doi: 10.7690/bgzdh.2026.10.008

基于策略感知 RAG 的任务链协同调度方法

张广庆[1]，李荣宽[1]，何重阳[1]，王翔意[2]，吴　聪[2]
(1. 电科云(北京)科技有限公司云产品事业部，北京 100041；2. 南京邮电大学通信与信息工程学院，南京 210003)

摘要：针对现有调度器以最小可部署计算单元(Pod)或单个任务(Job)为决策边界，难以统筹跨 Job 依赖与资源竞争等问题，提出基于策略感知检索增强生成的任务链协同调度方法。统一建模任务链与集群状态，将调度策略转化为可检索、可组合的结构化知识，采用逆向资源预留与正向拓扑调度相结合的双向规划机制生成跨 Job 调度方案，并通过程序化验证剔除不可执行结果。实验结果表明，该方法在保持目标策略调度倾向的同时，能够尽可能消除高负载下的悬决(Pending)任务，显著提升任务链整体成功率，为边端异构环境下的任务链调度提供了可行路径。

关键词：任务链调度；检索增强生成；策略感知；边端协同
中图分类号：TP393　　**文献标志码：**A

A Collaborative Task-chain Scheduling Method Based on Strategy-aware RAG

Zhang Guangqing[1], Li Rongkuan[1], He Chongyang[1], Wang Xiangyi[2], Wu Cong[2]
(1. *CETC Cloud* (*Beijing*) *Technology Co*., *Ltd*., *Beijing* 100041, *China*; 2. *School of Communications and Information Engineering*, *Nanjing University of Posts and Telecommunications*, *Nanjing* 210003, *China*)

Abstract: To address the limitations of existing schedulers that use individual Pods or Jobs as the scheduling decision boundary and therefore cannot effectively coordinate inter-Job dependencies and resource competition, this paper proposes a task-chain collaborative scheduling method based on policy-aware retrieval-augmented generation. The method uniformly models task chains and cluster states, and transforms scheduling policies into structured knowledge that can be retrieved and composed. A bidirectional planning mechanism combining backward resource reservation with forward topology-aware scheduling is employed to generate cross-Job scheduling plans, while programmatic validation is introduced to eliminate infeasible results. Experimental results demonstrate that the proposed method preserves the scheduling tendencies of the target policy while minimizing Pending tasks under high-load conditions, thereby significantly improving the overall success rate of task chains. This work provides a feasible approach to task-chain scheduling in heterogeneous edge-terminal environments.

Keywords: task-chain scheduling; retrieval-augmented generation; strategy awareness; edge-device collaboration

0　引言

随着智能感知、无人平台和边缘计算在国防领域的深化应用，协同侦察、多源感知和联合研判等业务正由单节点处理转向边缘节点与端侧设备协同执行[1]。此类业务通常由多个具有独立生命周期和差异化 CPU、内存、GPU 需求的任务(Job)构成，并通过执行与数据依赖形成任务链；任一关键 Job 因资源或通信条件不足而阻塞，均可能导致完整业务失败[2]。因此，调度目标应由单个 Job 可执行拓展为任务链整体可执行。

现有调度器主要以计算单元(Pod)或单个 Job 为决策单元[3]；编组(Gang)调度能够保证单个 Job 或调度组(PodGroup)达到最小可运行规模[4]，有向无环图(directed acyclic graph，DAG)调度也可在异构环境中协调多个工作流[5]，但多个独立提交的 Job 仍常被分散决策。在高负载和异构资源条件下，前置 Job 可能占用后续关键 Job 所需的稀缺节点，并行分支也可能竞争同一资源窗口，造成局部调度成功而完整任务链不可执行。

针对依赖任务，异构最早完成时间(heterogeneous earliest finish time，HEFT)依据任务优先级与最早完成时间进行异构调度[6]；GRAPHENE 兼顾依赖关系与资源装箱，提高集群吞吐量[7]；云边工作流联合优化及面向云边连续体的 DAG 编排进一步将容器部署、节点资源和通信开销纳入调度[8-9]。此外，动态资源调度研究通过刻画任务与资源扰动，并以调度收益和方案变更为目标增强方案的动态适应能力[10]。上述工作改善了特定场景下的完工时间、吞吐量或资源利用率，但尚缺少以多个独立 Job 组成的完整任务链为统一决策边界、

收稿日期：2025-02-07；**修回日期：**2025-03-12
基金项目：预研专用技术项目(315057209)
第一作者：张广庆(1988—)，男，安徽人，硕士。

在前序 Job 调度前显式预留后续瓶颈资源的机制。

另一方面，任务链结构、集群负载和业务目标的变化，对资源聚合、负载均衡、站点亲和及通信优化提出差异化策略需求。Tiresias 利用作业服务量与 GPU 放置优化深度学习集群调度[11]；Decima 以图神经网络和强化学习自动生成 DAG 调度策略[12]；LarS 引入大语言模型提高云任务调度的跨场景泛化能力[13]。专用策略依赖特定负载，学习型策略又将知识隐含于模型参数或训练轨迹，均不利于直接解释、复用和组合，检索增强生成(retrieval-augmented generation，RAG)可借助外部知识提高生成的事实性与可追溯性[14]，但仍可能产生任务遗漏、节点不存在或资源超配等幻觉[15]，难以直接保证方案可执行。

针对上述问题，笔者提出基于策略感知 RAG 的任务链协同调度方法，主要工作：

1) 将调度决策边界由单个 Pod 或 Job 扩展至完整任务链，统一建模跨 Job 依赖、资源、站点和通信约束，采用逆向资源预留与正向拓扑调度相结合的双向规划机制，使前序节点选择服从后续瓶颈 Job 的资源需求。

2) 将调度器源码、插件文档和策略规则转化为包含适用条件、过滤边界、评分偏好和状态更新规则的策略原语，结合用户策略元语义与任务链场景形成可检索、可组合的调度意图。

3) 构建程序化验证与迭代重调度机制，对任务完整性、节点真实性、累计资源、依赖关系和策略硬规则进行确定性校验；基于 Volcano 和无需 Kubelet 的 Kubernetes(kubernetes without kubelet，KWOK)开展高负载实验，从策略行为对齐与任务链整体可执行性 2 方面验证方法有效性。

1　方法设计

笔者提出的方法接收任务链拓扑、资源需求、业务约束、集群状态和用户策略，输出完整调度方案及任务链状态。总体流程如图 1 所示。

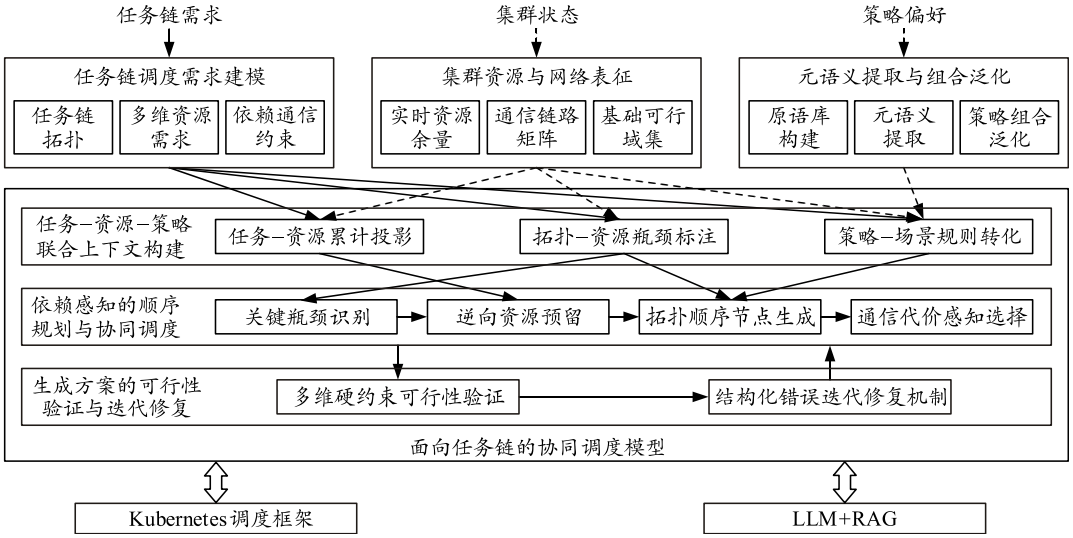

图 1　基于策略感知 RAG 的任务链协同调度方法总体流程

1.1　边端多任务场景与任务链定义

边端异构集群中，节点在资源容量、角色标签、站点归属和网络条件等方面存在显著差异[9]，而完整业务则由多个具有独立资源需求和生命周期的 Job 组成，并通过执行与数据依赖形成任务链[10]，调度目标不仅是完成单个 Job 调度，还需保证跨 Job 联合调度后的任务链整体可执行。

笔者以完整任务链为调度对象。设任务链编号为 c，调度时刻为 t，则方法输入为：

$$I_c(t)=\langle Q_c, S(t), p\rangle 。 \tag{1}$$

式中：Q_c 为任务链调度需求模型；$S(t)$为时刻 t 的集群状态；p 为用户指定的调度策略。三者分别描述调度对象、资源可行边界和节点选择倾向。

笔者根据任务链需求与集群状态确定基础可行域，再检索策略知识并形成调度意图；通过累计资源投影识别关键 Job，采用逆向资源预留、正向拓扑调度和通信代价感知选择生成跨 Job 方案，经程序化验证后提交。

对于通过输入和拓扑合法性检查的请求，方法输出为：

$$O_c=\langle Z, \mathrm{stat}_c\rangle,\ \mathrm{stat}_c\in\{\mathrm{Scheduled}, \mathrm{Pending}\} 。 \tag{2}$$

式中：stat_c 为任务链状态；若 stat_c=Scheduled，则

$Z=[z_{ij}]$为完整调度决策矩阵；$z_{ij}=1$ 为 Job J_i 调度至节点 n_j；若当前状态下不存在完整可行方案，则 $\text{stat}_c=$ Pending 且 $Z=\varnothing$，不提交局部结果。

1.2 任务链调度需求建模

任务链调度需求建模旨在将用户提交的多个独立 Job 及其业务关系转化为结构化的调度输入。该过程需要同时刻画任务链拓扑结构、各 Job 的资源需求、节点属性约束以及依赖边上的通信与站点关系，为后续候选域构建、瓶颈识别和可行性验证提供统一的数据基础。

为统一表示多个独立 Job 及其业务关系，任务链 c 的调度需求模型定义为：

$$Q_c=\langle G_c, D_c, C_c, R_c\rangle,\ G_c=(J_c, E_c)。\tag{3}$$

式中：$G_c=(J_c, E_c)$为任务链拓扑；$J_c=\{J_1, J_2, \cdots, J_I\}$为 Job 集合；$(J_i, J_k)\in E_c$ 为 Job J_k 依赖 Job J_i；$D_c=[d_{ir}]\in R_+^{I\times D}$ 为资源需求矩阵，d_{ir} 为 Job J_i 对第 r 类资源的需求，资源维度集合 $D=\{1, 2, \cdots, D\}$包含 CPU、内存和 GPU，即 $D=3$；C_c 为角色、能力标签和固定站点等 Job 约束；R_c 为相对站点、最大时延和最小带宽等依赖关系约束。未指定项不参与硬性过滤。

为避免下标含义混淆：i 和 k 表示 Job 索引；j 和 q 表示节点索引；r 表示资源维度；h 表示策略原语索引；s 表示规划步。2 个维度的下标直接连写，例如 z_{ij} 表示 Job J_i 与节点 n_j 之间的调度决策，d_{ir} 表示 Job J_i 对第 r 类资源的需求，R_{ik} 表示 Job J_i 与 J_k 之间的关系判定。

建模阶段首先检查 G_c 是否为有向无环图。对于合法任务链，Pre(i)表示 Job J_i 的直接前置任务集合，PreAll(i)表示所有可沿有向路径到达 J_i 的前置任务集合；前者确定正向调度顺序，后者用于逆向传播资源预留要求。

进一步计算 Job 的归一化拓扑层级 $l_i\in[0, 1]$和汇聚程度 $k_i\in[0, 1]$。l_i 由 Job 拓扑层级归一化得到，取值越大表示越接近下游；k_i 由直接前置任务数量归一化得到，取值越大表示汇聚分支越多。无拓扑层级或汇聚关系时相应值取 0。二者仅用于关键 Job 识别。

定义 $F_i(n_j)$为 Job J_i 与节点 n_j 的角色、标签和固定站点匹配结果，$R_{ik}(n_j, n_q, t)$为依赖 Job 分别调度到节点 n_j、n_q 时的站点与链路约束满足结果；满足时取 1，否则取 0。

设 N 为集群节点集合，$a_{jr}(t)$为节点 n_j 的第 r 类可用资源，$Z=[z_{ij}]$为候选调度决策矩阵，则任务链整体可执行性为：

$$\Phi_c(Z,t)=v_u(Z)\cdot v_r(Z,t)\cdot v_m(Z)\cdot v_d(Z,t)。\tag{4}$$

式中 v_u、v_r、v_m 和 v_d 均为二值判定量，分别表示所有 Job 均对应唯一调度节点、节点累计资源未超限、任务与节点属性匹配及全部依赖边满足站点与链路约束。具体而言，对任意 Job J_i 均应满足 $\sum_{n_j\in N} z_{ij}=1$；对任意节点 n_j 和资源维度 r 均应满足 $\sum_{J_i\in J_c} d_{ir}z_{ij}\leqslant a_{jr}(t)$；对任意 Job 与节点组合均应满足 $z_{ij}\leqslant F_i(n_j)$；对任意依赖边(J_i, J_k)，若 $z_{ij}=1$ 且 $z_{kq}=1$，则实际节点组合应满足 $R_{ik}(n_j, n_q, t)=1$，当 4 项均取 1 时，$\Phi_c(Z, t)=1$，任务链整体可执行。

1.3 集群资源与网络状态表征

集群资源与网络状态表征负责获取调度时刻边端异构集群的实时资源视图与网络视图。由于边端场景中节点资源容量、站点归属和链路质量均存在显著差异，且这些状态在调度过程中随任务提交和资源释放动态变化，系统需在每次重调度前刷新可用资源矩阵、节点属性以及节点间时延和带宽信息，从而为任务链调度提供准确的资源侧可行边界。

调度时刻 t 的集群状态为：

$$S(t)=\langle N, \boldsymbol{A}(t), M, Y, L(t), B(t)\rangle。\tag{5}$$

式中：$\boldsymbol{A}(t)=[a_{jr}(t)]$为节点可用资源矩阵；$M$ 记录节点角色和能力标签；$\boldsymbol{Y}=[Y_j]$为节点站点归属向量，Y_j 表示节点 n_j 所属站点；$\boldsymbol{L}(t)$和 $\boldsymbol{B}(t)$分别为节点间时延和带宽矩阵，其元素 $L(n_j, n_q, t)$和 $B(n_j, n_q, t)$表示时刻 t 节点 n_j 与 n_q 之间的链路时延和带宽。可用资源与网络状态均在每次重调度前刷新。

基础匹配函数 $F_i(n_j)$由 Job 约束 C_c、节点角色与能力标签 M 以及站点标识 Y_j 共同确定。在此基础上结合资源需求，构建 Job J_i 的基础候选域：

$$\Omega_i(t)=\{n_j\in N\mid F_i(n_j)=1, a_{jr}(t)\geqslant d_{ir}, \forall r\in D\}。\tag{6}$$

式中 $\Omega_i(t)$为 Job 在当前状态下能够独立调度的节点，不包含多个 Job 共置后的累计资源影响。若任一必要 Job 的基础候选域为空，则当前任务链无法形成完整方案。

对于依赖边(J_i, J_k)，跨 Job 关系函数 $R_{ik}(n_j, n_q, t)$由依赖关系约束 R_c、站点标识 Y_j 与 Y_q 以及时刻 t 的链路时延和带宽共同确定，据此构建节点对可行域：

$\Gamma_{ik}(t)=\{(n_j,n_q)\in\Omega_i(t)\times\Omega_k(t)\mid R_{ik}(n_j,n_q,t)=1\}$。(7)

式中 $\Gamma_{ik}(t)$ 为依赖边的站点、时延和带宽硬约束，若任一必要依赖边的节点对可行域为空，则完整任务链不可调度；$\Omega_i(t)$与 $\Gamma_{ik}(t)$ 分别为单 Job 和依赖 Job 对的基础可行域，后续规划不得越过该边界。

1.4　策略元语义提取与组合泛化

笔者将策略知识划分为 3 个层次：策略原语是从策略实现中提取的最小可检索单元；策略元语义描述用户策略的目标和边界；策略调度意图是结合当前任务链场景形成的规则与偏好。三者分别对应离线知识、在线查询和推理输出。

1.4.1　策略原语库构建

离线阶段解析调度器源码、插件文档和策略规则，将候选过滤、评分方向、资源权重、冲突处理及连续调度时的状态更新逻辑拆解为策略原语。策略原语库记为 $K=\{k_1, k_2, \cdots, k_K\}$，$K$ 为原语数量。第 h 个原语 k_h 包含策略标识 id_h、适用条件 cond_h、过滤边界 f_h、软偏好 b_h 和状态更新规则 o_h。

每个原语同时保存可追溯的源码或规则证据、语义向量 $\boldsymbol{v}_h$ 和结构化元数据。元数据用于按策略标识和作用类型过滤；语义向量用于场景匹配，原语库只保存调度原则，不保存历史任务、节点名称和调度结果，因而检索对象是策略知识而非历史案例。

1.4.2　用户策略元语义提取

在线阶段解析用户提供的策略标识、实现代码或明确说明，形成策略元语义：

$$m_p=E_{\mathrm{meta}}(p)=\langle a_p,w_p,c_p,f_p,o_p\rangle。\tag{8}$$

式中：$E_{\mathrm{meta}}(\bullet)$为元语义提取过程；$a_p$ 为资源聚合、分散或均衡倾向；w_p 为资源维度权重；c_p 为通信与站点偏好；f_p 为过滤边界和冲突处理顺序；o_p 为连续调度时的策略状态更新要求。已覆盖策略直接读取预存元数据；自定义策略则从用户提供的实现或明确说明中提取上述信息。若 2 类信息均缺失，系统将策略标记为信息不足，不根据未知名称推测调度意图。

1.4.3　基于原语义的组合泛化

任务链拓扑、资源需求、候选节点和可行节点对共同编码为场景查询向量 $\boldsymbol{q}_c$。当用户指定的策略已被原语库覆盖时，系统先按策略标识精确过滤，再依据 $\mathrm{cond}_h(\boldsymbol{q}_c)$选择当前场景适用的原语，所得集合记为 K_{pc}。适用的过滤边界全部保留，评分偏好按场景相关度排序，避免因截断遗漏策略硬规则。

当策略尚未被原语库覆盖时，系统同时比较场景语义相似度与策略结构兼容度。第 h 个原语的检索得分定义为：

$$\mathrm{Rel}(k_h)=\alpha\mathrm{sim}(q_c,v_h)+(1-\alpha)\mathrm{comp}(m_p,k_h)。\tag{9}$$

式中：$\mathrm{Rel}(k_h)$为原语 k_h 的综合相关度；$\mathrm{sim}(\bullet)$为场景语义相似度；$\mathrm{comp}(\bullet)$为策略元语义兼容度，$\alpha\in[0,1]$为平衡系数。系统按 $\mathrm{Rel}(k_h)$选择相关原语，组成检索结果 K_{pc}。未覆盖策略的组合必须以用户实现或明确说明为依据；仅由相似性获得的内容只能形成软偏好，不能提升为硬规则。

2 条路径统一形成策略调度意图：

$$g_{pc}=\langle H_{pc},B_{pc},\lambda_{pc},U_{pc}\rangle=F_{\mathrm{intent}}(q_c,m_p,K_{pc})。\tag{10}$$

式中：pc 为面向任务链 c，依据策略 p 形成的场景化结果；$F_{\mathrm{intent}}(\bullet)$为受检索证据和用户元语义约束的调度意图推理过程；$H_{pc}$ 为候选过滤，由具有明确证据的 f_h 与 f_p 形成；B_{pc} 为单节点排序，由 b_h、a_p、w_p 及 c_p 中的站点偏好共同确定；$\lambda_{pc}=\langle\lambda_L,\lambda_B,\mu\rangle_p$ 为依赖节点对比较，由通信偏好确定；U_{pc} 为连续决策状态更新，由 o_h 与 o_p 形成，策略调度意图不包含具体节点。

1.5　面向任务链的协同调度模型

协同调度模型以 $I_c(t)$和 g_{pc} 为输入，输出完整调度决策矩阵 $\boldsymbol{Z}$。任务与资源硬约束限定基础可行域，策略硬规则进一步过滤候选节点，策略软偏好用于可行方案排序。模型先联合分析任务、资源与策略，再执行逆向资源预留和正向拓扑调度。

1.5.1　任务-资源-策略联合上下文构建

1) 任务-资源累计投影。设 s 为规划步，$\boldsymbol{Z}(s)=[z_{ij}(s)]$为当前部分调度决策矩阵；已规划将 Job J_i 调度到节点 n_j 时 $z_{ij}(s)=1$，否则取 0。节点的资源投影余量定义为：

$$e_{jr}(s)=a_{jr}(t)-\sum\nolimits_{J_i\in J_c}d_{ir}z_{ij}(s)。\tag{11}$$

式中 $e_{jr}(s)$为规划到第 s 步后节点 n_j 的第 r 类资源投影余量。初始部分调度决策矩阵 $\boldsymbol{Z}(0)$全为 0，此时 $e_{jr}(0)=a_{jr}(t)$。任一资源维度小于 0 均表明部分方案超配，该分支不再扩展。投影余量随后用于安全窗口和有效候选域计算。

2) 拓扑-资源瓶颈标注。对基础候选域非空的

Job，候选节点数量和需求占可用资源的比例共同反映其资源压力：

$$\rho_i=\sum\nolimits_{r\in D}\omega_r\frac{d_{ir}|\ \Omega_i(t)|}{\sum\nolimits_{n_j\in\Omega_i(t)}a_{jr}(t)+\varepsilon}+\frac{\omega_o}{|\ \Omega_i(t)|+\varepsilon}。\quad(12)$$

式中：ρ_i 为 Job J_i 的资源压力；ω_r 为瓶颈识别中第 r 类资源的权重；ω_o 为候选节点数量项权重；ε 为避免分母为 0 的极小正数。各资源量在计算前归一化，权重均为非负数，并满足 $\sum_{r\in D}\omega_r+\omega_o=1$ 与策略元语义中的资源偏好权重相互独立。第一项比较 Job 需求与候选节点平均可用量，第二项反映可替代节点数量；需求比例越高且候选节点越少，ρ_i 越大。

Job J_i 的安全资源窗口为：

$$W_i(s)=\{n_j\in\Omega_i(t)\,|\,e_{jr}(s)\geqslant d_{ir},\forall r\in D\}。\quad(13)$$

式中 $W_i(s)$ 为扣除已规划 Job 占用后，仍能完整承载 J_i 的节点集合。资源压力与前述拓扑特征共同用于识别关键 Job，安全资源窗口则构成逆向资源预留的边界。

3) 策略−场景规则转化。将调度意图实例化到当前任务链和集群状态。策略硬规则集合 H_{pc} 转换为二值判定 $H_{pc}(J_i,\ n_j)$：Job 与节点满足全部规则时取 1，否则取 0；硬规则集合为空时默认取 1。对软偏好集合 B_{pc} 中的单节点偏好归一化至[0, 1]，依据 w_p 与原语元数据确定权重并求加权平均，得到策略得分 $S_{pc}(J_i,\ n_j)$，得分越大表示策略越倾向该调度方式，软偏好集合为空时取 0。通信偏好由 λ_{pc} 转换为链路代价权重，U_{pc} 则规定连续调度后需要刷新的策略状态。任务与资源硬约束始终具有最高优先级，策略规则不得扩大 $\Omega_i(t)$ 或 $\Gamma_{ik}(t)$。

1.5.2 依赖感知的顺序规划与协同调度

联合上下文构建后，模型先由关键资源瓶颈向前置任务传播预留要求，再按正向拓扑顺序生成节点，逆向过程只调整候选域，不提交任务，正向过程在预留边界内确定调度节点，使局部节点选择服从完整任务链的资源需求。

1) 关键瓶颈识别。关键 Job 需要同时反映资源压力和拓扑影响。综合 ρ_i、归一化拓扑层级 l_i 与汇聚程度 k_i，定义瓶颈强度：

$$\sigma_i=\rho_i+\omega_l l_i+\omega_f K_i。\quad(14)$$

式中 ω_l 和 ω_f 分别为拓扑层级和汇聚程度权重，与式(12)中的资源权重均取非负值。模型从尚未规划的 Job 中选择 σ_i 最大者作为关键 Job J_b；若出现并列，则依次比较拓扑层级、候选节点数量和 Job 编号。关键 Job 不必优先执行，但在其完成调度前必须持续保留完整资源窗口；完成后，从剩余 Job 中选择新的关键 Job。

2) 逆向资源预留。设 J_s 为第 s 步前已完成规划的 Job 集合，初始时 $J_0=\varnothing$。模型从 J_b 出发，按反向拓扑顺序遍历 PreAll(b)，将安全窗口非空要求传播给其全部前置任务。对于任一尚未规划 Job J_i 的候选节点 n_j，先在当前投影余量中临时扣除其各类资源需求 d_{ir}，随后重算关键 Job 的资源窗口，剔除不满足 $H_{pc}(J_b,\ n_q)=1$ 的节点，以及无法与已调度直接前置任务构成可行节点对的节点；若 $J_i\in\mathrm{Pre}(b)$，还需检查 $(n_j,n_q)\in\Gamma_{ib}(t)$。$W_b(J_i,n_j)$ 表示假设将 J_i 调度到 n_j 后，关键 Job J_b 仍可使用的有效安全窗口。并行分支虽不属于 PreAll(b)，仍可能竞争同一节点，因此在关键 Job 调度前同样接受窗口非空检查。有效候选域为：

$$V_i(s)=\{n_j\in\Omega_i(t)\,|\,e_{jr}(s)\geqslant d_{ir},\ \forall r\in D,\ H_{pc}(J_i,n_j)=1,\ J_i=J_b\vee|\ W_b(J_i,n_j)|\geqslant 1\}。\quad(15)$$

式中：$V_i(s)$ 同时满足当前资源余量、策略硬规则和关键 Job 预留要求；J_b 自身可直接使用安全窗口。关键 Job 完成调度后，模型从剩余 Job 中重新选择 J_b，并据此更新预留条件。该过程既约束依赖前置任务，又防止并行任务提前消耗关键资源。

3) 拓扑顺序节点生成。记 A_s 为尚未规划且其直接前置任务均属于 J_s 的 Job 集合。若某个就绪 Job 的有效候选域为空，当前部分方案需要回退；否则优先选择瓶颈强度高且候选节点少的 Job：

$$u=i:J_i\in A_s\left[\sigma_i+\eta/(|V_i(s)|+\varepsilon)\right]。\quad(16)$$

式中：u 为第 s 步选中的 Job 索引；η 为候选空间紧迫度权重。由于 J_u 的直接前置任务已完成调度，还需利用节点对可行域过滤其候选节点。为直观表示已有调度结果，定义 $N(J_k)$ 为直接前置任务 J_k 已确定的调度节点；$\Gamma_{ku}(t)$ 个 Job 下标依次表示前置 Job J_k 和当前 Job J_u，则：

$$\Lambda_s=\{n_j\in V_u(s)\,|\,(N(J_k),n_j)\in\Gamma_{ku}(t),\forall J_k\in\mathrm{Pre}(u)\}。\quad(17)$$

若 J_u 没有直接前置任务，则 $\Lambda_s=V_u(s)$；若 Λ_s 为空，则回退并更换此前节点组合。至此，拓扑顺序、资源预留与依赖边硬约束被统一到同一候选域中。

通信代价感知的成对选择。$\Gamma_{ki}(t)$ 已过滤不满

足站点、时延和带宽要求的节点对，模型只在剩余组合中比较通信代价。当前 Job 的候选节点为 n_j、直接前置任务 J_k 的调度节点为 $N(J_k)$时，2 节点间的通信代价为：

$$C_k(n_j,t)=\lambda_L\frac{L(N(J_k),n_j,t)}{L_{\max}+\varepsilon}+\lambda_B\left(1-\frac{B(N(J_k),n_j,t)}{B_{\max}+\varepsilon}\right)。\quad(18)$$

式中：$C_k(n_j, t)$为直接前置任务 J_k 的调度节点与候选节点 n_j 之间的通信代价；$L_{\max}$ 和 $B_{\max}$ 分别为当前可行链路的最大时延和最大带宽；λ_L、λ_B 来自 λ_{pc}，均为非负数且和为 1。归一化后的 $C_k(n_j, t)$位于[0, 1]，数值越小表示通信条件越好。结合单节点策略得分，目标节点选择为：

$$j_s=n_j\in\Lambda_s[S_{pc}(J_u,n_j)-\mu\sum_{J_k\in Pre(u)}C_k(n_j,t)]。\quad(19)$$

式中：J_s 为第 s 步选中的目标节点索引；μ 为通信代价相对策略得分的权重，无直接前置任务时，通信项取 0。确定节点后，将调度关系 $J_u\rightarrow n_{j_s}$ 写入 $Z(s+1)$，其余已确定元素保持不变，并将 J_u 加入已规划 Job 集合 J_{s+1}。随后从目标节点各资源余量中扣除当前 Job J_u 的第 r 类资源需求 d_{ur}，更新安全窗口、有效候选域和受 U_{pc} 响的策略得分。上述过程循环执行，直至生成完整调度决策矩阵 $\boldsymbol{Z}$。

1.5.3　生成方案的可行性验证与迭代重调度

生成结果采用固定结构，至少包含任务链标识、Job 标识、依赖关系和目标节点。资源需求及业务硬约束由可信输入 $I_c(t)$按 Job 标识关联，不允许生成模型改写，最终是否可执行由独立程序验证器判定。

1) 可行性验证。验证器重新获取最新集群状态，检查输出格式、Job 数量与标识以及目标节点真实性，并依次计算调度唯一性、累计资源、任务节点匹配和跨 Job 关系 4 项判定结果。在此基础上，验证器进一步检查已实例化的策略硬规则。仅当 $\Phi_c(Z, t)$=1 且全部策略硬规则通过时，输出 $O_c=\langle Z,\text{Scheduled}\rangle$并提交完整任务链；策略软偏好仅用于可行方案排序，不作为否决条件。

2) 迭代重调度机制。验证失败时，系统记录错误类型、关联 Job 与节点、违反的约束及资源差额，并同步最新集群状态后返回依赖感知的顺序规划与协同调度模块重新生成完整方案，若仍存在完整候选方案，则排除失败组合并重新执行逆向预留和正向调度；若必要 Job 无基础候选域、策略硬规则使有效候选域为空、依赖边无节点对可行域，或所有组合均无法保留关键 Job 的安全资源窗口，则输出 $O_c=\langle\Phi,\text{Pending}\rangle$，待资源状态变化后重新调度，重调度保持策略元语义和调度意图不变。

2　实验结果与分析

2.1　实验设置与评价指标

为验证策略感知 RAG 调度方法的有效性，笔者基于 Volcano 和 KWOK 构建边端异构集群实验环境，Volcano 执行传统资源策略与批任务调度，KWOK 模拟边缘节点、端侧节点及背景负载。节点与任务资源特征参考 Alibaba Cluster Trace GPUv2020 数据集，在保留 CPU、内存和 GPU 差异的基础上，补充节点站点归属、链路时延与带宽属性。

对比方法包括 default、binpack、spread、rsf−least 和 rsf−most。default 不强调资源聚合或分散；binpack 倾向复用已有负载节点；spread 倾向分散调度；rsf−least 优先选择调度后剩余资源较多的节点；rsf−most 优先选择调度后资源利用率较高的节点，RAG+rsf−most 表示本文中方法。实验在高负载环境下比较 6 种调度方法，用于评价目标策略的成功率、资源分配倾向和通信表现，同时构建 9 条任务链、共 33 个独立 Job，分别设置整体均衡高负载与 GPU 极高负载初始环境，统计 Pending Job 数、Job 成功率、任务链成功率，以及成功任务链的平均站点数、时延和带宽。RAG 输出在统计前接受格式、数量、节点真实性、角色、累计资源和依赖完整性等程序化校验，验证失败时依据结构化错误和最新状态重新生成，真实资源不足则保持 Pending。

2.2　面向单任务调度实验结果与分析

该实验在高负载条件下比较 5 种基础方法与 RAG+rsf−most 方法的单任务调度效果，共进行 20 轮单任务实验，高负载场景的初始 CPU、内存和 GPU 利用率分别为 78.7%、77.4%和 77.3%，用于检验 RAG 在紧张资源条件下能否维持目标策略的可行性和资源聚合方向。

比较 5 种基础方法的实验结果如表 1 所示。

表 1 高负载下不同调度方法的实验结果

方法	成功轮数	成功率/%	调度后			Gang 任务		
			CPU 利用率/%	内存利用率/%	GPU 利用率/%	平均站点数	平均时延/ms	平均带宽/Mbps
default	11	55.0	80.4	79.1	79.1	3.6	19.1	598.3
binpack	15	75.0	80.6	79.2	79.2	1.2	5.6	1 139.8
spread	10	50.0	80.3	79.0	79.1	4.4	22.5	507.2
rsf-least	10	50.0	80.3	79.0	79.1	4.4	25.6	455.2
rsf-most	17	85.0	80.8	79.4	79.2	1.1	8.0	882.3
RAG+ rsf-most	17	85.0	80.8	79.4	79.2	1.0	6.3	928.6

由表 1 可知：rsf-most 在高负载下成功率为 85%，在同类方法中达到最高水平，并呈现稳定的资源聚合倾向；因此，笔者选择 rsf-most 作为 RAG 学习的目标策略。若 RAG+rsf-most 在成功率和资源利用率上与目标策略一致，并保持相近的通信指标，则说明其能够理解目标策略。由表 1 可知：高负载下 rsf-most 与 RAG+rsf-most 均成功 17 轮，成功率为 85%，高于其余基础方法；两者成功轮数资源利用率一致的同时，RAG 将平均站点数由 1.1 降至 1.0，时延由 8.0 ms 降至 6.3 s，带宽由 882.3 bps 升至 928.6 bps，说明其保持了目标策略的成功边界，并进一步优化了可行节点组合。

2.3 面向任务链协同调度实验结果与分析

为区分单个 Job 内多副本调度与跨 Job 多任务调度，实验构建 9 条由 33 个独立 Job 组成的任务链，覆盖串行、并行、汇聚和串并行混合关系，如表 2 所示。

表 2 多任务实验中的任务链结构

任务链结构	关系特点	典型业务	资源特点
A→B→C	三阶段串行，前一任务完成后进入下一任务	视频流解码与预处理→运动目标分析→复杂目标判识	前轻后重，后续 GPU 需求递增
A→{B_1, B_2}	前置任务完成后 2 个后续任务并行	多维数据传输→关联研判+动态跟踪	前置 CPU 占用，双分支竞争 GPU
{A_1, A_2}→B	2 个前置任务完成后汇聚	多传感器预处理→融合决策	多源前置占用，汇聚端 GPU 需求高
A→B→C→D	四阶段长链串行	数据采集→数据清洗→逻辑推理→结果上报	长链多维资源占用，尾端资源紧张
A→{B_1, B_2, B_3}	前置任务触发 3 个并行分支	任务分发→烟火识别+人群密度估计+车辆异常检测	并发需求高，GPU 节点易被占用
{A_1, A_2, A_3}→B	3 个并行任务完成后汇聚	多路视频分析→综合告警	多路并行输入，汇聚任务负载高
A→B→{C_1, C_2}	前两阶段串行，末端并行扩展	数据采集→特征提取→分类+存储	前段消耗稳定，末端并发突增
A→{B_1, B_2}→C	中间阶段并行，最后汇聚	数据接入→边缘、云端联合推理→统一决策	中段双分支占用 GPU，尾端需保留 GPU
{A_1, A_2}→B→C	前端并行，后续串行	两路传感器检测→特征融合→控制执行	前端并行占用，后续链路需求高

这些任务链均由可独立提交的 Job 构成，而非单个 Gang 任务。串行任务链用于检验前序 Job 是否挤占后序资源，并行与汇聚结构用于检验多分支竞争及关键 Job 可调度性，因而能够区分 Job 成功与任务链整体成功。

2 个场景均包含 30 个节点。整体均衡高负载场景的 CPU、内存、GPU 利用率分别为 77.0%、77.0%、75.7%；GPU 极高负载场景分别为 76.9%、78.1%、94.3%。前者考察多类资源同时紧张，后者突出前序 Job 占用稀缺 GPU 窗口对后序 Job 的影响。

整体均衡高负载下的多任务调度结果如表 3 所示。

表 3 整体均衡高负载下的多任务调度结果

方法	任务链数	总 Job 数	总 Pending 数	任务链成功率/%	Job 成功率/%	成功任务链		
						平均站点数	平均时延/ms	平均带宽/Mbps
default	9	33	3	66.7	90.9	3.0	29.9	261.1
binpack	9	33	2	77.8	93.9	3.1	26.7	274.7
spread	9	33	10	0	69.7	–	–	–
rsf-most	9	33	3	77.8	90.9	2.9	27.7	245.8
rsf-least	9	33	10	0	69.7	–	–	–
RAG+rsf-most	9	33	0	100.0	100.0	2.7	22.3	259.0

由表 3 可知：传统方法的 Job 成功率最高为 93.9%，但任务链成功率最高仅为 77.8%；spread 与 rsf-least 虽完成 23 个 Job，9 条任务链均因关键 Job 缺失而失败，说明局部成功率不能替代整体可执行性。本文中方法将 Pending 数降为 0，Job 和任务链成功率均为 100%，平均站点数为 2.7 时延为 22.3 s。

其带宽并非所有方法中的最高，但在完整执行全部任务链的前提下取得了较好的站点聚合与时延结果。

GPU 极高负载下的多任务调度结果如表 4 所示。

表 4　GPU 极高负载下的多任务调度结果

方法	任务链数	总 Job 数	总 Pending 数	任务链成功率/%	Job 成功率/%	成功任务链		
						平均站点数	平均时延/ms	平均带宽/Mbps
default	9	33	11	0	66.7	–	–	–
binpack	9	33	11	0	66.7	–	–	–
spread	9	33	0	100	100.0	3.4	28.7	258.8
rsf-most	9	33	11	0	66.7	–	–	–
rsf-least	9	33	0	100	100.0	2.8	24.0	290.5
RAG+rsf-most	9	33	0	100	100.0	2.0	16.2	319.8

由表 4 可知：GPU 利用率升至 94.3 时，default、binpack 和 rsf-most 均出现 11 个 Pending Job，任务链成功率为 0；spread 与 rsf-least 则达到 100%，但两者在表 4 场景中的任务链成功率均为 0。这一反向结果表明，固定的逐 Job 资源倾向对瓶颈类型较敏感。本文中方法在 2 类负载下均实现 100%的 Job 和任务链成功率；在 GPU 极高负载下，其平均站点数、时延和带宽分别为 2.0、16.2 ms 和 319.8 bps，均优于同样完成全部任务链的 spread 与 rsf-least。目标策略 rsf-most 在该场景中未完成任何任务链，而 RAG+rsf-most 仍保持聚合方向并完成全部任务链，说明场景化调度意图与任务链资源预留能够避免机械复现逐 Job 评分结果。

2.4　典型多任务链联合调度案例

为解释 GPU 极高负载下 rsf-most 与 RAG+rsf-most 的差异，选取 21 和 121 2 种结构的 3 任务与 4 任务案例。21 表示 $\{A_1, A_2\}\rightarrow B$，121 表示 $A\rightarrow\{B_1, B_2\}\rightarrow$C，所有 Job 均要求部署在端侧节点。

候选节点中，exp2-terminal-dc1-001 和 exp2-terminal-dc2-002 各保留一个由 4 核 CPU、8GiB 内存和 1 块 GPU 组成的完整窗口；exp2-terminal-dc1-003 和 exp2-terminal-dc2-003 具有 CPU 资源但无 GPU，前序 CPU Job 是否占用 GPU 窗口将直接决定后序关键 Job 能否调度。

3 任务案例资源需求与调度结果如表 5 所示。

表 5　3 任务案例资源需求与调度结果

任务	业务阶段	依赖关系	CPU core	内存 GiB	GPU 块	调度结果	
						rsf-most	RAG+rsf-most
A_1 Job-001	图像增强与预处理	无	4	8	0	exp2-terminal-dc2-002	exp2-terminal-dc1-003
A_2 Job-002	候选目标提取	无	3	8	1	exp2-terminal-dc1-001	exp2-terminal-dc1-001
B Job-003	主判识推理	依赖 A_1、A_2	4	8	1	Pending	exp2-terminal-dc2-002

由表 5 可知：rsf-most 方法将无 GPU 需求的 A_1 调度至 exp2-terminal-dc2-002，并由 A_2 占用另一 GPU 窗口，导致 B 无法同时获得 4 核 CPU 和 1 块 GPU。本文中方法识别 B 为后序关键 Job，将 A_1 转移至无 GPU 节点，分别为 A_2 和 B 保留 2 个完整 GPU 窗口，完成三 Job 任务链调度。

4 任务案例资源需求与调度结果如表 6 所示。

由表 6 可知：rsf-most 方法调度结果中，前序 Job A 和 B_1 占用 2 个 GPU 窗口，末端汇聚 Job C 因无完整窗口而 Pending，本文中方法将 A 和 B_2 放入无 GPU 节点，将稀缺窗口保留给 B_1 和 C，从而完成 4 个 Job。2 个案例均表明，逆向资源预留能够将后序关键 Job 需求提前作用于前序调度，正向拓扑生成则在预留边界内维持目标策略倾向。

表 6　4 任务案例资源需求与调度结果

任务	业务阶段	依赖关系	CPU core	内存 GiB	GPU 块	调度结果	
						rsf-most	RAG+rsf-most
A Job-001	数据准备与预处理	无	4	8	0	exp2-terminal-dc2-002	exp2-terminal-dc1-003
B_1 Job-002	候选模型推理	依赖 A	3	8	1	exp2-terminal-dc1-001	exp2-terminal-dc1-001
B_2 Job-003	结果归一化	依赖 A	3	6	0	exp2-terminal-dc5-004	exp2-terminal-dc2-003
C Job-004	高精度判识	依赖 B_1、B_2	4	8	1	Pending	exp2-terminal-dc2-002

3　结论

笔者针对边端场景下任务链调度的核心挑战，即逐 Job 决策损害任务链整体可执行性与策略知识固化于实现代码，提出了一种基于策略感知检索增强生成的协同调度方法。

该方法的核心思想是将调度决策边界从单个 Pod 或 Job 扩展至完整任务链，使资源分配与节点选择在任务链全局视图中进行统筹，从根本上避免

局部最优决策破坏整体可执行性，并通过将分散于调度器源码、插件文档和策略规则中的调度逻辑转化为结构化、可检索的策略原语，使调度策略能够依据场景语义按需检索与组合，摆脱了传统方法中策略迁移需修改代码或重新训练模型的限制，还通过程序化验证与迭代修复机制，确保了调度结果满足资源、依赖和策略硬约束。实验结果表明：该方法能够在保持目标策略调度倾向的同时，有效减少高负载下的 Pending 任务，显著提升任务链整体成功率与通信效率，验证了任务链级统一决策、策略知识可检索组合与确定性验证相结合的技术路线的有效性。

目前，该方法依赖策略原语库的覆盖范围，当用户策略缺乏明确实现或说明时，组合泛化的准确性受限于语义相似度计算的精度；逆向资源预留以当前集群状态快照为依据，尚未考虑调度过程中其他并发任务同时提交带来的资源动态变化。未来工作将探索策略原语的自动发现与增量更新机制，降低人工构建成本；并将方法扩展至动态连续调度场景，在执行过程中根据实际资源消耗与节点状态变化动态调整预留策略，进一步提高调度结果的鲁棒性。

参考文献：

[1] SATYANARAYANAN M. The Emergence of Edge Computing[J]. Computer, 2017, 50(1): 30−39.

[2] AN X, FAN R, HU H, et al. Joint Task Offloading and Resource Allocation for IoT Edge Computing With Sequential Task Dependency[J]. IEEE Internet of Things Journal, 2022, 9(17): 16546−16561.

[3] BURNS B, GRANT B, OPPENHEIMER D, et al. Borg, Omega, and Kubernetes[J]. Communications of the ACM, 2016, 59(5): 50−57.

[4] FEITELSON D G, RUDOLPH L. Gang Scheduling Performance Benefits for Fine-Grain Synchronization[J]. Journal of Parallel and Distributed Computing, 1992, 16(4): 306−318.

[5] ZHAO H, SAKELLARIOU R. Scheduling Multiple DAGs onto Heterogeneous Systems[C]//20th International Parallel and Distributed Processing Symposium(IPDPS 2006). Rhodes Island, Greece: IEEE, 2006.

[6] TOPCUOGLU H, HARIRI S, WU M Y. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260−274.

[7] GRANDL R, KANDULA S, RAO S, et al. GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters[C]//12th USENIX Symposium on Operating Systems Design and Implementation(OSDI 16). Savannah, GA: USENIX Association, 2016: 81−97.

[8] 李金, 樊腾飞, 高红亮, 等. 边缘计算网络中多核任务卸载调度和资源适配研究[J]. 兵工自动化, 2025, 44(3): 29−34.

[9] MARCHESE A, TOMARCHIO O. Orchestrating DAG Workflows on the Cloud-to-Edge Continuum[C]//Proceedings of the 2nd Italian Conference on Big Data and Data Science (ITADATA 2023). Naples, Italy: CEUR-Workshop Proceedings, 2023: 3606.

[10] 刘文文, 熊伟, 韩驰. 基于改进 NSGA-II 的通信卫星资源动态调度方法[J]. 兵工自动化, 2022, 41(4): 69−75.

[11] GU J, CHOWDHURY M, SHIN K G, et al. Tiresias: A GPU Cluster Manager for Distributed Deep Learning[C]//16th USENIX Symposium on Networked Systems Design and Implementation(NSDI 19). Boston: USENIX Association, 2019: 485−500.

[12] MAO H, SCHWARZKOPF M, VENKATAKRISHNAN S B, et al. Learning Scheduling Algorithms for Data Processing Clusters[C]//Proceedings of the 2019 ACM Special Interest Group on Data Communication (SIGCOMM 2019). Beijing: ACM, 2019: 270−288.

[13] PEI H, GU Y, SUN Y, et al. LLM-Based Cost-Aware Task Scheduling for Cloud Computing Systems[J]. Journal of Cloud Computing, 2025, 14: 81.

[14] LEWIS P, PEREZ E, PIKTUS A, et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks[C]//Advances in Neural Information Processing Systems. Neural Information Processing Systems Foundation, 2020, 33: 9459−9474.

[15] JI Z, LEE N, FRIESKE R, et al. Survey of Hallucination in Natural Language Generation[J]. ACM Computing Surveys, 2023, 55(12): 1−38.