设在一个时间周期 T_{cycle} 内，用 $S_{i,k}$ 代表任务 T_i 的调度策略，因为每个任务只能选择一个节点部署，所以有 $\sum_{k=0}^{K} S_{i,k} = 1,\ i \in M$。综合来讲，$S_{i,k}$ 可用式(2)表示：

$$S_{i,k} = \begin{cases} 1, & T_i \ is \ scheduled \ to \ the \ server \ k \\ 0, & otherwise. \end{cases} \tag{2}$$

3.2. 任务时延

编排策略 $S_{i,k}$ 第一个任务 T_0 的部署延迟如式(3)所示：

$$t_o\left(S_{0,k}\right) = \max\left(t_{network}, t_{resource}, t_{init}\right) \tag{3}$$

其中网络传输延迟 $t_{network}$ 可以用 $\dfrac{D_I^0}{B_{avg}}$ 表示，B_{avg} 代表平均可用带宽，而 $t_{resource}$ 则是获取执行任务需要的前置资源时间，t_{init} 是任务实例化、加载依赖项、环境配置等初始化步骤所需的时间。

任务执行时间与任务的复杂度和当前边缘节点 k 的计算能力有关。首先对任务的复杂度 T_i^{Com} 进行建模如式(4)所示：

$$T_i^{Com} = \frac{D_I^i \cdot T_i^{Cd} \cdot O^{T_i}}{P_k} \tag{4}$$

其中，T_i^{Cd} 是任务的计算密度，通常用 Flops/Byte(每字节浮点运算次数)来表示，O^{T_i} 表示任务的操作数，P_k 表示边缘节点 k 的计算能力。

通过引入计算资源的抽象特性，我们可以将边缘节点 k 的计算能力 P_k 描述如式(5)：

$$P_k : \left\{P_k^{Calculate}, P_k^{Storage}, B_k^{Network}\right\} \tag{5}$$

计算能力 $P_k^{Calculate}$ 用设备的 CPU/GPU 频率、核数和指令集等指标来衡量，存储能力 $P_k^{Storage}$ 用设备的内存容量和存储速度来衡量，节点带宽 $B_k^{Network}$ 可以用设备与云端或其他设备之间的网络连接速度和带宽来衡量。综合上述的计算资源和网络资源，对于任务 T_i，它的执行时间如式(6)：

$$t_e\left(S_{i,k}\right)=\frac{T_i^{Com}}{P_k^{\ 2}} \tag{6}$$

我们假设任务 T_i 的起始时间为 $t_{S_{i,k}}^{start}$，部署到服务器后的结束时间为 $t_{S_{i,k}}^{end}$。它们可以从首要任务递归计算到最终任务，如式(7)、(8)所示：

$$t_{S_{i,k}}^{start}=\max\left\{t_{S_{p,l}}^{end},\ T_p\in pre\left(T_i\right)\right\} \tag{7}$$

$$t_{S_{i,k}}^{end}=\begin{cases}t_e\left(S_{i,k}\right)+t_o\left(S_{i,k}\right),\ i=0\\ t_{S_{i,k}}^{start}+t_e\left(S_{i,k}\right),\ i\in\{1,2,\cdots,M-1\}\end{cases} \tag{8}$$

在 DAG 模型中，工作流任务应用程序的整个时间延迟是指从初始任务开始到最终任务完成的时间段。在所有任务被映射并部署到不同边缘节点后，完成一个工作流任务的时间延迟如式(9)：

$$t_{total}=\max\left\{t_{S_{i,k}}^{end},\ i\in M\right\} \tag{9}$$

3.3. 资源消耗

假设 C_A 代表处理应用程序 A 中所有任务的总成本，而 $C_{i,k}$ 表示在边缘节点 k 上处理任务 T_i 的成本。我们将边缘节点的可用资源分为 CPU/GPU 资源和内存资源两种，在处理每个任务的过程当中，都会对这些资源造成不同程度的消耗。

3.3.1. CPU/GPU

假设边缘节点 k 的 CPU 利用率为 $ur_{i,k}$，那么节点 k 处理任务 T_i 所消耗的 CPU 资源如式(10)所示：

$$C_{i,k}^{cpu}=ur_{i,k}\cdot cr_k^{cpu}\cdot core_k \tag{10}$$

其中，cr_k^{cpu} 表示边缘节点 k 上的 CPU 的时钟频率，$core_k$ 指边缘节点 k 上的 CPU 的核数。所以，边缘集群系统中的所有 CPU 资源消耗可以描述为如式(11)所示：

$$C_{all}^{cpu}=\sum_{i}^{M}\sum_{k}^{K}C_{i,k}^{cpu}\quad i\in M,k\in K \tag{11}$$

同理，边缘节点 k 上任务 T_i 的运行时间为 $t_e\left(S_{i,k}\right)$，节点 k 处理任务 T_i 所消耗的 GPU 的资源 $C_{i,k}^{gpu}$ 可以用式(12)表述：

$$C_{i,k}^{gpu}=\frac{E_k\cdot\left(wr_k^{gpu}\right)^2}{cr_k^{gpu}} \tag{12}$$

其中，E_k 代表边缘节点与硬件工艺相关的常数，wr_k^{gpu} 表示节点上的 GPU 的工作频率，而 cr_k^{gpu} 则是指节点上的 GPU 的时钟频率。所以，边缘集群系统中的所有 GPU 资源消耗可以描述为如式(13)所示：

$$C_{all}^{gpu}=\sum_{i}^{M}\sum_{k}^{K}C_{i,k}^{gpu}\quad i\in M,k\in K \tag{13}$$

3.3.2. 内存

将任务 T_i 部署到边缘节点 k 上以后，节点 k 所消耗的内存资源 Ω_k 可以采用在时间序列上的内存占用

积分来反映一段时间周期 T_{cycle} 内系统的资源消耗情况，如式(14)所示：

$$\Omega_k = \int_0^{T_{cycle}} \Omega(t) + \Omega_k^{fixed},\ t \in \left[0, T_{cycle}\right] \tag{14}$$

其中 $\Omega(t)$ 代表系统进程中内存占用量随时间变化的函数，Ω_k^{fixed} 代表边缘节点运行时的固定开销，所以整个边缘集群系统的资源消耗 Ω_{all} 可以表示为式(15)：

$$\Omega_{all} = \sum_{k}^{K} \Omega_k,\ k \in K \tag{15}$$

根据上述时间延迟模型和资源消耗模型要求，我们可以将工作流任务部署问题表述为多目标约束优化问题如式(16)所示：

$$\begin{aligned} &\min t_A = t_{total} \\ &\min rc_A = \sum_{k=1}^{K} C_{all}^{cpu} + C_{all}^{gpu} + \Omega_{all} \\ &s.t.:\ G_{task},\ G_{con}\ \ are\ satisfied \end{aligned} \tag{16}$$

其中，rc_A 是处理应用程序 A 的总成本，由每个边缘节点的资源消耗构成，包括 CPU/GPU 和内存资源的使用。t_A 是处理应用程序 A 的总延迟，以上所有优化目标的数据均在一个时间周期 T_{cycle} 内完成。

4. 方法

4.1. 任务资源估计

本文在实际部署之前提出了一个预分析阶段，将边缘节点 Jetson nano 的 CPU 利用率设置为 10%到 90%，拟合测量值的置信带，提供区间估计并预测不确定性范围。在该模型中，我们将处理器的 CPU 资源利用率和任务作为输入，并给出任务的处理时间估计，得到结果如图 4 所示。

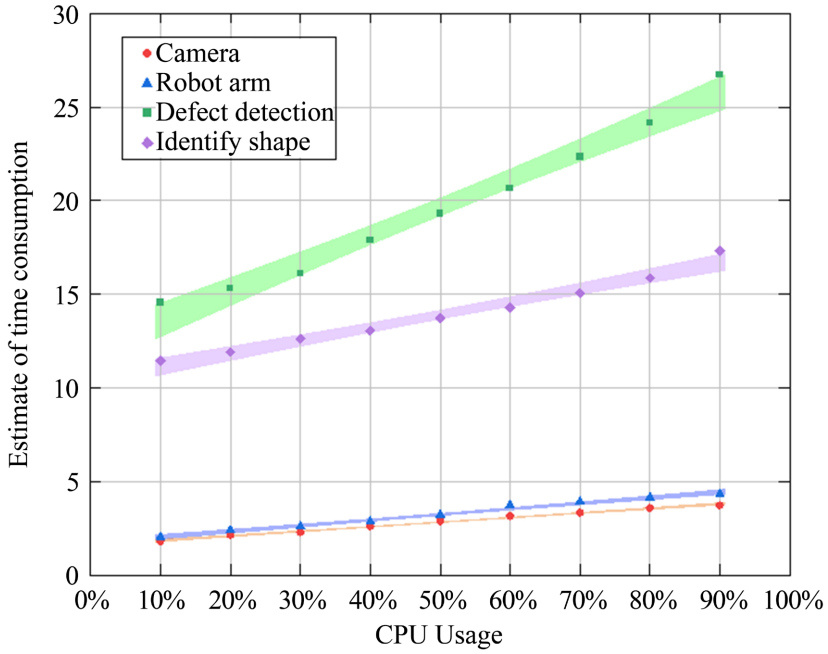

Figure 4. Task processing time estimates
图 4. 任务处理时间预估计

4.2. 集群资源监控

本文利用 Prometheus 的优势和性能结合设计的集成控制组件，实现了集群资源监控模块采集、存储和查询各节点的性能指标、任务状态和可用资源等功能。通过对边缘集群的资源监测，有效提高系统容错能力，迅速响应系统故障，降低资源消耗。

4.2.1. 监控采集

数据汇总模块 SD Client 从边缘集群系统中采集硬件资源信息 HIS (Hardware Resource Information)和软件资源信息 SSI (Software Resource Information)后，以 K/V 的形式发送给集成控制件，之后再经集成控制件的转换模块 FC 将数据转换为时间序列数据形式。集成控制件还会暴露一个 HTTP 接口供查询，Prometheus 的数据采集模块 DC 通过基于 HTTP 的 pull 方式来周期性的拉取数据。集成控制件和集群系统以及 Prometheus 的数据流向接口如图 5 所示。

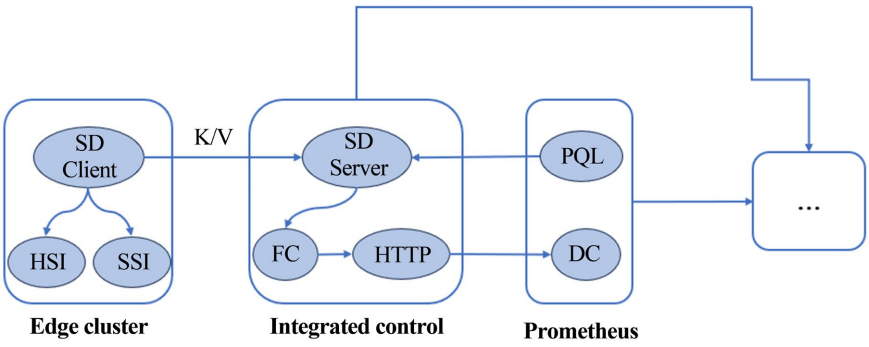

Figure 5. System data flow direction
图 5. 系统数据流向

4.2.2. 信息交互

本文定义了传输协议 MWP (Monitoring and Warning Protocol)来实现交互功能。集成控制件通过将获取的数据信息以 MWP 协议封装后传输给 Grafana，用户可在交互界面上与系统进行交互。协议格式如图 6 所示。

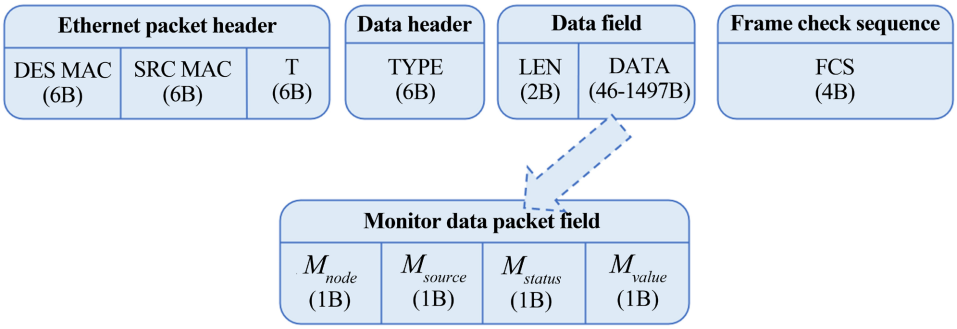

Figure 6. System data flow direction
图 6. 协议格式

MWP 协议以四元组形式 M (M_{node}，M_{source}，M_{status}，M_{value})表示，Data 字段核心内容为主要传输数据。其中：M_{node} 代表节点的 IP 地址；M_{source} 代表节点的资源类别，分硬件资源和软件资源；M_{status} 代表节点状态，分为正常状态和故障状态；M_{value} 代表监控指标的具体值。

4.3. 任务编排调度

强化学习中的 SAC 算法结合了策略梯度和 Q-learning 的方法，并通过最大化策略的熵来鼓励探索。与传统的 Actor-Critic 算法相比，SAC 算法在处理连续和离散动作空间以及稳定学习方面具有优势[13]。因此，本文开发了一种基于 SAC 的 MO-SAC 算法，以实现最优工作流调度结果。

4.3.1. 问题转化

在本文中，时间以固定的时隙持续时间划分。整个任务部署过程，从发出请求到获得最终结果，持续 τ 个状态。t 表示时刻，$t\in\{0,\ 1,\cdots,\ \tau-1\}$。

第一，关于状态空间。对任意时刻 t 所处的系统状态描述如下：

$$s_t=\left\{\chi_{T_i,k}(t),\ \delta_{T_i}(t),\ \omega_k\right\} \tag{17}$$

$$\chi_{T_i,k}(t)=\begin{bmatrix} B_{T_i,1}(t),\ B_{T_i,2}(t),\cdots,\ B_{T_i,K}(t) \\ P_1(t),\ P_2(t),\cdots,\ P_K(t) \end{bmatrix} \tag{18}$$

其中 $\chi_{T_i,k}(t)$ 是一个 $2\times K$ 的矩阵，$B_{T_i,K}(t)$ 代表 t 时刻接收任务 T_i 的边缘节点 k 的网络带宽，$P_K(t)$ 是指边缘节点 k 的计算能力，$\delta_{T_i}(t)$ 包含工业物联网应用程序的 DAG 内要处理的当前任务 T_i 的信息，$\delta_{T_i}(t)=\left(D_I^i,\ wl_{i,k},\ D_O^i\right)$，$\omega_k(t)=\left[\omega_1(t),\ \omega_2(t),\cdots,\ \omega_K(t)\right]$ 代表每一个边缘节点 k 的系统配置。

第二，关于动作空间。本文设计一个混合动作空间来描述这种决策结构[14]，将动作空间定义为一个高维向量，动作的选择决定了两个相邻系统状态之间的转移规则，我们将任务 T_i 的编排决策定义为代理做出的动作如式(19)所示：

$$a_{T_i}(t)=\left\{a_{T_{i,0}}(t),\ a_{T_{i,1}}(t),\ a_{T_{i,2}}(t),\cdots,\ a_{T_{i,N}}(t)\right\} \tag{19}$$

首先，为节点选择这个离散动作设计一个连续的概率分布向量 $\mathbf{a}_{node}$，可以通过一个原始向量 $\mathbf{z}\in\mathbb{R}^K$ 计算得到，其中 z_k 表示原始的未归一化得分，K 是边缘节点的数量。用 *soft* max 函数将其转换为概率分布：

$$p\left(a_{node,k}\right)=\frac{\exp\left(z_k\right)}{\sum_{k=1}^{K}\exp\left(z_k\right)} \tag{20}$$

其中，$p\left(a_{node,k}\right)$ 表示选择第 k 个节点的概率，且满足 $\sum_{k=1}^{K}a_{node,k}=1$。

对于选定的节点，假设每种资源的比例分配由三维向量 $\mathbf{a}_{res}=\left[a_{res,\lambda}\right]$（$\lambda=1,\ 2,\ 3$ 分别对应 CPU/GPU 和内存)。

为了确保资源分配总和不超过 1，考虑一个原始向量 $\mathbf{b}=\left(b_1,\ b_2,\ b_3\right)\in\mathbb{R}^3$，通过软约束方法来控制总和，在强化学习环境中比较常见的一种是直接在策略网络输出后通过后处理来保证位于预设的区间内，如式(21)所示：

$$a_{res,\lambda}=clip\left(b_\lambda,\ 0,\ 1\right) \tag{21}$$

综合以上两部分，最终的混合动作空间 $\mathbf{a}$ 可以用式(22)表示：

$$\mathbf{a}=\left(\mathbf{a}_{node},\ \mathbf{a}_{res}\right)=\left(a_{node,k},\ a_{res,1},\ a_{res,2},\ a_{res,3}\right) \tag{22}$$

其中，$\mathbf{a}_{node}$ 是离散动作空间，$\mathbf{a}_{res}$ 是连续动作空间，$\mathbf{z}$ 和 $\mathbf{b}$ 由策略网络直接输出，通过后处理转换为有效动作空间中的动作。

第三，关于奖励函数。本文设计了一种综合考虑了处理执行时间、资源消耗与负载均衡、任务优先级等三个性能指标实现系统优化奖励函数。

执行时间奖励函数时采用了饱和函数，用来反映随着任务的执行时间减少，边际收益递减的情况，如式(23)所示：

$$R_{delay} = -\alpha \cdot \left(1 - e^{-\theta \cdot t_e\left(S_{i,k}\right)}\right) \tag{23}$$

上式中，θ是一个正参数，代表时间收益的饱和程度。当$t_e\left(S_{i,k}\right)$较小时，收益增加显著；随着$t_e\left(S_{i,k}\right)$减小，收益增加速度放缓。

本文通过引入各节点资源使用率的标准差σ_{res}来直接反映负载均衡的状态，同时考虑单个节点的极值惩罚项来鼓励更加均匀且高效的资源分配，如式(24)：

$$R_{balance} = -\beta \cdot \left(\sigma_{res} + \xi \cdot \max_k \left(wl_k - wl_{avg}\right)\right) \tag{24}$$

其中，β是负权重，ξ是调整参数，控制极值惩罚的强度，wl_k是边缘节点k的工作负载，wl_{avg}是边缘集群系统的平均工作负载。

为了更精确地反映优先级和等待时间的关系，本文使用 sigmoid 函数，以体现优先级任务及时处理的紧迫性，如式(25)：

$$R_{priority} = \sum_i \kappa \cdot I_i \cdot \left(1 - \frac{1}{1 + e^{-\zeta \cdot (W_i - W_{th})}}\right) \tag{25}$$

其中，κ是调整参数，ζ是指数衰减因子，I_i是任务T_i的优先级，W_i是任务的等待时间，W_{th}是一个阈值，超过这个阈值的等待时间，高优先级任务的奖励迅速下降。

4.3.2. 策略函数

在 MO-SAC 算法中，我们为两个动作价值函数Q(参数分别为ω_1和ω_2)和一个策略函数π(参数为θ)建模[15]。基于 Double DQN 的思想，SAC 使用两个Q网络，但每次用Q网络时会挑选一个Q值小的网络，从而缓解Q值过高估计的问题。任意一个函数Q的损失函数如式(26)：

$$\begin{aligned} L_Q(\omega) &= \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim R}\left[\frac{1}{2}\left(Q_\omega(s_t,a_t) - \left(r_t + \gamma V_{\omega^-}(s_{t+1})\right)\right)^2\right] \\ &= \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim R, a_{t+1}\sim\pi_\theta(\cdot|s_{t+1})}\left[\frac{1}{2}\left(Q_\omega(s_t,a_t) - \left(r_t + \gamma\left(\min_{j=1,2} Q_{\omega_j^-}(s_{t+1},a_{t+1}) - \alpha \log \pi(a_{t+1}\mid s_{t+1})\right)\right)\right)^2\right] \end{aligned} \tag{26}$$

其中，R是策略过去收集的数据。为了让训练更加稳定这里使用了目标Q网络Q_{ω^-}，同样是两个目标Q网络，与两个Q网络一一对应。

策略的损失函数由KL散度得到，化简后为：

$$L_\pi(\theta) = \mathbb{E}_{s_t\sim R, a_t\sim\pi_\theta}\left[\alpha \log\left(\pi_\theta(a_t\mid s_t)\right) - Q_\omega(s_t,a_t)\right] \tag{27}$$

对于连续动作空间的环境，SAC 算法的策略输出高斯分布的均值和标准差，但是根据高斯分布来采样动作的过程是不可导的。所以需要用到重参数化技巧，先从一个单位高斯分布N采样，再把采样值乘以标准差后加上均值。这样就可以认为是从策略高斯分布采样，并且这样对于策略函数是可导的[16]。我们将其表示为$a_t = f_\theta(\varepsilon_t; s_t)$，其中$\varepsilon_t$是一个噪声随机变量。同时考虑到两个函数$Q$，重写策略的损失函数如下：

$$L_{\pi}(\theta)=\mathbb{E}_{s_t\sim R,\varepsilon_t\sim N}\left[\alpha\log\left(\pi_{\theta}\left(f_{\theta}\left(\varepsilon_t;s_t\right)|s_t\right)\right)-\min_{j=1,2}Q_{\omega_j}\left(s_t,f_{\theta}\left(\varepsilon_t;s_t\right)\right)\right] \tag{28}$$

在初始化阶段，MO-SAC 定义了策略网络(参数为 θ)以及一对 Q 价值网络(参数分别为 ω_1 和 ω_2)，工作流应用程序中的每个任务根据策略 $\pi_{\theta}(s_t)$ 选择混合动作 $\mathbf{a}(t)$，动作价值函数 $Q_{\omega_j}(s_t,\mathbf{a}_t)$ 衡量状态 s 下当前动作 $\mathbf{a}(t)$ 的质量。训练循环阶段始于采样环境交互数据，利用策略 $\pi_{\theta}(s_t)$ 在环境中执行动作并收集当前状态、动作、奖励函数以及下一个状态作为一个四元组 $\{\mathbf{s}_t,\ \mathbf{a}_t,\ \mathbf{r}_t,\ \mathbf{s}_{t+1}\}$ 存放到经验回放池。这些样本随后被用于策略更新及价值函数的学习。

5. 实验

5.1. 实验设置

在本节中，我们用数值结果验证了提出的 MO-SAC 算法部署工作流的性能。MO-SAC 的神经网络是由一个输入层($|S|$神经元)、两个隐藏层和一个输出层($|A|$神经元)组成的。$|S|$和$|A|$分别是状态和动作的维度空间。激活函数采用 Sigmoid 函数。该奖励效用在一个 Episode 中累积，在进行 2000 个学习片段后进行收集，将 MO-SAC 算法的学习率和衰减率分别设置为 0.0001 和 0.5。温度系数 α 参照采用自动确定的方法。采用自适应贪心算法，避免在模型有足够的学习经验之前陷入次优策略。贪心因子最初设置为 0.9，然后呈指数衰减，直到达到 0.05。

本实验数据通过实验室自主研发的工业软件实训平台设备采集。平台主要由 UR 机械臂、机械臂末端执行器(主要为不同类型夹爪)、Hikvision 视觉相机以及针对不同工业应用场景的专用设备组成。实验于 2024 年 10 月至 11 月在信息物理融合系统实验室进行，共从四个边缘节点组成的边缘集群中采集节点资源利用率、任务处理时延等原始数据 500 组，剔除因网络异常、设备故障等导致的异常值后，保留有效数据 447 组。数据处理采用 python3.8 中的 pandas 库进行缺失值插补和归一化。

5.2. 训练表现和收敛性

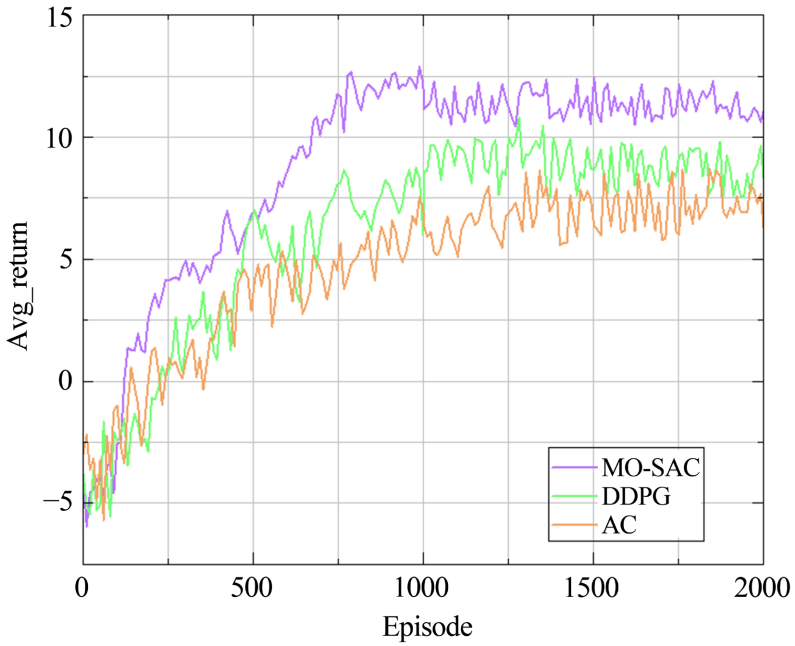

Figure 7. Convergence performance of different algorithms
图 7. 不同算法的收敛表现

为了验证算法的有效性和泛化能力，我们首先进行了不同初始状态的实验。图 7 比较了三种算法的训练集获得的平均累积收益。在训练初期，策略更新很快，收益在每次迭代之间的波动较大。随着时间推移，收益逐渐收敛到稳定的值。MO-SAC 在第 920 个 Episode 左右收敛到 11，平均累计收益最高。

我们还研究了梯度下降训练的学习因子对系统性能的影响，如图 8 所示，随着学习率的提高，收敛速度更快。学习率低导致收敛速度慢，使代理无法从经验中学习。然而，快速收敛会导致较大的波动，并可能导致代理陷入局部最优而错过更好的策略。因此，在设置学习率时，不宜过高或过低。

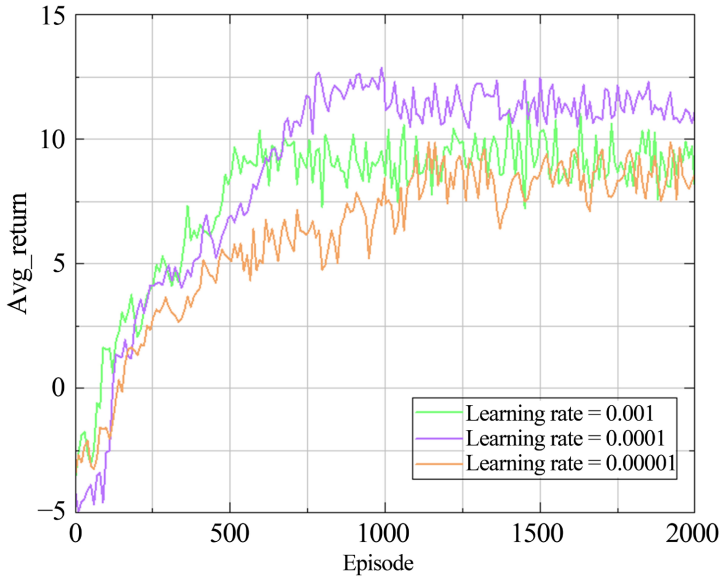

Figure 8. Convergence performance of MO-SAC at different learning rates
图 8. MO-SAC 在不同学习率下的收敛表现

5.3. 集群负载均衡

本文比较了三种算法(DDPG、AC、MO-SAC)在边缘集群中的 CPU 和 GPU 负载情况。图 9 和图 10 通过采集每隔一定时间的集群资源负载信息展示 CPU 和 GPU 利用率的离散程度，MO-SAC 算法箱体间距最小，在整个实验期间的 CPU 和 GPU 利用率最高且最为均衡。

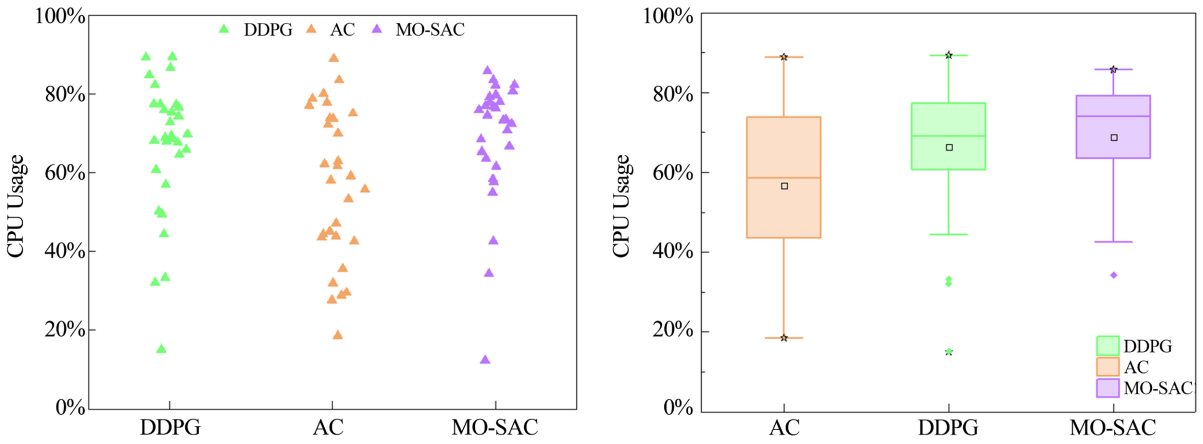

Figure 9. CPU load information
图 9. CPU 负载信息

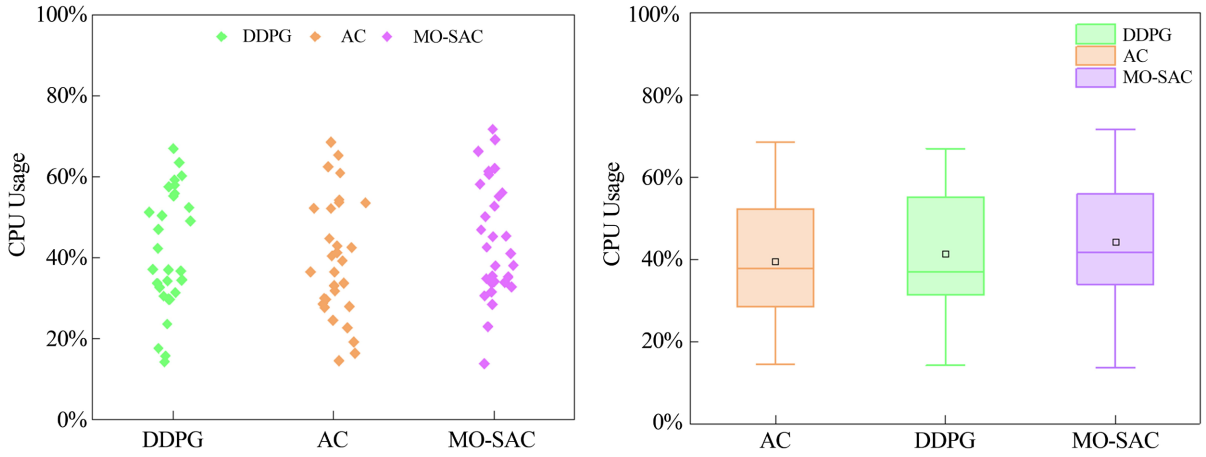

Figure 10. GPU load information
图 10. GPU 负载信息

为了进一步量化负载的均衡性，我们计算了每种算法在 CPU 和 GPU 上的负载标准差，如表 1 所示。从表中可以看出，MO-SAC 算法在 CPU 和 GPU 上的负载标准差均低于 DDPG 和 AC 算法。这表明其能够更有效地分配计算资源，从而提高负载均衡，这得益于其高效的策略更新机制和资源分配策略。

Table 1. Standard deviation of three algorithms
表 1. 三种算法的标准差

Standard Deviation	CPU Usage	GPU Usage
DDPG	0.1511	0.1424
AC	0.1866	0.1535
MO-SAC	0.1237	0.1376

5.4. 多维度评估

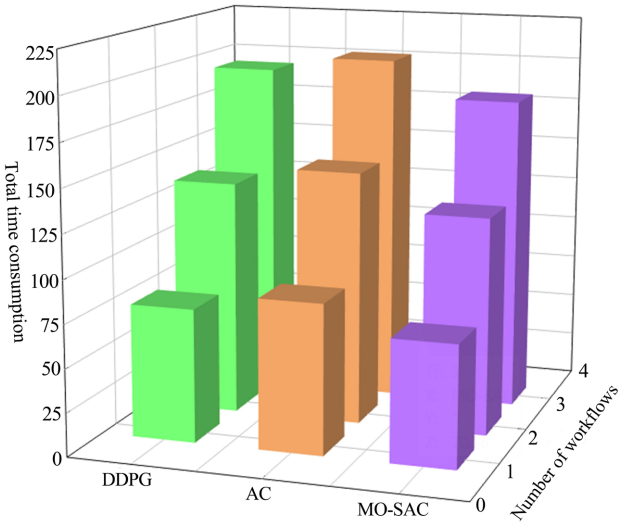

Figure 11. Total time consumption in multiple dimensions
图 11. 多维度下总时间消耗

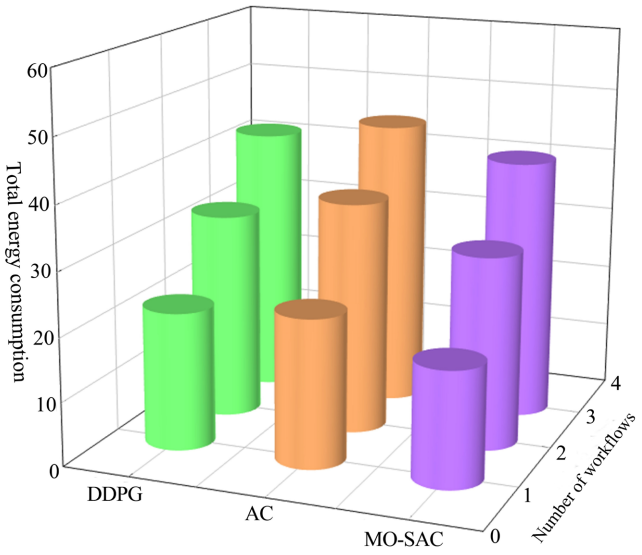

Figure 12. Total resource consumption in multiple dimensions
图 12. 多维度下总资源消耗

本文比较了三种算法在不同工作流数量下的总处理时间和总资源消耗。从图 11 可以看出，MO-SAC 算法在所有工作流数量下的总处理时间均显著低于 DDPG 和 AC 算法。类似地，图 12 显示其在总资源消耗方面也表现出色，其总资源消耗始终低于其他两种算法。当工作流数量为 1 时，任务处理时间比 DDPG 和 AC 算法分别降低了 8.68%和 18.65%，总资源能耗分别降低了 15.96%和 21.93%。实验结果表明，MO-SAC 算法在处理时间和资源消耗方面均优于其他两种，尤其是在资源受限的环境中，SAC 算法能够提供更高的处理效率和资源利用效率。

6. 结语

本文研究了面向工业软件组件在边缘集群中的工作流任务部署调度问题。我们通过对边缘集群、工作流任务以及优化目标等进行数学建模，将任务部署调度问题抽象为多目标优化问题。接着进一步研究了基于 MO-SAC 的算法来寻找解决方案，利用深度强化学习进行离线训练，不断更新部署策略。最后，我们通过设置的对比实验来验证所提出方案的有效性。

我们的方法存在一些缺点。针对工作流任务部署性能的影响因素，除了直观的资源供给、任务量和算法效率，还能考虑节点动态性与可靠性、工作流复杂程度以及数据传输等因素。在后续工作中，我们将探索强化学习驱动的动态调度，结合边缘节点的实时状态预测，构建端到端智能决策模型。

基金项目

国家自然科学基金项目(U20A600004)：广东省信息物理融合系统实验室。

参考文献

[1] Urrea, C. and Benítez, D. (2021) Software-Defined Networking Solutions, Architecture and Controllers for the Industrial Internet of Things: A Review. *Sensors*, **21**, Article 6585. https://doi.org/10.3390/s21196585

[2] Bourechak, A., Zedadra, O., Kouahla, M.N., Guerrieri, A., Seridi, H. and Fortino, G. (2023) At the Confluence of Artificial Intelligence and Edge Computing in IoT-Based Applications: A Review and New Perspectives. *Sensors*, **23**, Article 1639. https://doi.org/10.3390/s23031639

[3] Hassan, N., Gillani, S., Ahmed, E., Yaqoob, I. and Imran, M. (2018) The Role of Edge Computing in Internet of Things. *IEEE Communications Magazine*, **56**, 110-115. https://doi.org/10.1109/mcom.2018.1700906

[4] Chen, B., Wan, J., Celesti, A., Li, D., Abbas, H. and Zhang, Q. (2018) Edge Computing in IoT-Based Manufacturing. *IEEE Communications Magazine*, **56**, 103-109. https://doi.org/10.1109/mcom.2018.1701231

[5] Qi, Q., Zhang, L., Wang, J., Sun, H., Zhuang, Z., Liao, J., *et al*. (2020) Scalable Parallel Task Scheduling for Autonomous Driving Using Multi-Task Deep Reinforcement Learning. *IEEE Transactions on Vehicular Technology*, **69**, 13861-13874. https://doi.org/10.1109/tvt.2020.3029864

[6] Walia, G.K., Kumar, M. and Gill, S.S. (2024) AI-Empowered Fog/edge Resource Management for IoT Applications: A Comprehensive Review, Research Challenges, and Future Perspectives. *IEEE Communications Surveys & Tutorials*, **26**, 619-669. https://doi.org/10.1109/comst.2023.3338015

[7] Tan, H., Han, Z., Li, X. and Lau, F.C.M. (2017). Online Job Dispatching and Scheduling in Edge-Clouds. *IEEE INFOCOM* 2017-*IEEE Conference on Computer Communications*, Atlanta, 1-4 May 2017, 1-9. https://doi.org/10.1109/infocom.2017.8057116

[8] Zhang, Y., Du, P., Wang, J., Ba, T., Ding, R. and Xin, N. (2019) Resource Scheduling for Delay Minimization in Multi-Server Cellular Edge Computing Systems. *IEEE Access*, **7**, 86265-86273. https://doi.org/10.1109/access.2019.2924032

[9] Chiang, Y., Zhang, T. and Ji, Y. (2019) Joint Cotask-Aware Offloading and Scheduling in Mobile Edge Computing Systems. *IEEE Access*, **7**, 105008-105018. https://doi.org/10.1109/access.2019.2931336

[10] Orhean, A.I., Pop, F. and Raicu, I. (2018) New Scheduling Approach Using Reinforcement Learning for Heterogeneous Distributed Systems. *Journal of Parallel and Distributed Computing*, **117**, 292-302. https://doi.org/10.1016/j.jpdc.2017.05.001

[11] Liu, H., Ma, Y., Chen, P., Xia, Y., Ma, Y., Zheng, W., *et al*. (2020) Scheduling Multi-Workflows over Edge Computing Resources with Time-Varying Performance, a Novel Probability-Mass Function and DQN-Based Approach. In: *Lecture Notes in Computer Science*, Springer, 197-209. https://doi.org/10.1007/978-3-030-59618-7_13

[12] Xiong, X., Zheng, K., Lei, L. and Hou, L. (2020) Resource Allocation Based on Deep Reinforcement Learning in IoT Edge Computing. *IEEE Journal on Selected Areas in Communications*, **38**, 1133-1146. https://doi.org/10.1109/jsac.2020.2986615

[13] Haarnoja, T., Zhou, A., Hartikainen, K., *et al*. (2018) Soft Actor-Critic Algorithms and Applications.

[14] Fan, Z., Su, R., Zhang, W. and Yu, Y. (2019) Hybrid Actor-Critic Reinforcement Learning in Parameterized Action Space.

[15] Haarnoja, T., Zhou, A., Abbeel, P., *et al*. (2018) Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. 2018 *International Conference on Machine Learning*, Stockholm, 10-15 July 2018, 1861-1870.

[16] Figurnov, M., Mohamed, S. and Mnih, A. (2018) Implicit Reparameterization Gradients. *Advances in Neural Information Processing Systems*, 31.