# 基于大模型辅助的云边协同工作流调度算法

黎广镕[1],李广军[1],尚 晶[2],吴文泰[1],王泽平[1],龙赛琴[1*]
(1. 暨南大学信息科学技术学院,广东广州 510632;2. 中移动信息技术有限公司,北京 100033)

**摘 要:** 工作流在云边协同环境中执行可以减少云与终端设备之间的数据传输时延. 由于云计算节点、边缘设备在计算能力、存储资源及通信延迟等方面存在显著差异,加之边缘服务器计算资源受负载压力、性能退化等因素影响具有动态性,同时工作流应用内部复杂的拓扑依赖关系进一步增加了调度约束条件,使得该场景下的工作流调度问题被证明为NP-hard问题. 针对上述问题,本文提出了基于大模型辅助的云边协同工作流调度算法(Large Language Model-Assisted Cloud-Edge Collaborative Workflow Scheduling Algorithm,LAWS). 该算法通过知识图谱结构化表征推理过程的思维链(Chain-of-Thought,CoT),将调度问题分解成多个子问题,并提取出子知识图谱作为子问题的思维链引导大模型协同推理调度决策. 实验结果表明,与传统算法相比,该算法使得工作流执行时延降低3%~83%,计算能耗降低2.4%~66.0%.

**关键词:** 工作流调度;云边协同;大模型;知识图谱;思维链;问题分解
**基金项目:** 国家自然科学基金(No.U23B2027);广东基础与应用基础研究基金(No.2024A1515010214)
**中图分类号:** TP311.1 **文献标识码:** A **文章编号:** 0372-2112(2025)09-3060-18
**电子学报 URL:**http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250494

# Large Language Model-Assisted Cloud-Edge Collaborative Workflow Scheduling Algorithm

LI Guang-rong[1], LI Guang-jun[1], SHANG Jing[2], WU Wen-tai[1], WANG Ze-ping[1], LONG Sai-qin[1*]
(1. *College of Information Science and Technology*, *Jinan University*, *Guangzhou*, *Guangdong* 510632, *China*;
2. *China Mobile Information Technology Co.*, *Ltd.*, *Beijing* 100033, *China*)

**Abstract:** Executing workflows in cloud-edge collaborative environments can reduce data transmission latency between the cloud and terminal devices. Significant differences exist between cloud computing nodes and edge devices in terms of computational capability, storage resources, and communication latency. Furthermore, the computational resources of edge servers exhibit dynamicity due to factors like workload pressure and performance degradation. The complex topological dependencies within workflow applications introduce additional scheduling constraints. These combined factors render the workflow scheduling problem in this context NP-hard. To address these challenges, this paper proposes large language model-assisted cloud-edge collaborative workflow scheduling algorithm (LAWS). The algorithm employs a knowledge graph to structurally represent the chain-of-thought (CoT) reasoning process. It decomposes the scheduling problem into multiple sub-problems and extracts sub-knowledge graphs to serve as chain-of-thought guides for the large model, facilitating collaborative reasoning for scheduling decisions. Experimental results demonstrate that compared with traditional algorithms, the proposed algorithm achieves a reduction in workflow execution latency of 3% to 83% and a decrease in computational energy consumption of 2.4% to 66.0%.

**Key words:** workflow scheduling; cloud-edge collaboration; large language models; knowledge graph; chain-of-thought; problem decomposition
**Foundation Item(s):** National Natural Science Foundation of China (No.U23B2027); Guangdong Basic and Applied Basic Research Foundation (No.2024A1515010214)

收稿日期:2025-06-08;录用日期:2025-09-16;责任编辑:覃怀银
*通讯作者:龙赛琴

## 1 引言

云计算平台凭借其集约化资源池提供弹性可扩展的计算资源,但其面临网络拥塞、高延迟的问题,不适用于实时性任务. 边缘计算节点相较于云计算平台具有更贴近端设备的空间分布特征,在降低服务时延、优化带宽利用率等方面表现出显著优势[1]. 由于边缘网络拓扑具有动态性和复杂性,加之边缘服务器存在固有的异构性与资源动态性等特征,同时受限于计算资源有限和处理能力不足,导致边缘计算无法独立满足高负载请求任务的执行需求. 云边协同架构通过深度融合二者的优势,减少端设备与云计算平台的数据传输,同时发挥云平台的计算能力,为多种类型任务执行提供解决方案[2].

工作流作为云边协同环境下常见的任务[3],现有研究通常使用有向无环图(Directed Acyclic Graph,DAG)表示工作流[4]. 有向无环图中的节点定义了一组原子计算任务,有向边则定义了任务间的依赖约束:当且仅当某子任务的所有前驱节点执行完成后,该子任务方可被执行.

在云边协同环境下,对子任务间具有复杂依赖关系的工作流动态调度优化是一项巨大挑战. 由于边缘计算节点的资源动态性及计算性能波动性,工作流执行过程易受节点状态扰动影响,导致工作流的执行时延显著增加. 此外,云边负载的动态失衡可能引发资源竞争,进一步加剧执行时延和计算能耗的增加.

针对上述挑战,本文提出一种基于大模型辅助的云边协同工作流调度算法(Large Language Model-Assisted Cloud-Edge Collaborative Workflow Scheduling Algorithm,LAWS),主要贡献如下:

(1)创新性地引入大模型对调度问题进行分解,提出更新调度策略和在线任务解耦异步执行框架.

(2)提出使用知识图谱表示模型推理过程的思维链(Chain-of-Thought,CoT),并使用子图匹配技术实现子问题思维链的提取.

(3)实验结果表明,LAWS相较于对比算法,使得工作流执行时延降低3%~83%,计算能耗降低2.4%~66.0%.

## 2 相关工作

### 2.1 工作流调度研究

DAG形式的工作流调度问题已被证明属于NP-hard问题,无法在多项式时间内求得精确解. 当前研究主要采用启发式调度算法、进化优化算法和强化学习方法来解决此类调度问题,各方法具有不同的技术特点和应用局限.

启发式调度算法是根据已有的经验获得调度方案. Kubernetes的默认调度器[5],其采用两阶段调度框架:在过滤阶段基于任务约束条件(如资源需求、节点标签等)筛选候选节点;在打分阶段综合负载均衡、镜像亲和性等指标进行加权评分,最终选择得分最高的节点完成调度. 传统启发式调度算法通常基于DAG拓扑特性,考虑各个子任务对整个工作流的影响,根据不同的场景设计相适应的算法. 例如,Tang等人[6]提出贪心算法降低工作流在云环境下的执行成本;Sun等人[7]针对云计算中截止时间约束的工作流调度问题提出启发式算法,旨在最小化总成本和总空闲率;Shin等人[8]根据任务数量采用两种不同的调度策略,任务量少时优先调度后继较多的任务,任务量多时,优先调度关键路径上的任务,并动态分配资源和规划任务并行度,以提高资源利用率和降低执行时延. 同时,也有研究从计算资源的硬件特性考虑,合理分配计算资源. Liao等人[9]从CPU(Central Processing Unit)硬件资源竞争的角度考虑,通过保守估计CPU资源使用率,来缓解CPU过度分配的现象,防止出现服务等级协议(Service Level Agreement,SLA)违规的风险. 尽管启发式调度算法具有环境适应性优势,但其规则驱动的特性导致该方法难以应对动态环境下的复杂优化目标.

进化优化算法通过模拟自然进化过程寻求近似最优解. 这方面的研究通常会改进现有的遗传算法,缩小问题的搜索空间来优化问题的求解过程. Pallewatta等人[10]提出了两阶段优化框架:第一阶段采用遗传算法实现微服务部署,第二阶段运用粒子群算法完成副本冗余配置;Xia等人[11]提出了多目标遗传算法实现运行能耗和执行时间之间的平衡;Zhou等人[12]改进了遗传算法的搜索方式,通过任务间转移知识优化具有约束条件的工作流调度问题. 尽管此类算法在解空间探索方面具有优势,但其迭代优化过程依然需要消耗大量计算资源,导致响应时延增加,在实时性要求较高的场景中适用性受限.

强化学习方法通过智能体与环境的交互学习最优策略. 目前大多数研究中,通过设计智能体与环境交互,为智能体设置合适的奖励函数,智能体自主探索学习得到调度策略. Chen等人[13]创新性地结合图神经网络提取服务器拓扑和微服务DAG的特征表示,并采用改进的序列到序列(Seq2Seq)强化学习模型建立微服务组件与服务器节点的映射关系;Lin等人[14]使用强化学习的方法,并引入自注意力学习机制,使工作流在复杂的云环境中能满足截止时间要求,同时降低成本;Jayanetti等人[15]使用多智能体强化学习模型,减少工作流在多云环境下的运行能耗;Chen等人[16]不预设并行任务的执行顺序,结合集群当前状态做出调度决策,实现任务的快速卸载,以达到移动应用的最优性能;Jian

等人[17]在Kubernetes平台上采用了深度强化学习算法，实现任务的负载均衡调度．值得注意的是，现有研究在设计调度方案时，未充分考虑边缘服务器的异构特征、云服务器与边缘服务器间的资源差异和边缘服务器计算性能波动性质．

### 2.2 大模型应用研究

生成式人工智能技术的快速发展及其多模态应用，凸显了大模型在复杂任务中所具备的推演与决策能力．在智能系统自动化领域，Wen等人[18]通过整合现有大模型实现安卓任务自动化；He等人[19]创新性地构建了基于大模型的日常事务助理框架，辅助人类进行规划与执行．在代码工程领域，Fakhoury等人[20]使用大模型实现自动化编程，展现出大模型代码生成的巨大潜力；Gu等人[21]借助大模型自动化生成golang编译器测试用例．这些前沿研究系统性论证了大模型在跨领域、多模态场景下的强大推理能力，为本文提供了理论基础和方法启示．本文创新性地将大模型引入工作流调度领域，提出基于大模型辅助的云边协同工作流调度算法LAWS.

## 3 问题建模

### 3.1 云边协同环境

在云边协同计算架构中，云服务器作为核心枢纽连接着多个边缘计算节点，形成层次化的计算资源分布体系，如图1所示．云平台凭借其集中式部署优势，具备充裕的计算资源与海量存储能力，但受限于网络传输带宽和时延约束，其通信能力存在显著瓶颈．相较而言，邻近终端设备部署的边缘节点具有低时延、高带宽的通信优势，然而受限于物理规模，其计算资源与存储容量相对有限．值得注意的是，边缘计算节点普遍存在计算性能波动现象，尤其在多任务并发场景下易出现资源竞争问题，这种动态资源特性对系统调度策略提出了特殊要求．

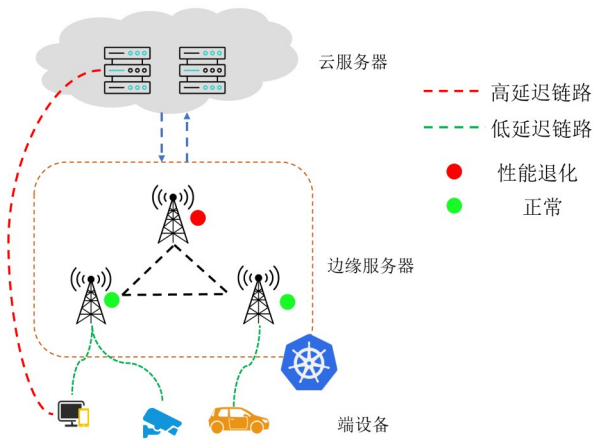



图1 云边协同计算环境

本文将云边协同环境形式化为$S=\{\text{Cloud},e_1,e_2,\cdots,e_i,\cdots,e_n\}$，$1\leqslant i\leqslant n$，Cloud表示云服务器，$e_i$表示第$i$个边缘节点．云服务器和边缘节点均拥有$m$种可以使用的计算资源，相应的资源利用率表示为

$$R_{s,r}=\frac{\text{Used}_{s,r}}{Q_{s,r}},\ s\in S,\ 1\leqslant r\leqslant m \tag{1}$$

其中，$\text{Used}_{s,r}$表示服务器$s$第$r$种计算资源的使用量，$Q_{s,r}$表示服务器$s$第$r$种计算资源的总量．并且云服务器的计算资源会比边缘节点的多，即$Q_{\text{Cloud},r}>Q_{e_j,r}$，$1\leqslant r\leqslant m$，$1\leqslant j\leqslant n$.

在云边协同环境中，端设备通过无线局域网(Wireless Local Area Network，WLAN)接入网络，其中边缘节点通过广域网(Wide Area Network，WAN)与云节点建立连接，边缘节点之间通过局域网(Local Area Network，LAN)进行内部通信[10]，依据网络架构特性，端设备与各节点的通信带宽$\text{Band}_{\text{end},s}$满足以下不等式：$\text{Band}_{\text{end,Cloud}}<\text{Band}_{\text{end},e_i}$，$1\leqslant i\leqslant n$.

虽然边缘节点具备较强的通信能力，但其计算性能的不稳定性会显著影响工作流执行效率．为此，本文提出一种基于概率建模的性能动态表征方法：定义性能退化因子$F_s\in\{0,1\}$，其中$F_s=0$表示服务器处于稳定运行状态，$F_s=1$表示性能达到最大退化阈值．本文针对边缘服务器的性能动态特征，采用威布尔分布[22]表征性能退化事件的发生规律，使用对数正态分布描述系统恢复事件的时间概率分布[23]．在此基础上，通过蒙特卡洛随机模拟方法，生成符合上述分布规律的性能退化事件序列与恢复时间序列．需要特别说明的是，与具有动态特性的边缘服务器不同，本文假设云服务器在架构设计和资源供给方面具有稳定性优势，即$F_{\text{cloud}}=0$．云边协同环境状态表示为

$$\text{State}_s=\{R_{s,r},\text{Band}_s,F_s\},\ s\in S,\ 1\leqslant r\leqslant m \tag{2}$$

$$\text{State}=\{\text{State}_s\},\ s\in S \tag{3}$$

### 3.2 工作流任务

工作流任务表示为一个有向无环图DAG $G=\{W,\varepsilon\}$，$W$为顶点集，$W=\{w_1,w_2,\cdots,w_i,\cdots,w_u\}$，$u=|W|$是工作流中子任务的个数．$\varepsilon$为边集，给定两个子任务$w_i$和$w_j$，$(w_i,w_j)\in\varepsilon$表示$w_j$的执行依赖于$w_i$的完成．图2展示了一个工作流的具体实例．工作流的各个子任务执行时间可以表示为$T=\{T_{w_1},T_{w_2},\cdots,T_{w_i},\cdots,T_{w_u}\}$．子任务的运行时间$T_{w_i}$由6个部分组成，即

$$T_{w_i}=T_{w_i}^{\text{schedule}}+T_{w_i}^{\text{load}}+T^{\text{receive}}+T_{w_i}^{\text{compute}}+T^{\text{send}}+T_{w_i}^{\text{tail}} \tag{4}$$

其中，$T_{w_i}^{\text{schedule}}$为子任务的调度时间；$T_{w_i}^{\text{load}}$为下载容器镜像并加载镜像的时间，这个阶段需要将镜像文件下载并写入磁盘，然后将镜像加载到内存中再执行；$T^{\text{receive}}$

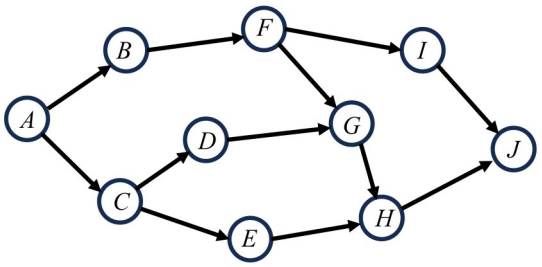



图2　工作流实例

为子任务接收前驱子任务数据的时间;$T_{w_i}^{\text{compute}}$为子任务的计算时间;$T^{\text{send}}$为子任务向后驱子任务发送数据的时间;$T_{w_i}^{\text{tail}}$为子任务$w_i$尾部处理时延,包括关闭容器、释放资源、收集数据等. 其中,$T^{\text{receive}}$和$T^{\text{send}}$作为子任务间的通信时延分别定义为

$$T^{\text{receive}}=\frac{D^{\text{receive}}}{\text{Band}_{a,b}} \tag{5}$$

$$T^{\text{send}}=\frac{D^{\text{send}}}{\text{Band}_{a,b}} \tag{6}$$

其中,$D^{\text{receive}}$表示子任务接收的数据量;$D^{\text{send}}$表示子任务发送的数据量;$\text{Band}_{a,b}$表示发送数据节点$a$和接收数据节点$b$之间的带宽.

子任务$w_i$最早结束时间可以表示为

$$\text{et}_{w_i}=\begin{cases}t_{\text{pre}}+T_{w_i}, & \text{if Pre}(w_i)=\varnothing\\ \max\limits_{w_j\in\text{Pre}(w_i)}\text{et}_{w_j}+T_{w_i}, & \text{if Pre}(w_i)\neq\varnothing\end{cases} \tag{7}$$

其中,$\text{Pre}(w_i)$表示子任务$w_i$的前序子任务集;$w_j\in\text{Pre}(w_i)$为$w_i$的直接前序子任务. 如果$w_i$没有前序子任务,则子任务开始时间为完成工作流$G$待处理数据上传完成的时间$t_{\text{pre}}$. 同理,在维持工作流关键路径持续时间不变的约束条件下,子任务$w_i$最晚开始时间表示为

$$\text{lt}_{w_i}=\begin{cases}t_{\text{end}}-T_{w_i}, & \text{if Post}(w_i)=\varnothing\\ \min\limits_{w_j\in\text{Post}(w_i)}\text{lt}_{w_j}-T_{w_i}, & \text{if Post}(w_i)\neq\varnothing\end{cases} \tag{8}$$

其中,$\text{Post}(w_i)$表示子任务$w_i$的直接后序子任务集;$w_j$为子任务$w_i$的直接后继子任务;$t_{\text{end}}$表示所有子任务完成的截止时间,且满足式(9):

$$t_{\text{end}}=\max_{w\in W}\text{et}_w \tag{9}$$

相应地,工作流的最早完成时间为

$$\text{CT}_W=t_{\text{end}}+T_{\text{clean}} \tag{10}$$

即所有子任务完成的截止时间加上工作流$G$审计工作所需的时间$T_{\text{clean}}$. 由此可以得到子任务$w_i$的松弛时间$\text{RT}_{w_i}$为

$$\begin{aligned}\text{RT}_{w_i}&=\left(\text{lt}_{w_i}+T_{w_i}\right)-\left(\text{et}_{w_i}-T_{w_i}\right)\\&=\text{lt}_{w_i}-\text{et}_{w_i}+2\times T_{w_i}\end{aligned} \tag{11}$$

即子任务$w_i$的最晚结束时间减去最早开始时间. 如果$\text{RT}_{w_i}=T_{w_i}$说明子任务$w_i$在工作流$G$的关键路径上. 相应地,子任务$w_i$的影响因子定义为

$$\text{IF}_{w_i}=\frac{T_{w_i}}{\text{RT}_{w_i}} \tag{12}$$

其中,$\text{IF}_{w_i}$的值与子任务$w_i$对工作流的真实执行时间$\text{CT}'_W$的影响程度正相关.

每一个子任务$w_i$执行时都需要占用计算资源,比如CPU、内存、磁盘等,占用计算资源的数量为

$$\text{Req}_{w_i}=\left\{\text{Req}_{w_i,1},\text{Req}_{w_i,2},\cdots,\text{Req}_{w_i,r},\cdots,\text{Req}_{w_i,m}\right\}$$

其中,$\text{Req}_{w_i,r}$表示子任务$w_i$运行时占用的第$r$种计算资源的数量. 所以,子任务$w_i$表示为

$$w_i=\left\{\text{Req}_{w_i},T_{w_i},\text{IF}_{w_i}\right\} \tag{13}$$

当子任务$w_i$调度到服务器$s$上时,对服务器计算资源使用率的改变可以表示为

$$\text{Used}_{\text{s,r}}(t')=\begin{cases}\text{Used}_{s,r}(t)+\text{Req}_{w_i,r}, & \text{子任务}w_i\text{正在运行}\\ \text{Used}_{s,r}(t)-\text{Req}_{w_i,r}, & \text{子任务}w_i\text{运行结束}\end{cases} \tag{14}$$

即子任务部署到服务器$s$上时,会占用相应的计算资源,当子任务完成时,会释放相应的计算资源. $t$表示子任务$w_i$状态改变前的时间,$t'$表示子任务$w_i$状态改变后的时间.

### 3.3　调度问题

#### 3.3.1　决策变量

在工作流调度模型中,每个工作流的子任务都需要映射到一个服务器$s\in S$执行. 为此,本文构建决策变量$x_{i,j,k}\in\{0,1\}$. 当工作流$i$的子任务$j$映射到服务器$k$上时,$x_{i,j,k}=1$,否则$x_{i,j,k}=0$. 本文将用户请求的工作流集合定义为$W_{\text{set}}$,调度决策变量集合表示为$X=\left\{x_{i,j,k}\middle|1\leqslant i\leqslant\left|W_{\text{set}}\right|,1\leqslant j\leqslant\left|W_i\right|,k\in S\right\}$.

#### 3.3.2　约束条件

当服务器出现资源请求数量超出节点资源配置时会导致工作流失效,因此需要建立严格的资源约束模型. 映射到服务器$k$上的子任务需要满足:

$$\sum_{i\in W_{\text{set}}}\sum_{j\in W_i}\sum_{k\in S}x_{i,j,k}\times\text{Req}_{i,j,r}\leqslant Q_{k,r}-Q_{k,r}^{\text{sys}},\quad 1\leqslant r\leqslant m \tag{15}$$

其中,$\text{Req}_{i,j,r}$表示请求的工作流集合$W_{\text{set}}$的第$i$个工作流中第$j$个子任务请求资源$r$的数量;$Q_{k,r}$表示服务器$k$所配置的资源$r$的数量;$Q_{k,r}^{\text{sys}}$表示服务器$k$的系统运行所需资源$r$的数量,即服务器$k$需要预留出系统运行所需要的计算资源.

每一个子任务都应该且只能被调度到一个服务器上,即

$$\sum_{k\in S}x_{i,j,k}=1,\ 1\leqslant i\leqslant\left|W_{\text{set}}\right|,\quad 1\leqslant j\leqslant\left|W_i\right| \tag{16}$$

#### 3.3.3 问题公式化

云边协同环境结合了云计算提供的弹性计算资源和边缘计算所带来的低时延优势. 在此环境下,工作流调度面临双重挑战:一是云节点远程数据传输所引入的通信时延及单位时间计算成本;二是边缘节点性能退化所导致的任务执行时延增加.

LAWS将边缘比例和时延比作为优化目标,具体定义如下:

边缘比例表示在边缘节点执行的子任务数量占整个工作流子任务总数的比例,具体定义为

$$I(w_i)=\begin{cases}1, & \text{子任务}w_i\text{部署到边缘节点}\\ 0, & \text{否则}\end{cases}$$

$$\text{Rate}_W=\frac{\sum_{i=1}^{|W|}I(w_i)}{|W|} \tag{17}$$

其中,$I(w_i)$为指示函数,该函数用于标识子任务$w_i$是否部署在边缘节点.

时延比表示工作流实际运行时间$\text{CT}'_W$与预期时间$\text{CT}_W$的比例,具体定义为

$$\text{TimeRate}_W=\frac{\text{CT}'_W}{\text{CT}_W} \tag{18}$$

LAWS在尽可能减少$\text{TimeRate}_W$的情况下,将子任务部署到边缘节点以减少通信时延的开销,优化目标可由式(19)计算:

$$\begin{aligned}&\max F(W)=\left\{\text{Rate}_W,\ \frac{1}{\text{TimeRate}_W}\right\}\\ &\text{s. t.}\quad \text{Eqs(15)(16)}\\ &x_{i,j,k}\in\{0,1\},\quad 1\leqslant i\leqslant\left|W_{\text{set}}\right|,\quad 1\leqslant j\leqslant\left|W_i\right|,\quad k\in S\end{aligned} \tag{19}$$

## 4 算法设计

### 4.1 调度算法概览

如图3所示,LAWS采用闭环优化架构,主要分成四个部分:调度样本集、策略优化器、策略思维链知识图谱、在线调度引擎. 其中,调度样本集由在线调度引擎收集,并提供给策略优化器进行采样作为提示样本. 策略优化器获取提示样本后,按照优化算法总结调度策略,并据此提取出思维链知识图谱. 当请求到达时,在线调度引擎按照思维链知识图谱进行工作流调度决策,并最终调度到集群中执行. 整个算法主要包含以下几个步骤.

#### 4.1.1 样本数据收集和处理

模型训练需要采集反映不同调度策略对工作流执行时延影响的调度实例数据,作为大模型策略学习和优化的依据. 本文通过收集样本构建初始样本库,并使用行列式点过程(Determinantal Point Process, DPP)算法采样. 采样所得样本经处理后,形成用于提示的样本集.

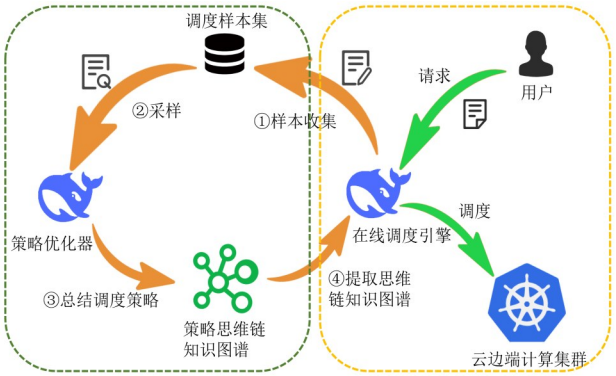



图3 算法调度框架示意图

#### 4.1.2 调度策略存储和提示

为实现调度策略提示的语义压缩与结构优化,本文采用知识图谱对调度策略进行结构化表示与存储,通过实体关系建模增强策略要素的可解释性,从而提升大模型对复杂调度逻辑的认知与推理能力. 知识图谱是一个包含实体和关系的图结构,能够有效地表示和存储复杂的信息[24].

#### 4.1.3 在线调度决策

当调度特定工作流时,本文将服务器的状态State、待调度的工作流$G$和表示调度策略的知识图谱进行组合,组成提示词,提供给大模型进行推理,得到决策变量$X$.

#### 4.1.4 策略的异步更新

LAWS的调度模块可以分为两个部分:策略更新模块是将调度策略提炼为思维链;任务调度模块在接收到任务请求时,根据策略更新模块提供的思维链进行推理决策. 为实现调度系统的实时响应能力,LAWS将调度思维链的更新与对工作流的调度决策解耦异步执行. 具体地,策略更新模块每隔时间$t$,从历史数据中提取提示样本,提供给大模型更新调度思维链. 任务调度模块在工作流任务请求到达时,结合策略更新模块提供的思维链,向大模型请求获取工作流决策变量$X$,工作流执行结束后,其相关信息将被存储至历史数据集,以供策略更新模块参考更新策略思维链.

### 4.2 学习样本收集和处理

为构建大模型学习的样本集,需要收集以下监控数据:子任务调度前特定时间段、任务执行时服务器状态的变化和工作流的实际执行时间$\text{CT}'_W$. 随后,将这些监控数据、工作流的有向无环图表示$G$和决策变量$X$构成学习样本,并存储在样本集中.

LAWS使用工作流的加权资源需求RA和奖励函数值reward作为工作流的特征值$F_W$表征调度样本之间的差异. $F_W$和各个特征值的定义如下:

$$F_W=\{\mathrm{RA}_{W,1},\mathrm{RA}_{W,2},\cdots,\mathrm{RA}_{W,r},\cdots,\mathrm{RA}_{W,m},\mathrm{reward}\} \quad (20)$$

$$\mathrm{RA}_{W,r}=\frac{\sum_{j=1}^{|W|}\mathrm{IF}_j\times\mathrm{Req}_{W_j,r}}{\sum_{k\in W}\mathrm{IF}_k},\ W\in W_{\mathrm{set}},\quad 1\leqslant r\leqslant m \quad (21)$$

$$\mathrm{reward}=\theta\times\frac{1}{\mathrm{TimeRate}_W}+(1-\theta)\times\mathrm{Rate}_W \quad (22)$$

本文通过两次实验进行数据采集:首次实验采用云端优先调度策略(标记为负样本NS),第二次实验采用边缘优先调度策略(标记为正样本PS). LAWS以概率$\gamma$筛选样本,旨在缩小样本量.

接下来将得到的样本集进行DPP采样[25],得到最终的提示样本. 其中,样本之间的相似度度量用内积表示,即

$$d_{i,j}=\sum F_{W_i,f}\times F_{W_j,f},\ 1\leqslant i,\quad j\leqslant|W|,f\in F_W \quad (23)$$

根据式(23)计算得到相似度,使用DPP采样算法最大差异化选取$k$个样本得到历史调度样本集$S$. 将样本集、数据字段含义描述和负载均衡提示组合成情景提示BackGround(BG). 提取样本集$S$过程在算法1中描述.

### 4.3　问题分解与解答

为缓解大模型在调度决策中的认知失准问题,本文使用知识图谱工具表示思维链,对调度策略问题进行分解,并提取知识图谱子图辅助大模型推理决策. 图4展示了问题分解的流程.

首先,基于第3节对云边协同环境和工作流的建模,通过大模型进行实体抽取与属性关系挖掘,最终构建了初始知识图谱InitKG. 知识图谱以三元组的形式表示,各个字段分别为{head, relation, tail},部分内容如表1所示.

**算法1　历史数据采样**

**输入**: 正样本PS,负样本NS,采样数量$k$
**输出**: 样本集合$S$

```
SubSet = [];
FOR each sample∈PS DO
f = uniform(0,1);
IF f ≤ γ THEN
SubSet.append(sample);
END IF
END FOR
FOR each sample∈NS DO
f = uniform(0,1);
IF f ≤ γ THEN
SubSet.append(sample);
END IF
END FOR
WHILE k > 0 DO
t = min(k, rank(d_{i,j}));
tmpS = DPP(SubSet, t); // 根据DPP算法从SubSet中提取出t个样本
S = S ∪ tmpS;
k = k − t;
END WHILE
RETURNS
```

**表1**　InitKG的部分内容

| head | relation | tail |
|---|---|---|
| 关键路径组件优先分配 | 影响 | 任务完成时间偏差 |
| 磁盘密集型组件 | 应避免分配至 | 高disk_write节点 |
| 并行组件 | 适合分配至 | 多低负载节点 |

对于每个问题,本文采用递归分解的方式来解决. 通过以下四个阶段实现调度策略的系统性求解:提取当前问题相关的知识图谱、问题分解、差异化解决子问

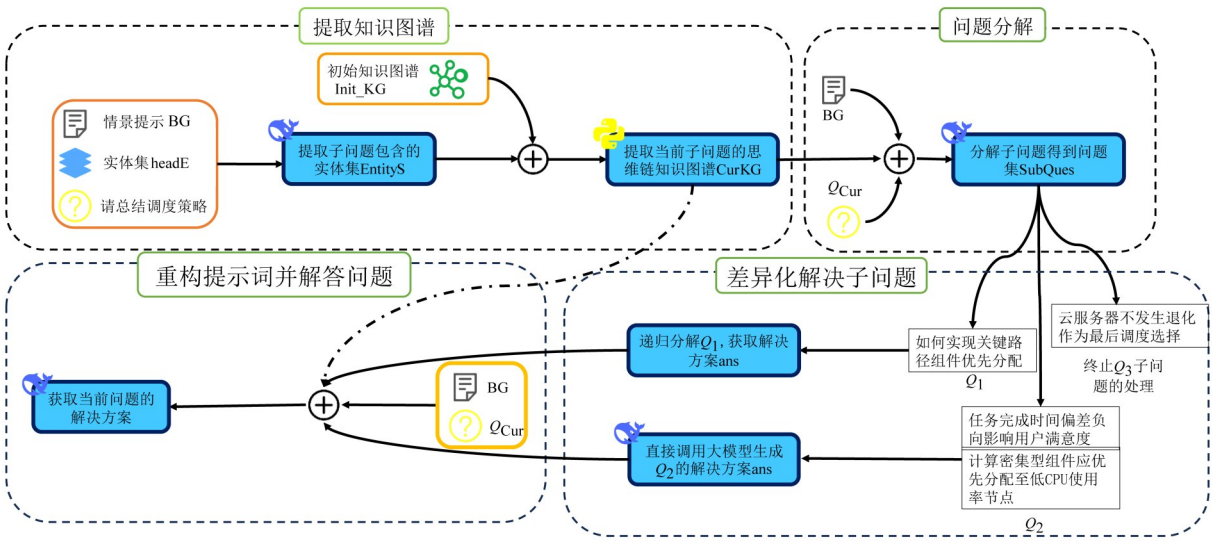



图4　问题分解流程概览

题和重构提示词引导大模型进行推理解答.

#### 4.3.1 提取知识图谱

本文提出一种基于知识图谱的注意力优化方法,旨在提升大模型在调度问题求解中的任务聚焦能力.依据子问题的内容,从InitKG提取出子图谱CurKG作为子问题的思维链.该方法能够有效消除冗余信息干扰,同时保持子问题与调度需求之间的语义关联.

该阶段的提示词为Extract_KG_Prompt={BG, HeadE, $Q_{cur}$}.其中,BG为4.2节中组合得到的情景提示词,HeadE是知识图谱InitKG中head字段所含有的实体,$Q_{cur}$是当前待解决的问题.提示大模型解析$Q_{cur}$语义,并提取出问题所包含的实体集EntityS,根据实体集EntityS从InitKG提取出当前子问题的思维链知识图谱CurKG.

提取知识图谱的过程在算法2中描述.其核心思想是利用大模型的语义理解能力,从全局知识图谱中为当前问题智能地筛选出一个高度相关的子集,作为当前子问题的思维链CurKG.详细步骤如下:算法首先遍历InitKG中的所有三元组,提取并汇集所有head字段的实体,构建实体集HeadE.接着,情景提示词BG、实体集HeadE、当前子问题$Q_{Cur}$三者共同构成Extract_KG_Prompt.将其输入大模型,指导大模型分析$Q_{Cur}$的语义,从HeadE中识别出与问题直接相关的实体,构成实体集EntityS.最后,再次遍历InitKG中的三元组,筛选出head字段包含于EntityS中的三元组组成子图谱CurKG.

通过上述算法步骤,将一个庞大且可能包含大量冗余信息的初始知识图谱,精炼成一个轻量级、高相关的子知识图谱CurKG,通过消除噪声干扰,为大模型提供了一个清晰、聚焦的思维链,从而显著提升其在特定任务上的表现和准确性.

**算法2 提取知识图谱**

**输入**: 初始知识图谱InitKG, 当前待解决的问题$Q_{Cur}$, 情景提示词BG

**输出**: 子图谱CurKG

```
HeadE = []
FOR each t ∈ InitKG DO
HeadE.append(t['head']);
END FOR
Extract_KG_Prompt = {BG, HeadE, Q_Cur};
结合Extract_KG_Prompt提示大模型提取Q_Cur包含HeadE的实体, 形成EntityS
CurKG = []
FOR each t ∈ InitKG DO
IF t['head'] ∈ EntityS THEN
CurKG.append(t);
END IF
END FOR
```

#### 4.3.2 问题分解

模仿人解决问题的思维,将问题解构为多维度子问题集,基于与调度需求相关性对子问题进行筛选、解答,从而达到对调度问题更加精细的分析,有效抑制了模型生成过程中的幻觉现象[22,26,27].

该阶段以Divide_Prompt={BG, CurKG, $Q_{Cur}$}作为提示,得到子问题集SubQues = $\{(q, \text{impact})\}$, SubQues是一组二元组的集合,每个元素包含子问题$q$和该子问题对当前问题的影响因子impact.

#### 4.3.3 差异化解决子问题

大模型分解得到的子问题与调度问题的相关程度不同,为保障策略的实时性,本文根据影响因子impact对子问题进行划分,并采取不同的策略.具体的实施过程如下:将SubQues中的问题按照字段impact进行降序排序,并将问题集划分成三类$\{Q_1, Q_2, Q_3\}$,满足$|Q_1|+|Q_2|+|Q_3|=|\text{SubQues}|$. $Q_1$包含与调度目标强耦合关系的核心问题,需通过递归分解算法进行二次解构;$Q_2$由中等相关性问题构成,直接调用大模型生成解决方案;$Q_3$则对应弱相关性问题,采用早期剔除策略终止处理流程.随后,将$Q_1$和$Q_2$的问题和解答组成问答对集合Q&A=$\{(q, \text{ans}) \mid q \in Q_1 \cup Q_2\}$.

#### 4.3.4 重构提示词并解答问题

将Prompt={BG, Q&A, CurKG, $Q_{Cur}$}输入大模型,得到最终调度决策.特别地,初始问题$Q_{Cur}$=$Q_0$="请总结调度策略".此时,CurKG=InitKG.本文将子问题的深度变量$\text{DEPTH}_{Cur}$作为停止问题分解的控制变量.其中$\text{DEPTH}_{Cur}=\text{DEPTH}_{fath}+1$, $\text{DEPTH}_{Q_0}=0$, $\text{DEPTH}_{fath}$为父问题的深度.当$\text{DEPTH}_{Cur}$=MAX_DEPTH时,不再对问题进行解构分析.

使用知识图谱划分子问题求解的过程在算法3中描述.

### 4.4 策略更新与调度决策

图5展示了策略更新阶段提供给大模型的样本数量与调用时间的关系.从图5可以看出,大模型的平均推理时延不低于375 s,对调度系统的实时性构成了瓶颈约束.为此,本文提出了策略更新模块与任务调度模块的异步协作架构.策略更新模块以知识图谱PolicyKG的形式为任务调度模块提供思维链,任务调度模块按照PolicyKG推理决策并记录和存储工作流执行结果供策略更新模块更新优化策略.

#### 4.4.1 策略更新

每间隔时间$t$,从历史调度数据中按照算法1获取采样样本集$S$,然后按照算法3得到文字描述的策略

Policy,最后大模型将 Policy 总结成一幅知识图谱 PolicyKG 供在线调度引擎推理和调度.

**算法3　问题分解与求解**

**输入**: 样本集 $S$,初始知识图谱 InitKG,当前问题 $Q_{Cur}$,深度 DEPTH
**输出**: 调度策略 ans

```
IF DEPTH ≠ 0 THEN
使用 Extract_KG_Prompt 提示大模型获取 EntityS
CurKG = [];
FOR each relation ∈ InitKG DO
IF relation['head'] ∈ EntityS THEN
CurKG.append(relation);
END IF // 根据 EntityS 提取当前问题的思维链知识图谱
END FOR
ELSE
CurKG = InitKG; // 初始问题 Q0 的思维链知识图谱为 InitKG
END IF
IF DEPTH = MAX_DEPTH THEN
    结合 Prompt 提示大模型解决问题获取答案 ans
    RETURN ans
END IF // 到达分解问题的边界,停止问题分解
结合 Divide_prompt 提示大模型获取子问题 SubQues
将 SubQues 按照 impact 字段降序排序
Q&A = []; // 问答对集合
Q1=SubQues[0...|Q1|]; // 第一类问题
Q2=SubQues[|Q1|...(|Q1|+|Q2|)] // 第二类问题
Q3=SubQues[(|Q1|+|Q2|)...|SubQues|] // 第三类问题
FOR each q ∈ Q1 DO
    递归分解问题 q 获得解答 curans,
    Q&A.append((q, curans));
END FOR
FOR each q ∈ Q2 DO
结合 Prompt 提示大模型解决问题获取答案 curans;
Q&A.append((q, curans));
END FOR
结合 Prompt 提示大模型获取 ans
RETURN ans
```

#### 4.4.2　任务调度

当工作流请求抵达系统时,首先构造提示向量 Schedule_Prompt={State,$G$,PolicyKG}作为提示输入,驱动大模型生成调度决策变量 $X$. 其中,State 为式(3)定义的云边协同环境状态,$G$ 为待调度工作流的 DAG 表示. 随后,系统依据决策变量 $X$ 将工作流部署到集群中. 在工作流任务执行完成后,系统记录 $CT'_W$、$Rate_W$ 并形成结构化调度样本存储到调度样本集中,供策略更新模块参考.

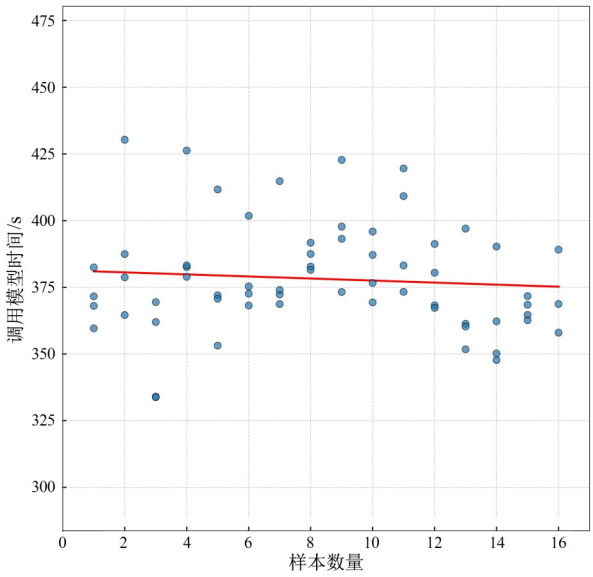



图5　样本数量与调用模型的时间关系

### 4.5　复杂度分析

LAWS 主要分成两部分:策略更新模块和任务调度模块. 鉴于调用大模型是算法执行时间的主要开销,所以将调用大模型作为复杂度分析的元操作. 其中,LAWS 的算法复杂度主要由策略更新模块主导,下面聚焦于对策略更新模块的时间复杂度分析.

策略更新模块的子问题可以组织成一棵以初始问题 $Q_0$ 为根节点的 $k$ 叉树,其中分支因子 $k=|Q_1|+|Q_2|$. 策略更新阶段的时间复杂度主要由 $k$ 叉树的非叶子节点数量决定,随着问题分解边界 MAX_DEPTH(记为 $n$)的增加,非叶子节点呈指数式增长. 设 $|Q_1|=m$,则策略更新阶段的时间复杂度为 $O(m^n)$.

## 5　实验分析

### 5.1　实验环境

本文在基于 kubernetes 的异构集群平台上开展实验验证,集群有 1 个主节点和 6 个工作节点,所有节点的操作系统均为 Ubuntu 22.04.4 LTS. 主节点负责集群监控、请求接收和任务调度,不参与实际工作负载执行.

为构建模拟云边协同异构环境的实验平台,本文通过差异化配置节点计算资源来实现节点间的性能差异. 仅保留 k8snode1 节点的资源未被缩减,使其具备相对集约化和丰富的计算资源,用于模拟云边协同环境中的云计算平台节点. 其余节点则被配置为资源受限状态(资源占用缩减),用于模拟边缘计算节点. 本实验中,终端设备向初始子任务执行节点传输数据所产生的通信时延通过仿真方式评估. 鉴于任务子节点间传输的数据量较小,其通信时延可忽略不计. 各个节点的计算资源如表 2 所示. 通过实验发现,当给节点注入

500 MB/s的磁盘I/O负载时，子任务的执行时延平均增加58%. 因此，本文采用向目标节点施加人工合成的高吞吐量磁盘读写负载方法，构建节点性能不稳定的仿真场景，并根据第3.1节的建模方式触发性能下降事件和恢复事件.

**表2 各节点的计算资源容量**

| 节点名 | 计算资源容量 |
|---|---|
| k8snode1 (云服务器) | CPU: 12 Core, Memory: 32 GB, Bandwith: 600 Mbit/s |
| k8snode2 (边缘服务器) | CPU: 7 Core, Memory: 24 GB, Bandwith: 900 Mbit/s |
| k8snode3 (边缘服务器) | CPU: 9 Core, Memory: 28 GB, Bandwith: 900 Mbit/s |
| k8snode4 (边缘服务器) | CPU: 8 Core, Memory: 27 GB, Bandwith: 900 Mbit/s |
| k8snode5 (边缘服务器) | CPU: 7 Core, Memory: 20 GB, Bandwith: 900 Mbit/s |
| k8snode6 (边缘服务器) | CPU: 6 Core, Memory: 20 GB, Bandwith: 900 Mbit/s |

基于包含100个随机任务的训练集，本文采用$K$近邻($K$-Nearest Neighbors，KNN)回归算法[28]训练子任务执行时间预测模型，该模型在测试集上表现出3 s的平均绝对误差. 除此之外，基于KNN算法进一步建立了子任务间调用概率$p$及非关键路径子任务占比的预测模型，用于控制工作流的并行度.

为验证第4.5节中时间复杂度分析的结论，我们对策略更新时延与关键参数MAX_DEPTH的关系进行了实验测试，结果如图6所示. 实验数据表明，更新时延随MAX_DEPTH的增加呈指数增长趋势. 为满足策略更新模块的实时性要求，并充分发挥算法在引导大模型分析云边协同环境、优化工作流调度中的作用，本文在参数配置中限定MAX_DEPTH=2，同时选取与调度策略问题具有耦合性最强的问题进行分解，同时对另一耦合性较强的问题直接调用大模型生成解决方案，即$|Q_1|=1, |Q_2|=1$.

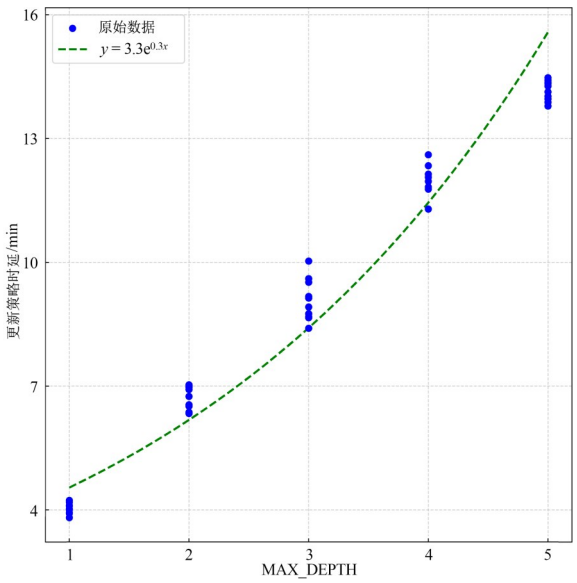



图6 MAX_DEPTH与更新策略时延关系曲线

## 5.2 参数对算法性能的影响

为探究不同参数对LAWS性能的影响，本文分别针对奖励函数参数$\theta$和采样样本数量$k$展开对比实验.

在参数$\theta$的敏感性分析中，选取了0.1、0.3、0.5、0.7、0.9五个等间距参数值，系统评估其对大模型决策的影响.

实验结果如图7所示. 当$\theta=0.7$时，相较于其他参数值，其工作流完成时间最短，计算能耗处于较低水平. 在该参数下，大模型在有效降低工作流执行时延的同时，将50.7%的子任务合理分配至边缘节点，符合调度目标需求.

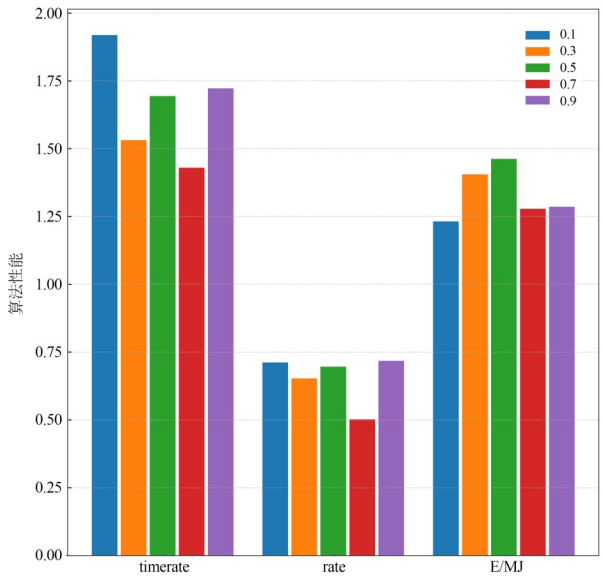



图7 不同$\theta$值下的性能指标

针对样本数量参数$k$的敏感性分析，本文采用等间距的$k$值系统探究样本规模对LAWS决策质量的影响. 实验结果如图8所示. 当$k=3$时，系统达到最优状态，工作流完成时间相较于其他参数值缩短了32%，计算能耗也处于较低水平. $k$值过大或者过小都会导致大模型调度到边缘节点的任务数量过多，导致边缘节点因计算资源不足引入额外的执行时延.

## 5.3 算法性能评估

用于实验的工作流是随机产生的，每个工作流包含4~10个具有依赖关系的子任务(服从均匀分布). 本次实验中，子任务不在关键路径上的比例控制在0.3~0.7之间. 根据第5.2节的分析，奖励函数的$\theta$设置为0.7，采样样本数量设置为3.

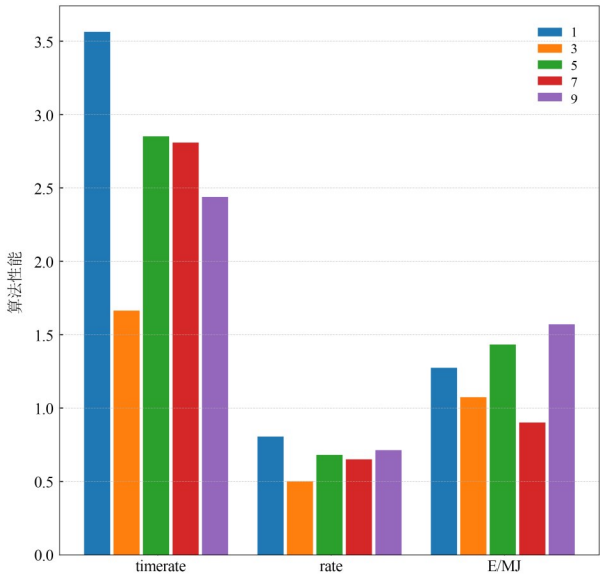



图8　不同$k$值下算法性能指标

### 5.3.1　对比算法

本文选取以下算法作为对比:Kubernetes(k8s)默认调度策略、基于深度强化学习的Kubernetes调度器(Deep Reinforcement learning based kubernetes Scheduler, DRS)、基于大模型的调度算法(Large Language Model-based Scheduling Algorithm, LLM),剩余CPU(Remaining CPU, RCPU). 具体算法内容如下:

Kubernetes(k8s)默认调度器预先设定了一些评分机制$F=\{f_i(\text{State}_s)|s\in S\}$,每个评分算法都有相应的权重$\text{Weight}_i$,每个节点的得分根据式(24)计算,然后从中选择得分最高的节点作为调度节点.

$$\text{Priority}_s=\frac{\sum f_i(\text{State}_s)\times\text{Weight}_i}{\sum\text{Weight}_i} \tag{24}$$

DRS使用深度强化学习驱动策略进行调度[17]. 马尔可夫决策过程的状态空间包含节点资源使用率和待调度任务的资源需求,动作空间为可选的节点集,基于全局资源利用率、节点间负载不均衡度设计奖励函数.

LLM只向大模型提供节点状态信息State和待调度的工作流$G$,调用大模型得到决策变量$X$.

RCPU在调度时,取同一物理核心上所有硬件线程的最大利用率作为该核心的利用率,避免对剩余资源的高估,减少多线程之间硬件的资源竞争问题[9].

### 5.3.2　性能指标

运行时间比例是指工作流实际运行时间与预期运行时间之比的平均值.

$$\text{timerate}=\frac{\sum_{W\in W_{\text{set}}}\text{TimeRate}_W}{|W_{\text{set}}|} \tag{25}$$

其中,$\text{TimeRate}_W$根据式(18)计算得到.

计算能耗是指工作流执行所消耗的能量,计算公式如下:

$$P_s=\theta_s\times P_{\text{base}} \tag{26}$$

$$E=\sum_{s\in S}\sum P_s t \tag{27}$$

其中,$\theta_s$为节点的资源利用率,$P_{\text{base}}$表示节点高负载运行时的功率,本实验设置$P_{\text{base}}$为50 W.

子任务部署在边缘节点的比例是工作流子任务部署在边缘比例的平均值.

$$\text{rate}=\frac{\sum_{W\in W_{\text{set}}}\text{Rate}_W}{|W_{\text{set}}|} \tag{28}$$

其中,$\text{Rate}_W$根据式(17)计算得到.

### 5.3.3　实验设置和结果分析

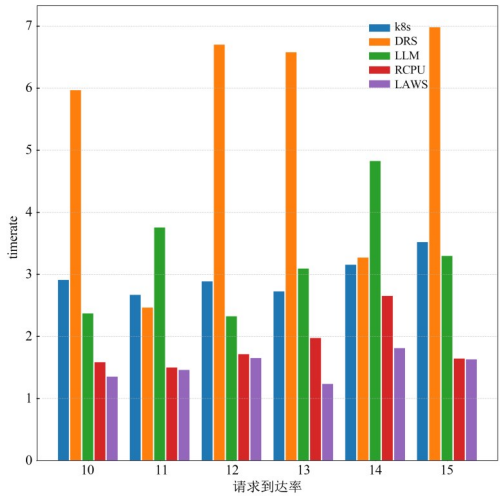
本研究设计了多组实验,从请求到达率、到达率分布以及工作流类型等多个维度对比不同算法的执行性能. 同时,设置了消融实验以分析LAWS算法各组件对算法性能的贡献.

#### 5.3.3.1　不同到达率对算法性能的影响

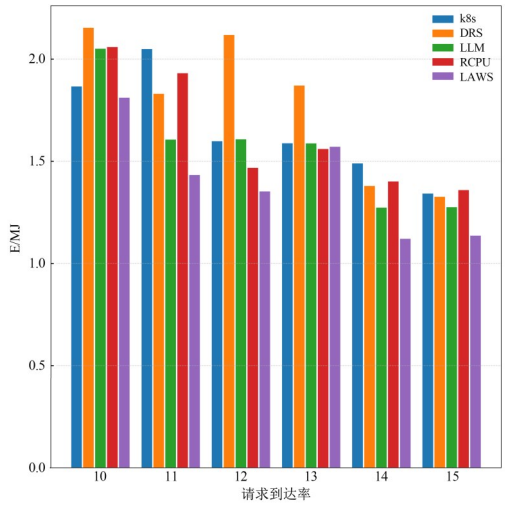
该实验假设请求到达率服从泊松分布,分别研究在不同请求到达率$\lambda$下的性能表现. 实验中,$\lambda$分别设置为10、11、12、13、14、15,实验结果如图9所示.

如图9($a$)所示,在工作流执行时延方面,LAWS算法相较于k8s默认调度策略、DRS、LLM、RCPU分别降低了49%、71%、54%、17%. 在对比算法中,k8s默认调度策略由于难以适应云边协同计算环境,导致子任务分配失衡,过度集中于云节点部署,这不仅会加剧资源竞争,还会增加通信时延,最终显著延长工作流的执行时延;DRS模型存在泛化能力不足的问题,难以习得高效的调度策略,易出现过拟合现象,从而降低子任务的执行效率;LLM算法在对云边协同环境下的工作流调度问题建模与分析方面存在不足,导致其决策能力低于LAWS算法;RCPU通过优化对资源利用率的建模有效缓解了资源竞争问题,然而其缺乏对子任务间依赖关系的深入分析,使得工作流整体完成时间仍高于LAWS算法.

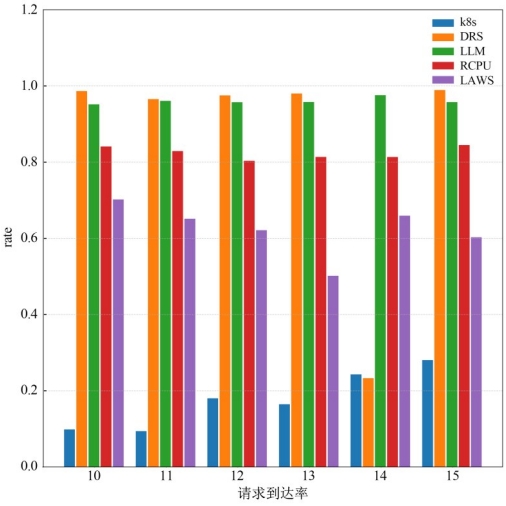
图9($b$)展示了不同调度策略下工作流的计算能耗. 随着请求到达率升高,集群闲置能耗开销降低,计算能耗呈现下降趋势. 实验表明,在LAWS调度策略下,相较于k8s默认调度策略、DRS、LLM、RCPU,计算能耗分别降低了15%、21%、10%、14%. 对比算法的计算能耗较高,其主要原因在于子任务间的资源竞争激烈. 具体而言,k8s默认调度策略因其难以适应云边协同环境,致使子任务过度集中部署于云节点,从而引发严重的资源竞争;DRS算法由于其强化学习模型存在过拟合问题,难以习得高效的调度策略,进而造成节点间子任务分配失衡,加剧了资源消耗;LLM算法受限于云边协同环境的建模能力不足,未能有效识别和缓解子任




(*a*) 工作流的平均执行时延




(*b*) 工作流的计算能耗




(*c*) 子任务在边缘节点执行比例

图9 各算法在不同请求到达率下的性能表现

务间的资源竞争;RCPU在请求到达率过大时,因调度决策发出与子任务实际占用计算资源间存在启动时延,导致决策结果与集群实际资源状态失配,进一步加剧了资源竞争,使得计算能耗增加.

图9(*c*)展示了在不同调度算法下,子任务部署位置分布. 在LAWS调度策略下,平均有62.3%的子任务被分配到边缘节点执行. 对比算法k8s、DRS、LLM、RCPU将子任务部署到边缘节点的比例分别为18%、85%、96%、82%. 分析表明,对比算法因为边缘节点子任务分配比例偏离合理区间,加剧了节点层级的资源竞争,进而导致工作流的计算能耗增加.

#### 5.3.3.2 不同分布的请求到达率对算法性能的影响

该实验设定服从泊松分布、正态分布及指数分布的请求到达率,并对比分析在不同分布的请求到达率下各算法之间的性能差异. 不同请求到达率的均值设为13,实验结果如图10所示.

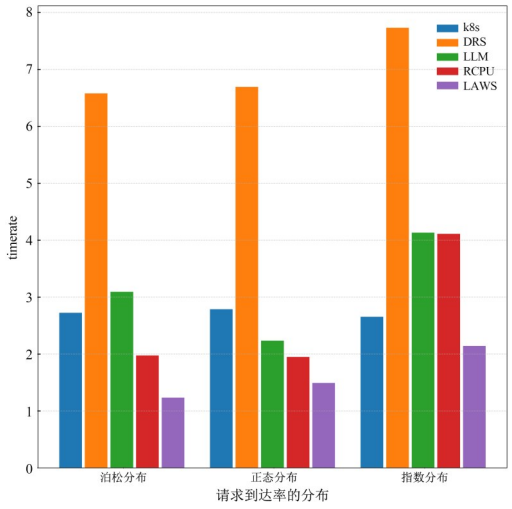
如图10(*a*)所示,在不同分布的请求到达率下,LAWS算法相较于k8s默认调度策略、DRS、LLM、RCPU,工作流执行时延分别降低了40%、77%、49%、39%. 分析表明,对比算法存在以下不足:k8s默认调度策略因难以适配云边协同环境,导致云节点上子任务间的资源竞争加剧,进而增加工作流执行时延;DRS算法仍受泛化能力不足和模型过拟合的制约,影响调度性能;LLM算法则因缺乏对集群环境的深入分析,其算法性能随请求到达率分布变化呈现波动,导致算法性能下降;RCPU算法基于集群监控数据进行调度决策,易受到启动时延影响,导致决策与实时集群资源状态失配.

同时,如图10(*b*)所示,LAWS算法相较于k8s默认调度策略、DRS、LLM、RCPU,工作流计算能耗分别降低了14%、22%、22%、11%. 具体而言:k8s默认调度策略下计算能耗的增加主要源于云节点上子任务间的资源竞争;DRS算法受到模型泛化能力的限制,在优化计算能耗方面能力有限;LLM算法受限于对云边协同环境的分析能力,其缓解资源竞争的效果弱于LAWS算法,导致计算能耗更高;RCPU算法虽优化了对资源利用率的建模,但受子任务启动时延影响,无法完全消除对剩余资源的高估问题,从而增加资源竞争带来的计算能耗.

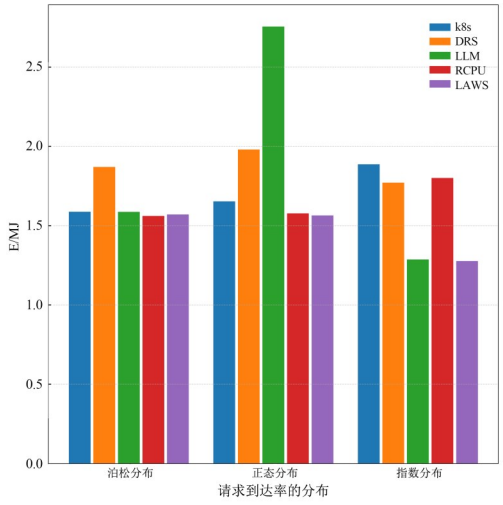
除此之外,LAWS算法将约58%的子任务调度到边缘节点,如图10(*c*)所示. 对比算法k8s、DRS、LLM、RCPU将子任务部署到边缘节点的比例分别为14%、97%、94%、82%. 该实验中,对比算法的边缘节点子任务分配比例偏离合理区间,是导致对比算法在工作流执行时延和计算能耗方面表现不佳的关键因素之一.

#### 5.3.3.3 不同类型任务对算法性能的影响

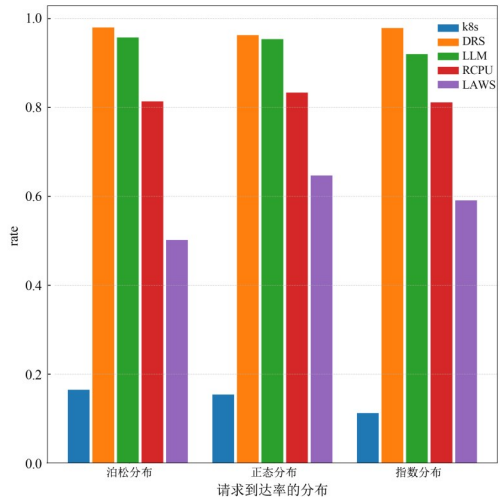
该实验设置了计算密集型和内存密集型两类工作流,以评估不同算法在调度不同计算需求工作流时的




(*a*) 工作流的平均执行时延




(*b*) 工作流的计算能耗



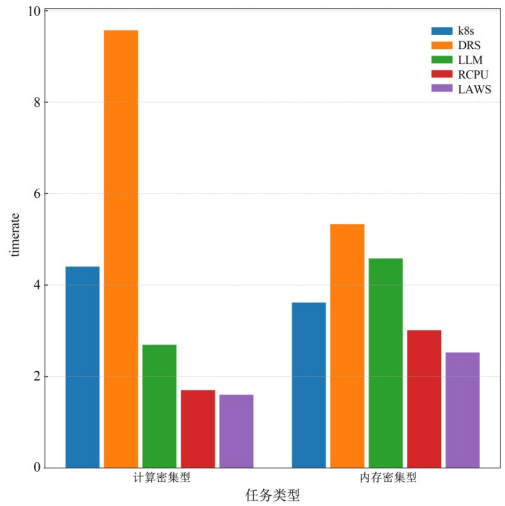

(*c*) 子任务在边缘节点执行比例

图 10　各算法在不同分布的请求到达率下的性能表现

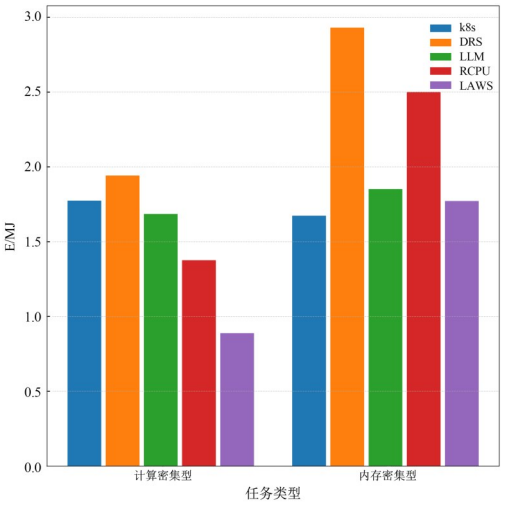
性能,实验结果如图 11 所示.

实验结果表明,在调度计算密集型工作流时,相较于 k8s 默认调度策略,LAWS 算法将工作流执行时延降低了 64%,计算能耗降低了 50%. k8s 默认调度策略下,在云节点上子任务间的资源竞争问题依然明显,导致工作流的执行时延和计算能耗大幅增加. 相较于 DRS 算法,LAWS 算法的工作流执行时延和计算能耗分别降低了 83% 和 54%. 这主要源于 DRS 算法受限于其强化学习模型泛化能力,难以有效缓解子任务间的资源竞争,进而导致性能指标上升. 相较于 LLM 算法,LAWS 算法的工作流执行时延和计算能耗分别降低了 41% 和 47%. LAWS 算法性能提升的关键在于,LAWS 算法指引大模型深入分析云边协同环境下的工作流调度问题,从而显著提升大模型在此类问题上的决策能力. 而相较于 RCPU 算法,LAWS 算法将工作流执行时延和计算能耗分别降低了 6% 和 35%. 虽然 RCPU 算法对 CPU 资源利用率的优化有效减缓了子任务间对 CPU 资源的竞争,但受子任务启动时延的影响,导致 RCPU 算法的决策结果与集群实际状态适配性不足,最终使得工作流执行时延和计算能耗高于 LAWS 算法. 在部署分布上,k8s 默认调度策略、DRS、LLM、RCPU、LAWS 将子任务部署在边缘节点的比例分别为 19%、97%、96%、83%、63%,实验结果说明,在调度计算密集型工作流时,LAWS 将子任务部署在边缘节点的比例控制在相对合理的区间,是缓解子任务资源竞争,进而降低工作流执行时延和计算能耗的关键因素之一.

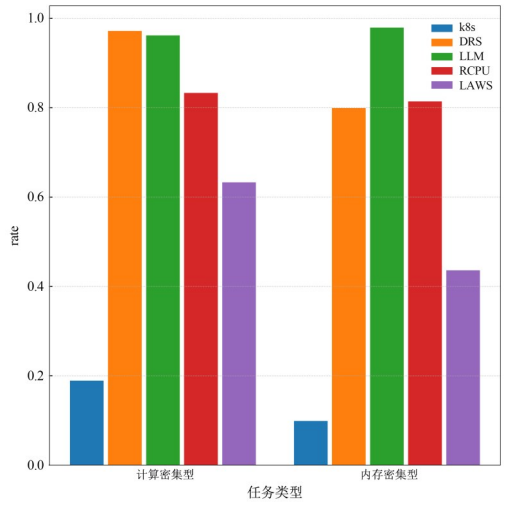
在调度内存密集型任务中,相较于 k8s 默认调度策略,LAWS 算法的工作流执行时延降低了 30%,而计算能耗仅高出 6%. 分析发现,k8s 默认策略倾向于将子任务集中部署于云节点,导致内存密集型子任务受限于云节点的内存带宽瓶颈,进而降低 CPU 资源利用率. 这种现象客观上减轻了子任务间对 CPU 资源的直接竞争强度,同时降低了工作流计算能耗,但也显著降低了子任务的执行效率,进而显著增加整体工作流的执行时延. 相较于 DRS 算法,LAWS 算法将工作流执行时延和计算能耗分别降低了 53% 和 40%. DRS 性能下降的原因在于其强化学习模型泛化能力不足,难以有效利用集群节点的内存读写带宽资源,同时子任务受到边缘节点的性能波动的影响,导致工作流的执行时延和计算能耗显著上升. 相较于 LLM 算法,LAWS 算法将工作流执行时延和计算能耗分别降低了 45% 和 4%. LAWS 算法的性能提升可归因于 LAWS 算法引导大模型对调度问题的深入分析,优化了算法对集群节点内存带宽的利用率,进而提升了子任务的执行效率,降低了计算能耗的开销. 相较于 RCPU 算法,LAWS 算法将工作流执行时延和计算能耗分别降低了 16% 和 29%. 这主要




(*a*) 工作流的平均执行时延




(*b*) 工作流的计算能耗




(*c*) 子任务在边缘节点执行比例

图 11 各算法调度不同类型工作流的性能表现

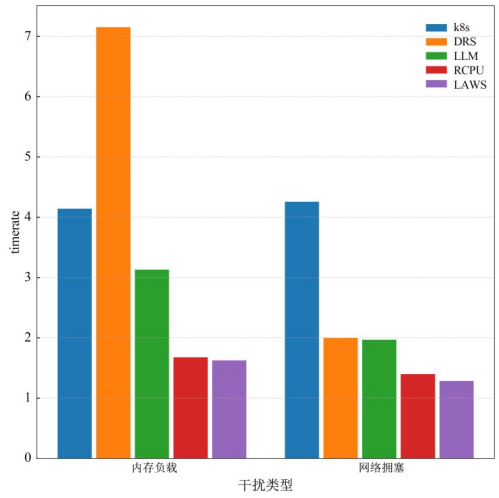
源于RCPU侧重于优化CPU使用率的建模,而缺乏对内存带宽使用率的优化分析,因此在调度内存密集型任务时表现欠佳. 在部署分布上,k8s默认调度策略、DRS、LLM、RCPU和LAWS算法将子任务部署在边缘节点的比例分别为10%、80%、98%、81%、44%. 实验结果表明,在调度内存密集型任务时,LAWS算法将子任务部署在边缘节点比例控制在相对平衡的区间,这有助于提高节点内存带宽使用率,同时减缓边缘节点性能波动对工作流执行时延的负面影响.

#### 5.3.3.4 不同类型干扰对算法性能的影响

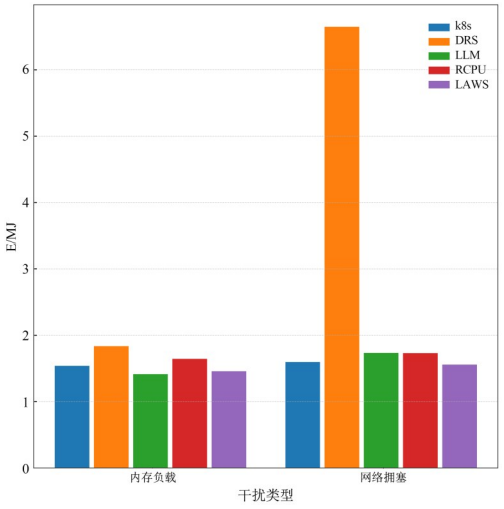
本实验通过施加内存负载和网络负载模拟边缘节点性能退化现象,以评估不同算法在多维干扰条件下的性能表现,实验结果如图12所示.

在人工注入内存负载的干扰条件下,与k8s默认调度策略相比,LAWS算法的工作流执行时延降低了61%,计算能耗降低了7%. 这主要是由于k8s默认调度策略下,云边负载不均衡的现象依然突出,导致云节点负载过高,从而引发资源竞争,致使执行时延增加并引入额外的计算能耗开销. 相较于DRS算法,LAWS算法在执行时延和计算能耗方面分别降低了77%和22%. 这表明,在内存负载干扰场景下,DRS算法未能有效学习并适应执行环境,导致工作流执行时延和计算能耗上升. 与LLM算法相比,LAWS算法在计算能耗方面仅仅增加0.8%,但工作流的执行时延显著降低了48%. 该结果表明,LAWS算法能够提升大模型对执行环境与待调度工作流的分析能力. 相比于RCPU算法,LAWS算法在执行时延和计算能耗方面分别降低了3%和13%. 这主要源于RCPU算法侧重于节点CPU使用率方面分析,而未充分考虑内存使用率及工作流图结构特性等因素,因此其性能表现不及LAWS算法. 在部署分布上,k8s默认调度策略、DRS、LLM、RCPU和LAWS算法将子任务部署在边缘节点的比例分别为19%、97%、97%、83%、68%. 实验结果说明,在内存负载干扰场景下,LAWS算法能够将边缘任务部署比例维持在相对均衡的水平,有效缓解了内存负载带来的干扰.

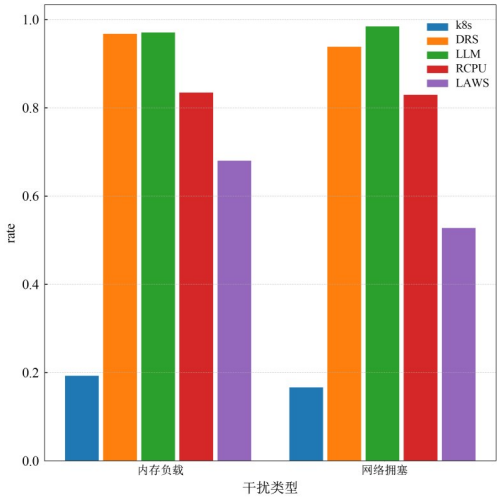
在人工注入网络负载的干扰下,相比于k8s默认调度策略,LAWS算法执行时延和计算能耗分别降低了70%和2.4%. 这主要源于k8s默认调度策略在云边协同环境中依然存在负载不均衡问题,加剧了云节点资源竞争,从而增加了工作流执行时延和计算能耗. 相比于DRS算法,LAWS算法将执行时延和计算能耗分别降低了32%和77%. 这归因于DRS算法的强化学习泛化能力不足,导致边缘节点上子任务资源竞争加剧. 相比于LLM算法,LAWS算法的执行时延降低了68%,计算能耗降低了10%. 这表明LAWS算法显著提升了大模型在人工注入网络负载的干扰场景下的上下文感知能




(*a*) 工作流的平均执行时延




(*b*) 工作流的计算能耗




(*c*) 子任务在边缘节点执行比例

图 12　各算法在不同类型干扰下的性能表现

力,并优化了资源分配,从而更好地满足用户请求. 相比于RCPU算法,LAWS算法的工作流执行时延和计算能耗分别降低了8%和10%. 这主要是由于RCPU算法未能充分考虑网络负载干扰对子任务容器下载时延的影响,导致其调度决策未达最优. 同时,k8s默认调度策略、DRS、LLM、RCPU和LAWS算法将子任务部署在边缘节点的比例分别为17%、94%、98%、83%、53%. 实验结果说明,LAWS算法能够将子任务部署在边缘节点的比例控制在平衡区间,有效降低网络负载干扰对工作流执行效率的负面影响.

#### 5.3.3.5　极端场景下各算法的性能

该实验设置边缘节点低计算能力和高干扰率两个极端模拟场景,旨在评估不同算法在极端场景下的性能表现,实验结果如图13所示.

如图13(*a*)所示,在极端环境下,LAWS算法相较于k8s默认调度策略、DRS、LLM、RCPU,工作流执行时延分别降低了40%、63%、43%、24%. 这主要归因于对比算法存在以下局限性:k8s默认调度策略未能有效利用极端环境下边缘节点的计算能力,导致执行时延显著上升;DRS的强化学习模型难以感知边缘环境的极端条件,引发边缘节点资源竞争加剧,进而导致执行时延的增加;LLM算法仅依赖于大模型的通用能力进行决策,对极端环境下的调度场景缺乏针对性感知,使得调度决策质量下降;RCPU算法则缺乏对边缘节点计算能力和不稳定性特点的分析,使得其在极端条件下的调度策略劣于LAWS算法.

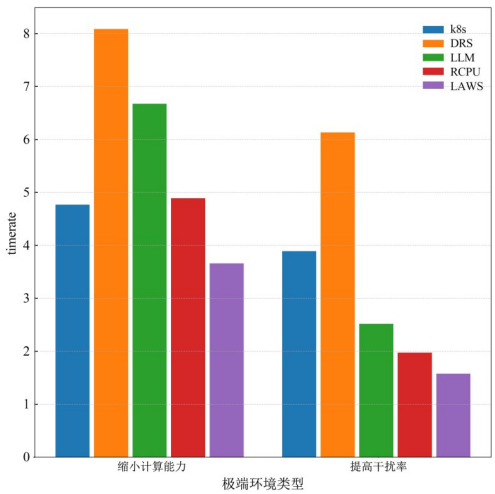
如图13(*b*)所示,LAWS算法相较于k8s默认调度策略、DRS、LLM、RCPU,计算能耗分别降低60%、66%、0.5%、27%. 这主要是因为k8s和DRS调度至边缘节点的任务,其资源需求与边缘节点计算能力较弱的特性不匹配,进而导致边缘节点上资源竞争加剧并产生额外的计算能耗;LLM算法在计算能耗优化方面与LAWS算法具有相近的能力,而RCPU算法仅从CPU利用率出发,缺乏对云边协同环境特有属性的分析,因此计算能耗反而有所增加.

如图13(*c*)所示,在极端环境下,k8s默认调度策略、DRS、LLM、RCPU和LAWS算法将子任务调度至边缘节点的比例分别为11%、98%、95%、83%、61%. 实验结果说明,LAWS算法能够有效适应调度环境的变化,动态调整部署在边缘节点的子任务比例,从而优化工作流的执行时延和计算能耗.

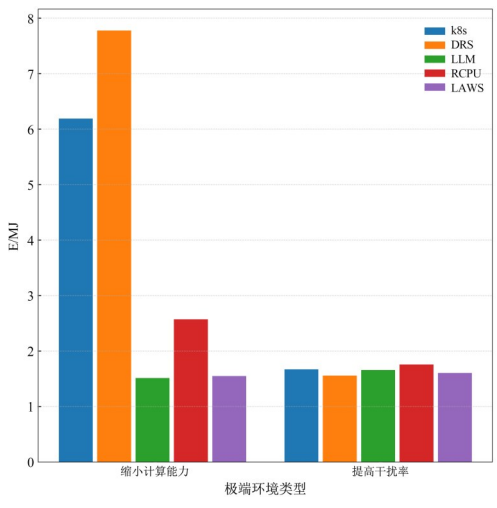
#### 5.3.3.6　不同集群规模下各算法的性能

本实验通过减少边缘服务器数量以改变集群规模,旨在评估不同规模集群下各算法的性能表现,实验结果如图14所示.

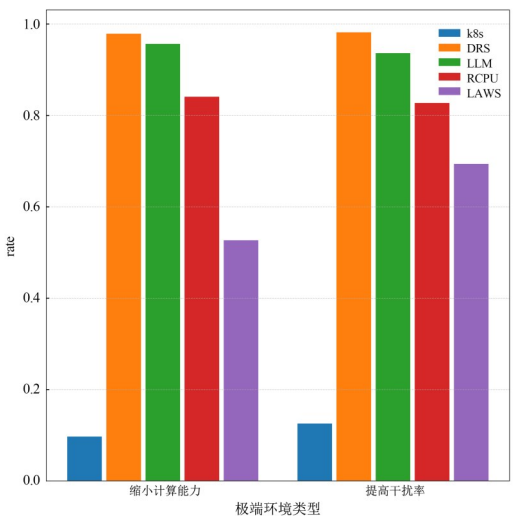
如图14(*a*)所示,相较于k8s默认调度策略,LAWS




(*a*) 工作流的平均执行时延

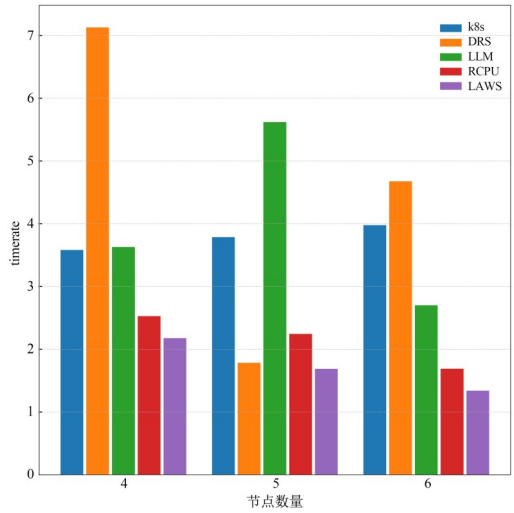



(*b*) 工作流的计算能耗

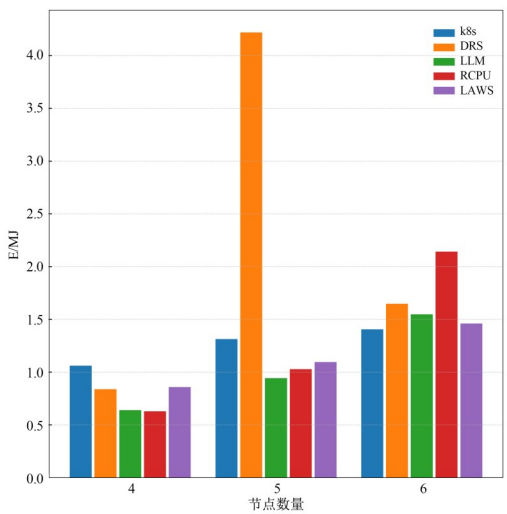



(*c*) 子任务在边缘节点执行比例

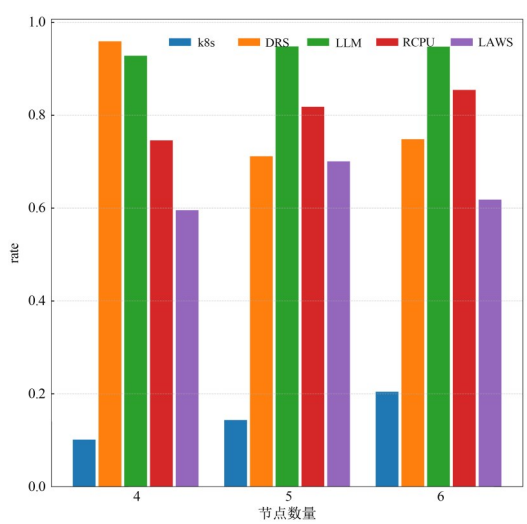
图 13 各算法在不同极端环境下的性能表现




(*a*) 工作流的平均执行时延




(*b*) 工作流的计算能耗




(*c*) 子任务在边缘节点执行比例

图 14 各算法在不同集群规模的性能表现

算法工作流执行时延降低了54%,这表明,即使缩小集群规模,k8s默认调度策略依然难以有效适应云边协同环境下的工作流调度需求. 相比于DRS算法,LAWS算法的工作流执行时延降低了62%,主要源于DRS算法的强化学习模型泛化能力不足,加剧了边缘节点上子任务间资源竞争,进而导致工作流执行时延显著增加. 相比于LLM算法,LAWS算法的工作流执行时延降低了56%,这主要得益于LAWS算法通过分解子问题引导大模型深入分析云边协同环境下的工作流调度问题,提高了大模型上下文感知的能力. 相较于RCPU算法,LAWS算法的执行时延降低了19%,原因在于RCPU算法侧重于通过分析CPU利用率来满足单个子任务的资源需求,缺乏对整体工作流执行时延的全局考量,导致执行时延上升.

如图14(*b*)所示,相较于对比算法k8s默认调度策略、DRS算法、RCPU算法,LAWS算法的计算能耗分别降低了10%、49%、10%. 这表明LAWS算法在优化资源利用和降低能耗方面具有明显优势. 这主要因为k8s默认调度策略倾向于将子任务过度集中调度至云节点,而DRS算法则过度集中调度至边缘节点,这两种策略均会导致云边服务器负载失衡和资源竞争加剧,进而产生额外的资源开销. 而相较于LLM算法,LAWS算法计算能耗增加了9%,这主要因为LAWS算法通过消耗较多的计算能耗来换取工作流执行时延的缩短. 除此之外,RCPU算法对CPU使用率建模的优化和细粒度的分析,使其在不同规模的集群环境中表现出较好的计算能耗优化效果.

如图14(*c*)所示,在不同集群规模下,k8s默认调度策略、DRS、LLM、RCPU和LAWS算法将子任务调度至边缘节点的比例分别为12%、84%、94%、78%、65%. 实验结果说明,LAWS在利用边缘资源与避免过度集中调度之间取得了更优的平衡,从而适应不同规模的集群环境.

### 5.3.4　消融实验

为了系统性地评估所提出的LAWS算法中关键组件的贡献,本文设计了一系列消融实验. 实验旨在清晰地量化知识图谱、问题分解策略和异步解耦方案各自带来的性能增益. 为此,我们设置了以下三个对比算法:

(1)Basic. 作为性能基准的下限,该模型模拟了最直接的调度方式. 它仅将当前的集群状态信息State和待调度的工作流$G$作为输入提供给大语言模型,未引入任何外部知识或复杂的推理机制.

(2)LAWS w/o DEC. 该变体在Basic模型的基础上引入了初始知识图谱InitKG,用于验证静态知识图谱对调度决策的直接影响,但它不具备后续通过问题分解动态更新思维链的机制.

(3)LAWS w/o Async. 该变体将LAWS算法中思维链PolicyKG的异步更新机制改为同步更新. 具体而言,每次工作流调度时均同步更新PolicyKG.

(4)LAWS. 该算法即为完整的LAWS算法实现.

各个算法的性能表现如图15所示.

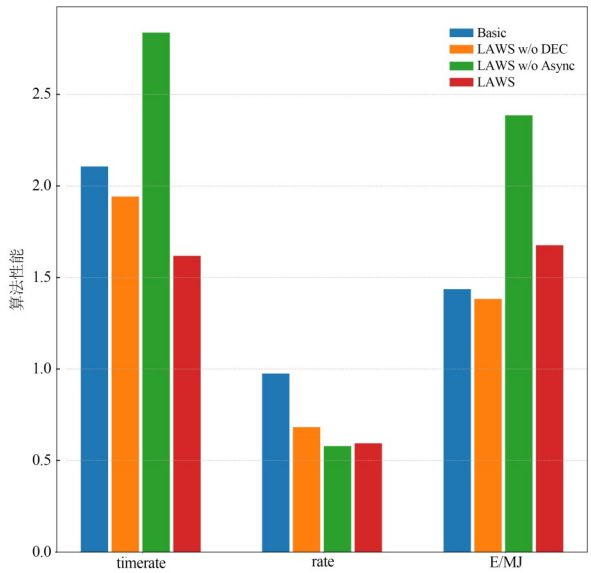



图15　LAWS算法消融实验结果

#### 5.3.4.1　基准性能

在没有引入任何领域知识和特定优化策略的情况下,Basic算法仅依赖大语言模型的通用能力进行调度. 其在运行时间比例timerate为2.1,计算能耗为1.4 MJ,并将98%的子任务调度到边缘节点. 这个性能表现作为后续所有比较的性能下限.

#### 5.3.4.2　知识图谱的贡献

在Basic算法基础上,LAWS w/o DEC算法引进本文构建的初始知识图谱InitKG. 实验结果表明,相较于Basic算法,LAWS w/o DEC的工作流执行时延降低了8%,计算能耗降低了4%. 这主要归因于InitKG为大模型提供关于工作流特性及云边协同环境调度特点的结构化知识,这些知识为大模型推理提供了依据,使其能够做出更具上下文感知能力的调度决策,从而提升了决策质量.

#### 5.3.4.3　问题分解策略的有效性

实验结果表明,相较于LAWS w/o DEC算法,LAWS算法在计算能耗增加21%的情况下,实现了工作流执行时延17%的降低. 问题分解策略通过逐步细化决策过程,引导大模型进行更深入的推理,减少其做出笼统决策的可能性. 同时异步更新的策略能够动态适应不同调度场景下的调度需求,从而进一步提升调度质量.

#### 5.3.4.4 异步更新机制的实时性

实验结果表明,相较于LAWS w/o Async算法,LAWS算法执行时延降低了43%,计算能耗降低了30%. 这主要得益于异步更新机制将耗时的PolicyKG更新步骤与实时调度决策解耦. 这种解耦有效保障了调度的实时性,使决策能够精准响应当前集群状态,从而显著提升了调度决策的质量.

## 6 结论

工作流调度问题属于NP-hard问题,在复杂的云边协同环境下更具挑战性. 现有研究在对动态变化的云边协同环境的适应性以及调度响应的实时性方面,存在局限性. 针对这些不足,本文提出了LAWS算法. LAWS算法调用大模型对调度问题进行分解,总结调度策略,并引入知识图谱表征思维链以辅助大模型推理分析. 为满足实时要求,本文提出策略更新模块与在线调度模块解耦异步执行机制. 通过与四种算法对比,LAWS算法在降低工作流执行时延和均衡云边任务部署比例方面取得显著优化效果.

### 参考文献


[1] WANG T, LU Y C, WANG J H, et al. EIHDP: Edge-intelligent hierarchical dynamic pricing based on cloud-edge-client collaboration for IoT systems[J]. IEEE Transactions on Computers, 2021, 70(8): 1285-1298.

[2] 许悦玥, 刘博文, 田臣, 等. 基于联盟链的可靠边缘计算任务卸载方法[J]. 电子学报, 2024, 52(1): 232-243.
XU Y Y, LIU B W, TIAN C, et al. Task unloading method for reliable edge computing based on alliance chain[J]. Acta Electronica Sinica, 2024, 52(1): 232-243. (in Chinese)

[3] 丁婧伊, 金嘉晖, 杨丰赫, 等. 基于云边协作的工业互联网排产方法: 以钢铁热轧生产为例[J]. 电子学报, 2024, 52(9): 2988-2999.
DING J Y, JIN J H, YANG F H, et al. Industrial Internet scheduling method based on cloud-edge collaboration: A case study of steel hot rolling[J]. Acta Electronica Sinica, 2024, 52(9): 2988-2999. (in Chinese)

[4] MA X J, XU H H, GAO H H, et al. Real-time multiple-workflow scheduling in cloud environments[J]. IEEE Transactions on Network and Service Management, 2021, 18(4): 4002-4018.

[5] SENJAB K, ABBAS S, AHMED N, et al. A survey of Kubernetes scheduling algorithms[J]. Journal of Cloud Computing, 2023, 12(1): 87.

[6] TANG X Y, CAO W B, TANG H Y, et al. Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(9): 2079-2092.

[7] SUN Z X, ZHANG B Y, GU C L, et al. ET2FA: A hybrid heuristic algorithm for deadline-constrained workflow scheduling in cloud[J]. IEEE Transactions on Services Computing, 2023, 16(3): 1807-1821.

[8] SHIN J, ARROYO D, TANTAWI A, et al. Cloud-native workflow scheduling using a hybrid priority rule, dynamic resource allocation, and dynamic task partition[C]//Proceedings of the 2024 ACM Symposium on Cloud Computing. New York: ACM, 2024: 830-846.

[9] LIAO H Y, LIU T Y, GUO J M, et al. Retrospecting available CPU resources: SMT-aware scheduling to prevent SLA violations in data centers[J]. IEEE Transactions on Parallel and Distributed Systems, 2025, 36(1): 67-83.

[10] PALLEWATTA S, KOSTAKOS V, BUYYA R. Reliability-aware proactive placement of microservices-based IoT applications in fog computing environments[J]. IEEE Transactions on Mobile Computing, 2024, 23(12): 11326-11341.

[11] XIA X W, QIU H X, XU X, et al. Multi-objective workflow scheduling based on genetic algorithm in cloud environment[J]. Information Sciences, 2022, 606: 38-59.

[12] ZHOU J J, GAO L, RAO S J, et al. Scheduling constrained cloud workflow tasks via evolutionary multitasking optimization with adaptive knowledge transfer[J]. IEEE Transactions on Services Computing, 2024, 17(6): 4254-4266.

[13] CHEN S W, YUAN Q F, LI J M, et al. Graph neural network aided deep reinforcement learning for microservice deployment in cooperative edge computing[J]. IEEE Transactions on Services Computing, 2024, 17(6): 3742-3757.

[14] LIN L D, PAN L, LIU S J. SpotDAG: An RL-based algorithm for DAG workflow scheduling in heterogeneous cloud environments[J]. IEEE Transactions on Services Computing, 2024, 17(5): 2904-2917.

[15] JAYANETTI A, HALGAMUGE S, BUYYA R. Multi-agent deep reinforcement learning framework for renewable energy-aware workflow scheduling on distributed cloud data centers[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(4): 604-615.

[16] CHEN X, HU S X, YU C J, et al. Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(3): 391-404.

[17] JIAN Z L, XIE X S, FANG Y Z, et al. DRS: A deep rein-

forcement learning enhanced Kubernetes scheduler for microservice-based system[J]. Software: Practice and Experience, 2024, 54(10): 2102-2126.
[18] WEN H, LI Y C, LIU G H, et al. AutoDroid: LLM-powered task automation in Android[C]//Proceedings of the 30th Annual International Conference on Mobile Computing and Networking. New York: ACM, 2024: 543-557.
[19] HE G L, DEMARTINI G, GADIRAJU U. Plan-then-execute: An empirical study of user trust and team performance when using LLM agents as a daily assistant[C]//Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems. New York: ACM, 2025: 1-22.
[20] FAKHOURY S, NAIK A, SAKKAS G, et al. LLM-based test-driven interactive code generation: User study and empirical evaluation[J]. IEEE Transactions on Software Engineering, 2024, 50(9): 2254-2268.
[21] GU Q H. LLM-based code generation method for golang compiler testing[C]//Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2023: 2201-2203.
[22] ALI J, MAQSOOD T, KHALID N, et al. Communication and aging aware application mapping for multicore based edge computing servers[J]. Cluster Computing, 2023, 26(1): 223-235.
[23] SCHROEDER B, GIBSON G A. A large-scale study of failures in high-performance computing systems[C]//International Conference on Dependable Systems and Networks. Piscataway: IEEE, 2006: 249-258.
[24] CHEN X J, JIA S B, XIANG Y. A review: Knowledge reasoning over knowledge graph[J]. Expert Systems with Applications, 2020, 141: 112948.
[25] GAUTIER G, POLITO G, BARDENET R, et al. DPPy: Sampling DPPs with Python[EB/OL]. (2019-08-12)[2025-08-30]. https://arXiv.org/abs/1809.07258.
[26] HUANG L, YU W J, MA W T, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions[J]. ACM Transactions on Information Systems, 2025, 43(2): 1-55.
[27] RAMPRASAD S, FERRACANE E, LIPTON Z. Analyzing LLM behavior in dialogue summarization: Unveiling circumstantial hallucination trends[C]//Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics. Stroudsburg: ACL, 2024: 12549-12561.
[28] DENIS C, HEBIRI M, ZAOUI A. Regression with reject option and application to kNN[EB/OL]. (2021-03-05) [2025-08-30]. https://arXiv.org/abs/2006.16597.

## 作者简介

**黎广镕**　男,2004年2月出生于广东省廉江市. 现为暨南大学信息科学技术学院软件工程专业本科生.
E-mail: lgr2172@stu2022.jnu.edu.cn

**吴文泰**　男,1992年11月出生于广东省广州市. 英国华威大学计算机科学博士、暨南大学信息科学技术学院副教授. 主要研究方向为并行与分布式计算、分布式机器学习以及节能计算.
E-mail: wentaiwu@jnu.edu.cn

**李广军**　男,2002年7月出生于贵州省遵义市. 现为暨南大学人工智能专业硕士研究生. 主要研究方向为大语言模型、机器学习系统.
E-mail: lniox_1103@163.com

**王泽平**　男,1999年4月出生于贵州省凯里市. 现为暨南大学信息科学技术学院博士研究生. 主要研究方向为云计算、算力网络、人工智能和算力拍卖.
E-mail: WangZP@stu2025.jnu.edu.cn

**尚　晶**　女,1978年1月出生于河北省沧州市. 博士,中国移动信息技术中心高级工程师,中国移动首席专家. 主要研究方向为大数据、云计算、数据库.
E-mail: shangjing@chinamobile.com

**龙赛琴**　女,1986年8月出生于湖南省益阳市. 华南理工大学计算机应用技术博士、中国计算机学会成员、暨南大学信息科学技术学院教授. 主要研究方向为云计算、云存储、并行和分布式系统、文件系统以及计算机系统架构.
E-mail: saiqinlong@jnu.edu.cn