华南理工大学硕士学位论文

面向协同边缘计算的资源联合任务调度系统研究

梁华倩

指导教师：杨磊 教授

华南理工大学

2026 年 3 月 1 日

摘　要

协同边缘计算作为一种新兴范式，通过利用地理分布且异构的边缘节点资源，为自动驾驶、工业互联网等低延迟、高可靠应用提供了技术支撑。然而，边缘环境的异构性、计算与网络资源的紧密耦合性，以及分布式智能应用（如分布式训练）所呈现的复杂网络依赖拓扑，使得传统的任务调度机制在资源效率和系统性能上面临严峻挑战。当前研究主要存在两大局限：其一，主流基于有向无环图的任务依赖模型难以有效刻画可能具有环状等复杂通信特征的分布式智能任务；其二，现有调度方案普遍将计算资源与网络资源进行独立管理与调度，忽略了二者间的内在权衡关系，容易导致资源竞争与性能瓶颈。为解决上述问题，本文在协同边缘计算环境下设计了一种高效的计算与网络资源联合调度方案，主要工作如下：

本文设计并实现了协同边缘调度系统 CES（Collaborative Edge Scheduler），首次将任务到节点的映射与网络带宽分配进行联合管理。其次，构建了一个计算与网络资源的联合优化模型，该模型能够精确描述分布式训练等任务的通信拓扑与资源需求，并弹性适应动态环境。我们进一步提出了一种基于深度强化学习的智能调度算法，采用双分支演员-评论家架构，并融入启发式奖励函数，使系统能够在动态环境中自主学习全局最优的联合调度策略，从而实现系统负载均衡与用户带宽满足度等整体性能的优化。仿真实验和系统实验结果表明，该算法在整体性能上显著优于现有基准方法。

此外，为应对多分布式学习作业并发的场景，本文在 CES 基础上进一步提出了 CES-Multi 算法。该算法以提升资源利用均衡度、保障作业带宽需求满足度及降低作业平均等待时间为目标，在边缘节点多资源约束下，融合滚动窗口机制、资源感知贪心排序与动态饥饿保护策略。该算法通过滚动窗口动态选择候选作业，依据资源适配度、业务优先级和等待时间进行综合评分与排序，并调用既有单作业调度器执行资源分配；同时通过优先级动态提升与最小资源预留等机制，避免低优先级作业饥饿。实验表明，CES-Multi 在资源利用率、作业等待时间与带宽满足度方面的综合表现优于现有方法，有效增强了 CES 系统在多作业调度场景中的适用性与综合性能。

关键词：边缘计算；任务调度；多目标优化；深度强化学习；分布式训练

Abstract

Collaborative edge computing, as an emerging paradigm, provides technical support for low-latency and high-reliability applications such as autonomous driving and industrial internet by leveraging geographically distributed and heterogeneous edge node resources. However, the heterogeneity of edge environments, the tight coupling between computing and network resources, and the complex network-dependent topologies exhibited by distributed intelligent applications (e.g., distributed training) pose severe challenges to traditional task scheduling mechanisms in terms of resource efficiency and system performance. Current research mainly suffers from two limitations: First, mainstream task dependency models based on directed acyclic graphs (DAGs) struggle to effectively characterize distributed intelligent tasks that may have complex communication patterns such as cycles. Second, existing scheduling schemes generally manage and schedule computing resources and network resources independently, overlooking the inherent trade-off between them, which can easily lead to resource contention and performance bottlenecks. To address the above issues, this paper designs an efficient joint scheduling scheme for computing and network resources in a collaborative edge computing environment. The main contributions are as follows:

This paper designs and implements a collaborative edge scheduling system named CES (Collaborative Edge Scheduler), which, for the first time, jointly manages task-to-node mapping and network bandwidth allocation. Secondly, a joint optimization model for computing and network resources is constructed, which can accurately describe the communication topology and resource requirements of tasks such as distributed training and elastically adapt to dynamic environments. We further propose an intelligent scheduling algorithm based on deep reinforcement learning, which adopts a dual-branch actor-critic architecture and incorporates a heuristic reward function, enabling the system to autonomously learn the globally optimal joint scheduling policy in dynamic environments, thereby optimizing overall performance metrics such as system load balancing and user bandwidth satisfaction. Simulation and system experimental results demonstrate that the proposed algorithm significantly outperforms existing baseline methods in overall performance.

Furthermore, to address scenarios with concurrent multiple distributed learning jobs, this

paper proposes the CES-Multi algorithm based on CES. Aiming to improve resource utilization balance, guarantee job bandwidth satisfaction, and reduce average job waiting time, this algorithm integrates a rolling window mechanism, resource-aware greedy sorting, and dynamic starvation prevention strategies under multi-resource constraints of edge nodes. The algorithm dynamically selects candidate jobs through a rolling window, performs comprehensive scoring and sorting based on resource suitability, business priority, and waiting time, and invokes the existing single-job scheduler for resource allocation. Meanwhile, mechanisms such as dynamic priority boosting and minimum resource reservation are employed to avoid starvation of low-priority jobs. Experiments show that CES-Multi outperforms existing methods in terms of resource utilization, job waiting time, and bandwidth satisfaction, effectively enhancing the applicability and comprehensive performance of the CES system in multi-job scheduling scenarios.

Keywords: Edge Computing; Task Scheduling; Multi-objective Optimization; Deep Reinforcement Learning; Distributed Training

目　录

插图目录

第一章　绪论

本章为本课题提供了简要介绍。首先概述了研究背景和意义，然后总结了国内外关于协同边缘计算下分布式训练任务调度方法的研究现状，进而详细阐述了本文的主要研究工作及创新点。最后，对本文的组织结构进行了说明。

1.1　研究背景和意义

近年来，随着物联网与人工智能技术的深度融合与规模化应用，计算范式正经历一场深刻的变革，从以云计算为中心的集中式处理模式，逐步向“云-边-端”协同的分布式智能架构演进[1]。在这一过程中，边缘计算作为关键一环，通过将计算、存储与分析能力下沉至网络边缘侧，直接在数据源头或近端进行实时处理，有效缓解了云中心在带宽消耗、服务延迟和隐私安全方面的巨大压力[2]。然而，单一、孤立的边缘节点往往受限于其固有的资源瓶颈（如算力、存储）与物理覆盖范围，难以独立支撑日益复杂、跨地域协作且对实时性有严苛要求的智能应用。在此背景下，协同边缘计算应运而生，它超越了对单一边缘节点的优化，演进为一种旨在实现地理分布广泛、形态异构（包括边缘服务器、网关、车载单元及各类移动设备）的众多边缘节点之间，进行数据、计算、模型与网络资源深度共享与协同调度的新兴范式[3]。该架构通过统一的管控平面，对大规模、异构且地理分散的边缘基础设施进行联合管理与智能编排，从而将离散的边缘节点整合成一张高效的协同网络。这不仅显著提升了系统在服务可靠性、资源利用率和任务完成效率等方面的表现，支持业务的灵活部署与快速扩展，更推动了无处不在的智能服务走向现实。

在上述发展趋势中，分布式训练是实现边缘智能的关键任务，广泛出现在智能驾驶（多车感知协同）、工业互联网（跨厂区质量检测）和智慧城市（分布式视频分析）等前沿领域[4-5]。此类任务通常将训练数据分散在多个边缘节点上，通过协同执行本地计算并频繁交换模型参数或梯度信息，共同完成全局模型的更新[6]。在协同边缘计算环境下，分布式训练任务的执行涉及多个边缘节点之间的紧密协作，形成一个包含数据并行或模型并行的计算—通信流程。在协同边缘计算环境下，这样一个分布式训练任务的执行，本质上构建了一个逻辑上紧密耦合、物理上分散的计算-通信协同体。其性能表现呈现出鲜明的双重依赖性：既依赖于各参与节点的本地计算能力，更严重依赖于节点间通信链路的带宽、延迟和稳定性。任务的逻辑通信拓扑（如环形、星形等）如何高效、低冲突地映射到底层物理网络拓扑上，直接决定了同步过程的通信开销，进而成为影响整体

训练效率（如达到目标精度所需的时间）的决定性瓶颈。

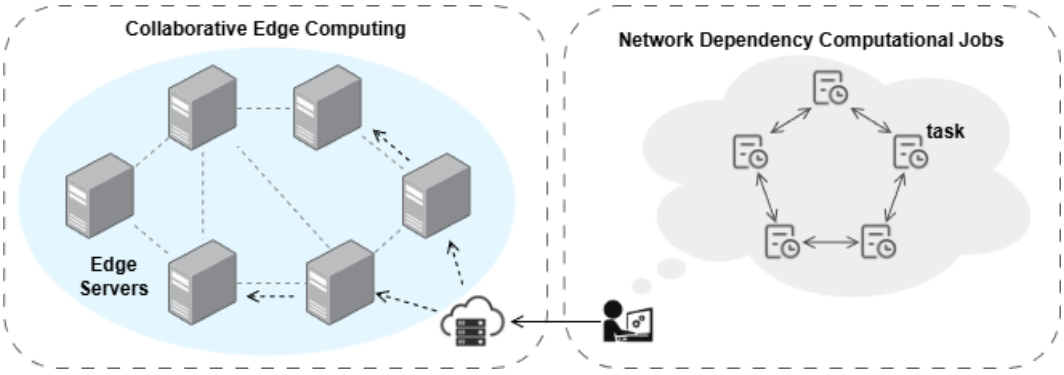

图 1-1 协同边缘任务调度场景图

这一将分布式训练等复杂任务调度到协同边缘网络中的场景（如图1-1），对资源管理和任务编排提出了一定的挑战。具体而言，该调度场景的核心特征体现在多维资源的耦合性与任务需求的复杂性上[7]。首先，边缘节点并非孤立的计算单元，其计算资源（如 CPU、内存）与网络资源（如带宽）紧密耦合，任务对一种资源的消耗往往会影响另一种资源的可用性。其次，任务内部（如分布式训练中的工作节点之间）存在着严格的数据依赖与同步点，形成了复杂的通信拓扑。任务的性能不仅由单个节点的计算能力决定，还会显著受到其内部通信链路所经历的物理网络性能的影响。任务的通信拓扑与底层物理网络拓扑的匹配程度，直接决定了通信开销的大小，从而成为系统整体性能的关键瓶颈。这些因素共同带来了如下核心难点：

（1）分布式训练等任务产生的密集、周期性通信流，在映射到共享的物理网络时，会竞争有限的带宽资源。由于任务逻辑通信拓扑可能非常复杂，不同任务间甚至同一任务内的数据流可能在不经意间共享某条关键物理链路，形成隐蔽的带宽竞争点[8]。这种竞争难以通过局部信息进行准确建模和预测，极易引发局部网络拥塞，造成部分任务同步延迟激增，从而破坏系统整体的负载均衡与性能稳定性。

（2）调度器在决策时面临固有的目标权衡。例如，为了最小化通信开销，理想策略是将频繁通信的任务子单元（如一个分布式训练的 Worker）尽可能集中部署在物理距离近、带宽充足的少数节点上。但这往往会导致这些节点形成计算热点，引发排队延迟，并违背了负载均衡的原则。反之，为了最大化计算资源的均衡利用，将任务分散部署到更广泛的节点上，又会显著增加节点间的通信距离与跳数，从而加大通信延迟和带宽压力。这种“计算聚集”与“通信分散”之间的矛盾，使得在协同边缘计算中实现计算与网络资源的联合均衡调度成为一个难以同时优化所有目标的复杂问题。

综上所述，在协同边缘计算环境下，如何设计一种能够同时考虑计算资源与网络资源，并在二者之间实现智能权衡的调度机制，具有广阔的应用前景及重要的研究意义。

1.2 研究现状与分析

1.2.1 协同边缘计算任务调度系统

随着边缘计算的兴起，各大云服务提供商推出了如 AWS IoT Greengrass[9]、Azure IoT Edge[10]和 KubeEdge[11]等平台，旨在将云计算功能扩展至资源受限的边缘设备，但这些平台仍遵循面向服务的集中式管理架构，对边缘自治和边到边协作支持有限，难以满足自动驾驶、工业物联网等新兴应用对超低时延、高连通性大规模部署和动态可靠服务的需求。在容器编排领域，Kubernetes 凭借其领导地位成为调度和管理应用程序的主流工具，但其设计初衷是针对云数据中心环境，假设资源丰富且网络稳定，并未专门考虑边缘原生应用的独特特性，如组件内部依赖性、资源异构性（数据、计算和网络紧密耦合）以及数据局部性，导致在边缘环境中资源利用不足和应用性能不佳。

尽管已有研究通过简化 Kubernetes（如 MicroK8s、K3s[12]）或扩展其能力到边缘（如 KubeEdge、OpenYurt[13]）来适应资源受限环境，但这些方案未改变核心调度逻辑，对应用性能感知不足；而其他工作[14-16]尝试引入网络感知调度以优化响应时间或任务部署，却忽略了计算资源异构性和数据局部性，且缺乏网络资源（如带宽）的协同编排。此外，在边缘任务调度中，许多研究[17-18]聚焦于最小化独立任务的平均完成时间或利用启发式、强化学习处理依赖任务，但往往忽视网络流调度，可能引发拥塞，而少数同时考虑任务分配和流程调度的研究[19-21]也未能优化应用吞吐量或缺乏实际系统实现。

随着边缘计算与分布式训练技术的融合发展，任务调度系统需同时应对计算资源异构性、网络状态动态性及任务依赖复杂性三大挑战，计算与网络资源的联合调度已成为提升分布式训练效率的核心突破口。现有基于 Kubernetes 的边缘调度方案虽在资源优化方面取得一定进展，但针对分布式训练任务的特性，仍存在显著局限：

FlexiTask[22]调度器构建了多维度资源调度框架，计算资源层面涵盖 CPU、内存、磁盘容量及 Pod 数量，通过短期与中长期资源利用率融合评估节点负载；网络资源层面将带宽纳入调度维度，突破了 Kubernetes 原生调度的局限。但针对分布式训练任务，网络资源管理中视带宽为静态资源，未考虑参数同步等动态网络依赖的时序性与突发性。Edge Service[23]框架通过双层架构扩展 Kubernetes，计算资源调度可感知节点硬件异构性（CPU/GPU 型号、算力差异），网络资源层面通过控制器动态维护节点拓扑与

延迟信息。但其计算与网络缺乏协同机制，优化目标仅聚焦节点级负载均衡，未关联算力分配与带宽占用，无法适配分布式训练对全局完成时间、端到端延迟的核心需求。FAOFE[24]基于 Argo 工作流引擎，在计算资源调度上，通过预调度阶段的 BFS 算法生成微服务执行序列，结合边缘节点 CPU、内存等计算资源消耗数据优化节点选择；在任务依赖建模上，采用有向无环图（DAG）梳理微服务逻辑关联，解决了 Kubernetes 调度与工作流任务顺序不一致的问题。但是 DAG 仅梳理逻辑依赖，未考虑计算节点与参数服务器间的动态网络交互，无法支撑并行训练的资源协同分配。ENTS[7]作为边缘原生调度系统，实现了计算与网络资源的初步联合编排。ENTS 通过 Profiler 解析任务计算需求，适配异构节点算力，此外，借助 Network Controller 管理带宽与路由，优化数据传输。但优化目标聚焦吞吐量，未考虑计算资源与网络资源的联动，导致资源配置失衡。

综上，现有调度方案虽在多资源均衡、QoS 感知、任务依赖建模等维度取得突破，但针对分布式训练任务的核心需求（计算-网络协同、动态网络依赖、全局性能优化）仍存在显著不足，缺乏能够同时适配训练任务特性、实现计算与网络资源精细化联合调度的机制。

1.2.2 协同边缘计算分布式训练任务调度算法

当前边缘计算任务调度算法研究普遍采用有向无环图（DAG）对任务进行建模，该模型在处理具有静态依赖关系的任务流时表现良好，例如 TF-DDRL[25]框架针对物联网应用的任务依赖建模，以及 MARS[26]框架对无人机辅助移动边缘计算系统中计算密集型任务的调度。然而，随着分布式训练等边缘智能应用的深入发展，任务间呈现出多种拓扑结构和动态交互的新特征，使得传统 DAG 模型在准确刻画此类复杂的数据流向与任务关联时面临挑战。现有研究[25]即便考虑了任务依赖，也往往局限于 DAG 的静态和无环假设，在适配实时变化的交互关系时效率受限，从而可能影响调度机制在复杂场景下的适应性与资源效率。更关键的是，当前调度研究多集中于独立或简单依赖任务（如 BD-TTS[27]中的物联网独立任务、A2C-DRL[28]中的随机到达任务），缺乏对多种网络拓扑下任务协同执行机制的专门设计，这进一步限制了其在分布式协同场景中的适用性。

另一方面，在虚拟网络嵌入（VNE）研究中，尽管考虑到了多种网络拓扑且智能化方法不断涌现，但其资源优化的核心焦点仍存在显著失衡。CE-VNE[29]、PPO-VNE[30]等算法虽引入了图卷积网络（GCN）自动提取拓扑特征，并设计了多目标奖励函数以同时优化资源收益与能耗，但其优化过程仍侧重于节点侧的计算与内存资源分配。网络带宽、I/O 等传输资源在状态设计中常仅作为特征之一，如 CE-VNE[29]包含带宽资源，

PPO-VNE[30]包含链路可用带宽，而未在奖励机制中被置于与计算资源同等的核心优化地位。RKD-VNE[31]虽创新性地引入了安全性（信任度）作为关键维度，尝试平衡资源利用与安全，体现了多维度优化的趋势，但其决策核心仍围绕节点 CPU、带宽等资源的约束满足展开。

这种“重计算、轻网络”的智能优化范式，并没有从根本上实现计算、带宽、安全等多维资源的深度协同。CE-VNE[29]等研究通用的两阶段式“节点-链路”映射动作，通常采用节点选择后接 BFS 或者 Dijkstra 路径搜索。这种做法虽提升了映射效率，但本质上仍将链路带宽优化视为一个被动的、满足约束的后续步骤，而非一个与节点计算资源同步、主动调度的核心决策变量。因此，在设备动态接入、多租户激烈竞争的真实边缘场景中，此类方法仍难以避免由带宽资源碎片化和竞争引发的传输瓶颈，导致整体系统性能受限。

综上所述，当前边缘计算任务调度研究面临双重核心挑战：一是传统 DAG 模型难以适配具有复杂网络拓扑的分布式训练任务；二是资源优化过程普遍忽视对网络带宽与拓扑的协同调度，导致系统在真实的多租户动态环境中面临严重的性能瓶颈。

1.3 本文研究工作及创新点

针对现有研究在建模分布式训练任务与实现计算-网络协同调度方面的不足，本文以分布式训练等具有复杂通信拓扑的网络依赖型任务为应用背景，旨在设计并开发一种能够联合调度计算与网络资源的智能任务调度系统。具体而言，本文的核心目标在于：突破传统有向无环图（DAG）模型在建模复杂交互拓扑方面的局限，构建能够准确刻画分布式训练中广泛存在的环状、星型等动态交互拓扑的任务表征与调度框架，从而更真实地反映参数同步、梯度聚合等过程带来的复杂通信行为；同时，将“任务应部署在哪些节点”（计算资源映射）与“应为任务分配多少带宽、选择哪条路径”（网络资源分配）这两个传统上分离的决策过程，深度融合为一个统一的联合优化问题，以系统化地避免因资源分别调度而引发的性能瓶颈。

为实现这一目标，本文构建了能够同时刻画计算约束与网络约束的联合优化模型，该模型不仅考虑了 CPU、内存等计算资源的可用性与异构性，还将网络拓扑结构、链路带宽及时延等关键通信因素纳入优化框架。在此基础上，引入深度强化学习方法，使系统能够在动态、异构的边缘环境中，通过与环境的持续交互自主学习全局近似最优的调度策略，实现从感知、决策到执行的闭环优化。最终，我们期望所提出的方案能够为协

同边缘计算构建一种资源感知能力强、整体性能优越且具备长期自适应演进能力的一体化任务—资源协同管理机制，从而有效弥补当前研究在应对复杂拓扑结构与多资源协同调度方面的缺陷。此外，为进一步提升系统在实际场景中的实用性，本文还将单作业调度问题系统性地推广至多任务队列调度问题，设计了相应的公平性与效率保障机制，以解决多作业并发执行时的资源竞争与整体调度优化问题。本文的主要贡献如下：

（1）提出了面向协同边缘计算的计算与网络资源联合调度系统 CES（Collaborative Edge Scheduler）：设计并实现了一个名为 CES 的协同边缘调度系统。该系统在分布式边缘基础设施中，将计算任务到异构节点的映射与网络带宽的动态分配进行统一编排与闭环管理，实现了对两类紧密耦合资源的协同调度。

（2）构建了联合优化模型并设计了基于深度强化学习的自适应智能调度算法 CES-PPO：建立了一个能够同时精准描述计算资源（CPU、内存）需求与网络资源（带宽、拓扑）需求的联合优化模型，并基于此提出了创新的深度强化学习算法。该算法采用双分支演员-评论家网络架构，并在奖励函数中融入启发式知识，使系统能够针对单次分布式训练任务，在动态环境中自主学习全局最优的联合调度策略，从而实现任务映射与带宽分配的协同优化。该模型与算法的有效性在仿真与真实系统实验的多场景测试中得到了验证。

（3）在 CES 系统基础上，面向多作业并发场景扩展并提出了 CES-Multi 算法：针对多用户、多分布式学习作业并发的实际场景，在单作业联合调度机制（CES）的基础上，进一步提出了 CES-Multi 多作业调度算法。该算法以提升资源利用均衡度、保障作业带宽需求满足度及降低作业平均等待时间为目标，融合了滚动窗口、资源感知贪心排序与动态饥饿保护策略，有效解决了多作业间资源竞争的公平性与效率问题，显著增强了 CES 系统在复杂生产环境中的综合性能与实用价值。

1.4 本文组织结构

本文共分为六个部分，整体组织结构如下：

第一章，绪论。本章首先阐述了协同边缘计算环境下分布式训练任务调度的研究背景与重要意义，分析了当前所面临的核心挑战。随后，从协同边缘计算任务调度系统与协同边缘计算分布式训练任务调度算法两个维度对相关领域的国内外研究现状进行了系统的梳理与深入的评析，指出现有工作的局限性。进而，在此基础上明确了本文的研究目标、研究思路与主要创新点。最后，对全文的组织结构进行了概要说明。

第二章，相关技术基础概述。本章旨在为后续研究提供必要的理论和技术铺垫。首先介绍了协同边缘计算的基本架构与核心特征，然后阐述了分布式训练的任务模型与通信模式，接着分析了任务调度问题的基本框架与关键指标，最后概述了深度强化学习的基本原理及其在资源调度领域的应用范式。这些内容共同构成了理解本文所提方法的基础。

第三章，协同边缘计算任务调度系统 CES 设计。本章首先明确了面向计算与网络资源联合调度的系统设计目标。随后，详细阐述了所提出的协同边缘调度系统（CES）的整体架构，说明了系统中各核心组件的功能与交互关系。进而，系统性地描述了从任务提交、资源感知、智能决策到最终部署的完整调度流程。本章内容为后续算法的实现与集成提供了系统级的框架支撑。

第四章，基于深度强化学习的联合任务调度算法。首先，对协同边缘环境下单次分布式训练任务的调度问题进行了形式化建模，将其定义为计算与网络资源联合优化的数学问题。随后，详细介绍了基于深度强化学习的智能调度算法设计，包括状态空间、动作空间与奖励函数的设计，以及双分支演员-评论家网络的结构与启发式思想引导训练机制。最后，通过仿真和系统实验与对比分析，验证了该算法在优化系统整体性能方面的有效性与优越性。

第五章，多任务队列调度算法。本章将研究场景从单任务扩展至多作业并发排队调度的更一般情况。首先对多作业调度问题进行了建模，定义了公平性、效率与服务质量等多重优化目标。随后，提出了 CES-Multi 算法，详细阐述了该算法滚动窗口机制、资源感知的作业排序策略以及动态饥饿保护机制的设计原理与执行流程。最后，通过实验评估了该算法在处理多作业并发时的综合性能。

最后，总结与展望。本章对全文的研究工作与创新成果进行了全面的总结，归纳了所提出的 CES 系统及其核心算法在解决协同边缘计算资源协同调度问题上的贡献。同时，客观分析了当前研究存在的局限性，并对未来可能的研究方向进行了展望。

第二章　相关技术基础概述

2.1　协同边缘计算

协同边缘计算是边缘计算技术面向分布式智能需求的进阶发展范式，其突破了传统云 - 边 - 端三层中心化架构的局限[3]。协同边缘计算通过大规模、地理分布式、异构边缘节点的自主协作与资源联动，实现计算、存储、网络与数据资源的跨节点协同调度，为自动驾驶、工业物联网、元宇宙等新兴复杂应用提供超低延迟、高可靠、泛连接的智能化服务支撑[5]。该技术将智能能力下沉至网络边缘并实现节点间的能力聚合，核心愿景是构建边缘分布式智能生态，使边缘节点在脱离云端主导的情况下，仍能完成复杂计算任务的协同处理与服务的持续供给，是实现物联网泛在智能的核心技术方向之一。

协同边缘计算的提出与发展，本质是传统边缘计算技术的局限性与新兴应用的严苛需求之间矛盾的必然结果，其核心驱动因素可分为应用需求与技术局限两方面[32]。新兴应用的技术需求方面，自动驾驶、元宇宙、工业物联网等应用对边缘计算提出了三大核心需求，一是超低延迟，要求毫秒级完成目标检测、轨迹规划、3D 点云重建等计算密集型任务；二是超连接与大规模部署，需支撑海量异构终端（VR 设备、工业传感器、车载终端等）的跨地理区域互联与数据交互；三是动态可靠服务，要求为高移动性终端提供跨区域的无缝服务供给，应对网络动态变化与设备移动带来的服务中断问题。其次是传统边缘计算的技术局限，传统边缘计算以云 - 边协作为核心，采用云端主导的中心化资源管理模式，存在显著短板。其一，云 - 边数据传输带来不可预测的延迟，且数据上云存在隐私泄露风险；其二，忽视边缘节点间的协作能力，单一边缘节点资源受限导致复杂任务处理能力不足；其三，中心化架构的可扩展性差，难以支撑海量终端的大规模部署与动态资源调度；其四，对云端的强依赖导致网络不稳定时服务可靠性大幅下降，无法满足高移动性应用的服务需求[33]。

协同边缘计算通过三大核心技术特征实现分布式智能，区别于传统边缘计算的技术体系，也是其支撑新兴应用需求的关键属性，三者相互协同构成协同边缘计算的技术基础：

(1) 边缘自治：指边缘节点具备离线或本地化独立运行能力，无需与云端进行通信即可完成服务的部署、执行与调度，能够在云 - 边网络中断、延迟过高的场景下保障服务的持续供给，是实现服务可靠性的核心基础[34]。

(2) 边缘-边缘协作：指边缘节点可与周边异构节点形成动态协作集群，实现计算资

源、数据资源的跨节点共享，完成 AI 算法推理、复杂任务分布式处理等计算密集型工作；同时支持跨平台、跨架构的无缝计算与动态接入，实现边缘能力的聚合与互补[35]。

(3) 资源弹性：指系统针对大规模、分布式、异构的边缘环境，具备资源的自主调度与动态适配能力，可根据任务负载与节点状态完成计算、存储、网络资源的自动化分配与回收，保障边缘节点资源的高效利用与任务的最优执行[36]。

目前协同边缘计算的代表性体系架构为面向服务的边缘即服务（Edge-as-a-Service, EaaS）框架，该框架以服务化架构为核心，实现了边缘资源的跨节点协同管理与边缘应用的端到端开发部署，整体分为基础设施即服务（IaaS）、平台即服务（PaaS）、软件即服务（SaaS）三层，各层自上而下实现能力支撑与服务调用，构成完整的协同边缘计算技术体系，具体设计如下：

(1) 基础设施即服务（IaaS）[37]：为协同边缘计算提供底层资源支撑，核心是解决大规模异构边缘资源的协同管理与调度问题。该层设计分布式边缘操作系统（EdgeOS）作为核心管理载体，采用容器轻量级虚拟化技术抽象异构的计算、存储、网络资源，实现服务在地理分布式边缘节点间的无缝迁移；同时设计网络调度器、计算调度器、存储调度器三大协同调度模块，联合考虑数据局部性、资源异构性与网络动态性，完成跨节点耦合资源的最优分配，满足边缘原生应用的低延迟、高吞吐量需求。

(2) 平台即服务（PaaS）[38]：为边缘应用开发提供通用化平台能力支撑，是实现边缘分布式智能的核心层，主要提供边缘学习服务与边缘区块链服务（EBaaS）两大核心能力。其中，边缘学习服务支撑 AI 模型的全生命周期管理，涵盖数据预处理、模型开发、分布式训练（联邦学习、流言学习、E-tree 学习等范式）与资源感知推理优化（模型压缩、分区、早退出）；边缘区块链服务则将区块链组件部署于边缘节点，实现低延迟、高安全的分布式数据存证与共享，满足边缘场景下的隐私保护与数据一致性需求。两层服务均提供标准化 API，实现与底层 IaaS 层的资源调用与上层 SaaS 层的应用开发支撑。

(3) 软件即服务（SaaS）[39]：面向具体业务场景提供边缘原生应用服务，要求应用采用模块化、无状态的设计范式，将复杂应用拆分为多个相互依赖的子模块并分布式部署于多个边缘节点，适配单一边缘节点的资源约束；同时依托底层 PaaS 与 IaaS 层提供的统一编程抽象与资源调度能力，屏蔽边缘设备的异构性与网络的动态性，实现应用的高效开发与跨节点协同运行，典型的边缘原生 SaaS 应用包括实时视频监控、智能建筑监测、自动驾驶决策等。

协同边缘计算的落地实现依赖于多领域技术的融合支撑，结合 EaaS 框架的设计需

求，其核心支撑技术围绕边缘资源管理、智能计算、数据交互、应用开发与安全保障展开，具体包括六大类[3]：

(1) 轻量级跨平台虚拟化技术：采用容器替代传统虚拟机，实现边缘资源的轻量级抽象，同时需适配资源受限的嵌入式边缘设备，支持 x86/ARM 等多架构、GPOS/RTOS 等多系统与 CPU/GPU/TPU 等多计算单元的跨平台部署，兼顾多租户支持与安全隔离。

(2) 边缘原生资源调度技术[40]：针对边缘场景的耦合资源、动态网络、大规模分布式特征，设计协同调度算法，实现计算、存储、网络资源的联合调度，同时考虑设备与网络的可靠性约束，采用分布式调度模式提升大规模场景下的调度效率与可扩展性。

(3) 分布式数据共享技术：设计适配边缘场景的轻量级共识机制，解决大规模、异构、网络不稳定条件下的状态数据一致性问题，支持多应用的差异化一致性需求，实现边缘节点间的数据安全共享与同步。

(4) 资源感知边缘 AI 技术：针对边缘场景的 NonIID 数据、无标注数据、流式数据特征，设计自适应的分布式模型训练方法；同时通过模型压缩、分区、早退出等手段实现资源感知的模型推理，适配边缘节点的资源约束。

(5) 轻量化安全与隐私保护技术：开发适配边缘资源受限特征的轻量级加密、访问控制机制，防范无线通信窃听、跨节点攻击扩散等安全风险，平衡多利益相关方的数共享与隐私保护需求。

(6) 统一编程抽象技术：构建轻量级、通用化、模块化的中间件，为应用开发者提供标准化的编程接口，屏蔽边缘设备异构、网络动态、资源不确定等底层复杂性，使开发者聚焦于应用逻辑设计，实现边缘应用的端到端开发与部署。

协同边缘计算依托低延迟、高可靠、分布式协作的技术优势，在实时视频监控、智能建筑、自动驾驶这类对计算延迟、服务可靠性与资源协同性要求严苛的新兴领域实现了典型落地并发挥显著应用价值：在实时视频监控场景，通过边-边协作满足校园、工厂、园区等中距离场景的低延迟智能分析需求，还能灵活调用云端算力支撑城市级大规模监控，兼顾效率与存储；在智能建筑场景，借助边缘节点协同通信实现各监测子系统联动，高效运行分布式数据处理算法，适配场景的低延迟与数据敏感需求；在自动驾驶场景，通过协同管理车载终端、路侧单元、基站等各类边缘资源，完成核心任务的跨节点高效卸载，解决单节点算力不足、基站网络与负载问题，保障自动驾驶决策的毫秒级执行要求。

2.2 任务调度

在边缘计算环境中，任务调度是实现系统高效运行的核心问题。该问题本质是在由云、边缘服务器和终端设备构成的异构、分布式系统中，动态地为一系列具有不同属性与需求的计算任务分配合适的计算、通信和存储资源，并确定任务的执行位置与顺序。由于该问题通常属于 NP-hard 复杂问题[2,5]，其求解需综合考虑多类约束与多个相互冲突的目标，因此设计和评估调度算法具有重要的理论和实际意义。

边缘计算中的任务调度可形式化描述为一个在多重约束下寻求最优决策的数学优化问题，其输入、决策、目标与约束如下：

首先，问题的输入包含三个关键部分：

• 待处理的任务集合：包含多个相互独立或具有依赖关系的计算任务。每个任务都有一系列特征属性，例如完成任务所需的计算强度（如 CPU 核数）、需要传输的数据量大小、必须完成的截止时间，以及可能存在的与其他任务的先后执行顺序（即工作流依赖关系）和多任务间需要建立的网络连接通信拓扑关系。

• 可用的资源集合：涵盖系统中所有可供调度的实体，包括远端云数据中心、网络边缘的服务器、以及终端设备本身。资源类型不仅指计算单元（如 CPU、GPU），也包括通信资源（如网络带宽）和存储资源。

• 动态的系统状态：指随时间变化的系统环境信息，例如各节点之间的网络延迟与可用带宽、各个计算节点的当前负载情况、终端设备的剩余电量、以及设备的移动轨迹等。

其次，调度器需要做出的一系列决策，即输出方案，主要包括：

• 为每个任务指定在何处执行（例如，本地设备、某个特定的边缘服务器或云端），这通常是一个二元选择变量。

• 决定为每个任务分配多少具体的资源量，例如分配多少计算能力（CPU 频率）、占用多少通信带宽。

• 对于可以拆分的大型任务，确定有多大比例的部分被卸载到远程执行，其余部分在本地执行。

• 确定所有任务执行的先后顺序或开始时间。

目标函数通常为多指标的综合优化，常见方向包括：

• 最小化任务完成延迟（含传输、排队、计算与回传）；

- 最小化系统或设备能耗；
- 最大化资源利用率或系统吞吐量；
- 在满足服务质量的前提下降低经济成本。

约束条件则包括：

- 任务截止时间约束；
- 节点处理能力上限（CPU、内存）；
- 网络带宽限制；
- 设备能量预算；
- 任务间的依赖关系约束。

因此，调度问题可表述为：在满足上述约束的前提下，通过优化决策变量，使目标函数达到最优或近似最优。

边缘计算任务调度的核心目标是一个多维度且通常存在内在冲突的权衡体系，构成了算法设计与性能评估的根本依据[4]。这些目标主要涵盖四个层面：在性能层面，首要追求最小化任务延迟（涵盖传输、排队、计算与回传的全过程时延），以满足自动驾驶、增强现实等实时应用的严苛要求，并力求最大化系统吞吐量以提升整体处理能力；在效率层面，核心关注最小化能耗以延长终端设备续航并降低运营成本，同时最大化资源利用率以实现基础设施的更经济化使用。此外，还需考量经济与服务品质，包括最小化经济成本（如资源租赁与数据传输费用）、保障服务质量与用户体验，以及通过合理机制保证多用户或多任务间的资源分配公平性，防止资源饥渴。最后，在系统稳健性层面，实现负载均衡是关键，旨在通过均匀分发任务来避免局部过载，从而提升系统的可靠性与可扩展性。实际调度算法必须综合协调这些相互竞争的目标，进而求解一个复杂的高维多目标优化问题。

针对边缘计算中复杂的任务调度问题，研究者提出了多种优化方法，各类方法在适用场景与性能上各有特点：

(1) 传统精确优化方法[41-42]：为调度问题提供了形式化的数学基础。该方法通常将问题建模为混合整数线性或非线性规划模型，并借助 CPLEX、Gurobi 等专业求解器进行计算。其核心优势在于能够获得理论上的最优解，为性能评估提供黄金标准。然而，其计算复杂度随问题规模呈指数级增长，难以适用于大规模、高动态的真实边缘场景。因此，这类方法主要角色是进行离线理论分析、为其他启发式算法提供性能上界参照，或用于小规模静态场景的精确求解。

(2) 启发式与元启发式算法[43-46]：为应对大规模实际问题提供了实用且高效的近似求解途径。启发式算法依赖直观规则（如最早截止时间优先）进行快速决策；而元启发式算法，如模拟生物进化过程的遗传算法、模仿鸟群协作的粒子群优化以及受热力学启发的模拟退火算法，则通过结构化地探索解空间来逼近全局优解。这类方法在可接受时间内能为大规模、非凸问题提供高质量解，且实现相对简单。但其性能往往依赖于经验性参数调优，无法保证最优性，并且在超动态环境中可能因收敛速度问题而难以适配。它们广泛应用于任务集相对固定、需在异构边缘节点间进行分配的静态或半静态调度场景。

(3) 博弈论方法[47-48]：从经济学与交互决策视角为调度问题提供了新颖的建模框架。它将资源提供方（边缘/云）与任务所有者（用户）建模为理性且自私的博弈参与者，通过分析纳什均衡等均衡概念，来研究分布式环境下的资源竞争、定价与分配策略。这种方法天然适合刻画边缘计算中多主体、分散决策的特性，能够有效模拟用户间的竞争与合作关系。其主要挑战在于均衡解的存在性证明与求解复杂性，且通常依赖于信息完全或部分完全的假设。该方法在多用户竞争边缘资源的卸载决策、以及服务提供商之间的资源定价策略设计等场景中具有独特价值。

(4) 基于学习的方法（如深度强化学习）[49-50]：特别是机器学习与深度学习，代表了应对高度动态与复杂环境的前沿方向。其中，深度强化学习通过让智能体（即调度器）以试错方式与环境交互，并根据延迟、能耗等指标构成的奖励信号不断优化策略，从而学会自适应调度。深度学习则可用于精准预测任务需求或网络状态，甚至端到端地直接输出调度决策。这类数据驱动的方法具有强大的环境感知与在线适应能力，擅长处理高维状态空间问题。然而，它们通常需要大量的训练数据与计算资源，其决策过程如同“黑箱”般可解释性差，在安全关键型应用中需格外谨慎。该类方法在车辆边缘计算、无人机辅助移动边缘计算等网络拓扑与任务负载快速变化的场景中展现出巨大潜力。

(5) 基于李雅普诺夫优化的在线算法[51-52]：提供了一种具有严格理论保证的实时决策框架。它将具有长期平均约束的优化问题（如平均功耗限制）巧妙地分解为一系列在每个时隙内可解的即时确定性子问题，并通过构造虚拟队列将约束违反转化为队列稳定性问题。这种方法的突出优点在于其在线特性——无需未来信息即可做出实时决策，并能从理论上保证系统的长期队列稳定与性能边界。其局限性在于决策本质上是贪心的，可能牺牲短期最优性，且对如何将原问题转化为李雅普诺夫框架有较高的数学技巧要求。它尤其适用于能量收集型边缘节点等资源受长期平均约束的动态调度场景。

(6) 考虑公平与负载均衡的专门方法[50,53]：是一种重要的设计原则与优化视角。它强调将公平性指标（如最大最小公平、比例公平）或负载均衡指标（如节点间负载方差）明确地纳入目标函数或作为约束条件。这一理念可以与上述多种方法深度结合：例如，在博弈论中采用纳什议价解来建模公平分配；在优化模型中引入公平性权重；或在强化学习的奖励函数中设计惩罚项以促进负载均衡。这类方法对于构建健康、可持续的多租户边缘计算平台至关重要，能够确保所有用户或区域都能获得基本可用的服务，防止资源分配失衡，从而提升系统的整体稳健性与社会福祉。

综上，边缘计算任务调度是一个多目标、多约束、动态 NP-hard 问题，其求解方法呈现多元化发展趋势，未来研究趋向于融合多种方法，以适应边缘环境的高动态、异构和安全可靠需求。

2.3 分布式训练

随着互联网技术的迅速发展，数据量急剧增加，单机处理大规模数据集变得越来越困难。这促使研究人员探索新的方法来提高数据处理能力和学习效率，分布式机器学习因此被提出。它通过在多个计算节点上并行执行算法，以加速机器学习模型的训练过程，从而实现处理大型模型、缩短训练时间和提升模型性能的目标。

在边缘计算环境中，分布式机器学习面临独特的挑战：边缘设备通常具有资源受限、异构性强、网络条件不稳定等特点，无法存储大量数据且难以实现高效的中间计算结果交换。因此，本文的研究重点是基于数据并行模式，在协同边缘计算背景下开展相关工作。

分布式机器学习按照并行模型可分为三种类型：模型并行、数据并行和混合并行[54]。在模型并行中，模型的不同部分被划分到不同的计算节点进行处理，适用于模型过大而无法在单节点内存中加载的场景，但对节点间通信效率要求较高。数据并行则是将数据集分割为多个子集，分配到不同计算节点，每个节点持有完整的模型副本进行本地训练，之后通过聚合本地更新（如参数平均或梯度更新）来同步全局模型。该模式适合数据量大而模型相对较小的任务，也更适应边缘设备存储与计算受限的特点。混合并行结合了数据并行与模型并行的优势，可同时应对大规模数据与大型模型，但其设计与实现更为复杂，需综合考虑模型划分与数据划分策略。

分布式机器学习的通信步调主要分为同步通信、异步通信和半同步通信[55]。同步通信要求所有节点完成当前轮次后再进入下一轮计算，保证模型一致性，但可能受限于

最慢节点。异步通信允许节点独立更新，提高了资源利用率，但可能因信息滞后影响收敛效果。半同步通信则尝试在两者间取得平衡，允许一定程度延迟，以减少等待时间并控制不一致性。在边缘协同环境中，网络异构与动态变化使得通信步调的选择与优化尤为关键，需要结合任务特性与资源状态进行动态调整。

当前，协同边缘计算中的分布式智能任务（如分布式训练）呈现出复杂的网络依赖拓扑，传统基于有向无环图的任务模型难以准确刻画其通信特征。同时，现有调度机制往往将计算与网络资源独立管理，忽视了二者在边缘环境中的紧密耦合与权衡关系，易引发资源竞争与性能瓶颈。因此，面向边缘环境的分布式机器学习需构建能够联合调度计算与网络资源的优化框架，以适应动态、异构的资源条件并保障任务整体效率与系统均衡性。

2.4 深度强化学习

2.4.1 深度强化学习概述

强化学习作为机器学习的一个重要分支，其核心目标在于通过与环境的交互，使智能体学会如何做出序列决策以最大化长期累积奖励。与监督学习和无监督学习不同，强化学习强调在动态和不确定的环境中通过试错进行学习，尤其适用于决策过程复杂、环境反馈延迟或稀疏的场景。

在强化学习框架中，智能体作为学习与决策的主体，通过感知环境状态并执行动作来与环境进行交互。环境则根据智能体的动作反馈新的状态和即时奖励。状态通常表示为观测的特征向量或张量，而动作可以是离散的或连续的，取决于具体任务。奖励作为环境对智能体行为的评价信号，引导智能体逐步优化其策略。策略作为状态到动作的映射，可以是确定性的，也可以是随机的，它决定了智能体在特定状态下如何行动。此外，价值函数用于评估状态或动作的长期价值，为智能体的决策提供评估依据。其中，状态价值函数 $V^{\pi}(s)$ 表示在状态 s 下遵循策略 π 的期望累积回报，而动作价值函数 $Q^{\pi}(s,a)$ 则表示在状态 s 下执行动作 a 后再遵循策略 π 的期望累积回报。二者关系可通过贝尔曼方程表达：

$$V^{\pi}(s)=\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim P(\cdot|s,a)}\left[R(s,a)+\gamma V^{\pi}(s')\right] \tag{2-1}$$

以及

$$Q^{\pi}(s,a)=\mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R(s,a)+\gamma\mathbb{E}_{a'\sim\pi(\cdot|s')}\left[Q^{\pi}(s',a')\right]\right] \tag{2-2}$$

在部分方法中，还会构建环境模型来预测状态转移和奖励，从而辅助规划和决策。

强化学习的最终目标是学习一个最优策略 π^*，使得从初始状态开始的累积折扣奖励最大化，即：

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right], \tag{2-3}$$

其中 $\tau = (s_0, a_0, s_1, a_1, \dots)$ 表示一条轨迹，$p_\pi(\tau)$ 表示在策略 π 下轨迹的分布，$\gamma \in [0, 1]$ 是折扣因子，用于平衡即时奖励与未来奖励的重要性。

从学习方式上看，强化学习可分为在线学习与离线学习。在线学习中，智能体直接通过与环境的交互更新当前策略；而离线学习则允许智能体从历史经验回放中学习，能够利用过往策略收集的数据，提高样本效率。根据学习目标的不同，强化学习方法主要分为三类：基于价值的方法通过逼近最优价值函数来间接得到策略，其核心是学习最优动作价值函数 $Q^*(s, a)$，满足贝尔曼最优方程：

$$Q^*(s, a) = \mathbb{E}_{s' \sim P} \left[R(s, a) + \gamma \max_{a'} Q^*(s', a') \right] \tag{2-4}$$

经典的 Q 学习及其深度扩展 DQN 即属于此类。基于策略的方法则直接优化策略函数 $\pi_\theta(a|s)$，通过参数 θ 进行参数化，并沿期望回报的梯度方向进行更新：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \right], \tag{2-5}$$

其中 $J(\theta)$ 是策略性能的度量（如期望回报），G_t 是从时刻 t 开始的累积回报。REINFORCE 及其各种改进算法属于此类。而演员-评论员方法巧妙地将两者结合，其中“演员”负责根据策略 $\pi_\theta(a|s)$ 选择动作，“评论员”则利用价值函数 $V_\phi(s)$ 或 $Q_\phi(s, a)$ 评估动作的价值，代表性算法包括 A3C、DDPG 等。此外，根据是否依赖环境模型，还可以分为基于模型的方法和无模型方法。

强化学习问题通常被形式化为马尔可夫决策过程。MDP 假设系统具有马尔可夫性，即未来状态仅依赖于当前状态和动作，而与历史状态无关，由五元组 $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ 定义。

深度强化学习将深度学习的强大表征能力与强化学习的序列决策能力相结合，通过深度神经网络来近似价值函数、策略或环境模型。这一融合使得智能体能够处理高维、复杂的原始输入数据，从而在诸多领域取得了里程碑式的成就。然而，DRL 也面临着若干挑战，例如样本效率低下、探索与利用之间的平衡难题、训练过程的不稳定性，以

及稀疏奖励环境下的学习困难等。

2.4.2 近端策略优化算法原理

近端策略优化算法 Proximal Policy Optimization(PPO)[56]是一种基于策略梯度的深度强化学习方法，其核心设计思想在于通过引入一种特殊的优化目标来约束策略更新的幅度，从而在追求性能提升的同时，确保训练过程的稳定性。

PPO 建立在策略梯度方法的基础之上。PPO 算法通过优化一个精心设计的替代目标函数来实现策略的渐进式改进。该目标函数的核心是“裁剪”机制。具体而言，算法首先计算新旧策略的概率比值：

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, \tag{2-6}$$

其中 $\pi_{\theta_{\text{old}}}$ 是更新前的旧策略参数。PPO 的目标是最大化一个裁剪后的优势函数期望值。其裁剪目标函数定义为：

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{2-7}$$

其中 $\hat{A}_t$ 是时刻 t 的优势函数估计值，衡量了特定动作相对于平均水平的优劣；ϵ 是一个小超参数（通常为 0.1 或 0.2），用于定义裁剪区间。$\text{clip}(x, l, u)$ 操作将 x 限制在 $[l, u]$ 之间。该公式通过取最小值，确保了目标函数是原始比值项与裁剪后比值项的下界，从而避免了因策略更新过大（$r_t(\theta)$ 偏离 1 过远）而导致的性能崩溃。

优势函数 $\hat{A}_t$ 通常通过广义优势估计进行计算：

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \tag{2-8}$$

其中 $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ 是时序差分误差，λ 是权衡偏差与方差的参数。

为了提升性能与稳定性，PPO 的完整目标函数通常结合了价值函数损失和策略熵奖励：

$$L^{\text{PPO}}(\theta, \phi) = \hat{\mathbb{E}}_t \left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\phi) + c_2 S[\pi_\theta](s_t) \right], \tag{2-9}$$

其中 $L_t^{\text{VF}}(\phi) = (V_\phi(s_t) - V_t^{\text{targ}})^2$ 是价值函数 V_ϕ 的均方误差损失，$S[\pi_\theta](s_t)$ 是策略在状态 s_t 下的熵，用于鼓励探索，c_1 和 c_2 是系数。

相较于早期的策略优化方法，如信赖域策略优化，PPO 在保持相近性能的同时，实现上更为简洁高效。其训练流程通常包括使用当前策略交互采样、计算优势估计、以及

进行多轮小批量梯度更新等步骤。由于其在连续和离散动作空间中的良好表现、优异的稳定性以及较高的样本效率，PPO 已成为最受欢迎的深度强化学习算法之一，被广泛应用于机器人控制、游戏智能、自动驾驶等诸多领域[57]。

2.5 本章小结

本章系统阐述了协同边缘计算、任务调度、分布式训练与深度强化学习四个基础技术领域，为后续研究提供理论支撑。首先，从协同边缘计算的演进、核心特征、体系架构及支撑技术入手，阐明边缘分布式智能平台的关键作用。其次，分析任务调度问题的形式化描述、优化目标及典型求解方法。随后，聚焦分布式训练在边缘场景中的应用挑战。最后，介绍深度强化学习基本原理，并重点剖析近端策略优化算法，为自适应调度算法设计提供方法论基础。通过对上述技术的系统梳理与分析，本章为后续章节中边缘任务调度系统与算法设计与评估奠定了坚实的理论基础，并明确了研究的核心问题与技术路径。

第三章　协同边缘计算任务调度系统 CES 设计

本节将详细阐述 CES 协同边缘计算调度系统的设计思路、核心架构与关键流程。首先，本节将明确系统旨在解决的核心问题与设计目标；其次，将深入剖析基于主从架构的系统组件设计与交互机制；最后，将系统地描述一个作业从提交到执行完毕的任务调度过程。

3.1　系统设计目标

CES 协同边缘计算调度系统是一种面向智能边缘环境的任务调度系统，旨在解决弹性环境中具有通信依赖关系的分布式任务调度问题。此处定义的“弹性环境”包含双重动态性：一是计算与网络物理资源可随节点加入或离开而动态变化；二是待处理的任务规模与资源需求可随时间动态波动。在此复杂环境下，传统的、孤立考虑计算或网络资源的调度策略难以实现整体效能最优。

因此，本系统的核心设计目标是实现对计算资源与网络资源的统一感知与联合调度。系统需构建一个全局的、融合的资源视图，统一纳管任务数据布局、节点计算能力与网络链路状态，并以此为基础进行协同决策。具体而言，CES 系统旨在达成以下三个关键设计指标：

- **确保系统在真实边缘环境下的调度可信度**：系统需直接部署并运行于真实的边缘计算环境中，该环境天然具备节点的异构性（如计算能力与网络接口的差异）、资源的动态弹性以及网络拓扑的复杂性。调度系统必须能够在此类真实、动态的条件下，持续感知环境变化，并生成与执行有效的调度决策。其目标是保证系统在实际部署中的调度行为与结果真实、可靠，从而验证其工程实用性与有效性。
- **提升跨边缘节点的资源利用均衡度**：避免部分节点过载而其他节点空闲的资源碎片化现象。调度算法应综合考虑 CPU、内存及网络等多维资源，力求在系统全局范围内实现负载均衡，从而提高基础设施的整体资源利用率与投资回报率。
- **保障作业间的性能隔离**：在共享的、多租户边缘环境中，确保不同作业的运行环境相互隔离，避免资源争用导致的性能干扰。系统需通过容器化技术与细粒度的资源配额管理，强制实施 CPU、内存及网络带宽的资源隔离，为共存的作业提供独立、可控的执行环境，保障其性能的独立性与可预期性。

为实现上述目标，CES 系统被设计为一个覆盖“资源感知-智能决策-精准执行”的一体化任务调度系统。该系统不仅强调调度算法在理论上的优化能力，更注重其在实际

异构、动态边缘环境中的工程可实现性与鲁棒性，从而最终为用户提供一个稳定、高效且资源感知的边缘计算服务基底。

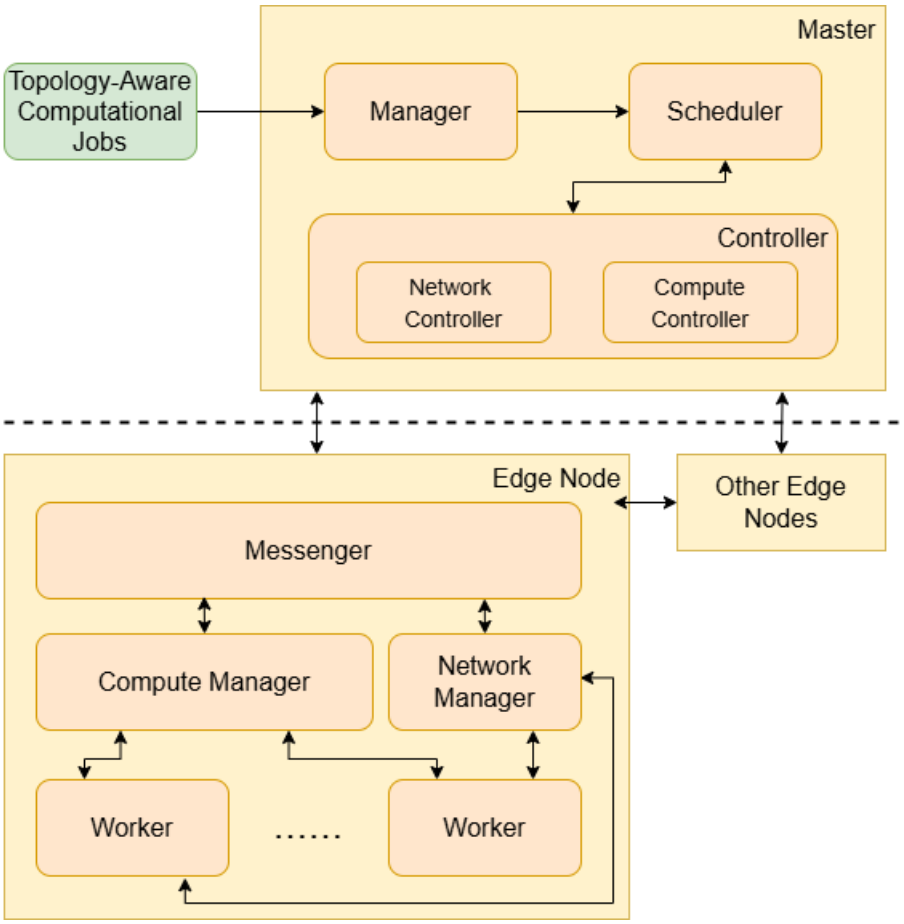

图 3-1 协同边缘任务调度系统 CES 架构图

3.2 系统架构

为高效协同异构、分布式的边缘计算资源，并满足分布式训练任务对计算与网络资源的联合需求，本系统采用经典的 Client-Server 架构进行设计。该架构通过中心化的协调与分布式的执行相结合，实现了资源管理的统一性与任务执行的可扩展性。系统主要由一个中心化的主节点（Master）和多个分布式的边缘节点（Edge Node）构成。其中，Master 节点作为系统的大脑，负责全局资源的管理、作业解析与全局任务调度；而 Edge Node 作为系统的四肢，负责接收调度指令，并通过容器化技术提供隔离、可控的执行环境来具体完成任务。整体架构的核心目标是在动态、异构的边缘环境中，实现计算与网络资源的协同分配与高效调度。CES 系统的具体架构如图3-1所示。

3.2.1 Master 节点设计

Master 节点是系统的控制中心，其设计着重于全局状态管理、决策制定以及与客户端的交互。它包含以下关键组件，各组件通过内部接口进行高效协作：

- **管理器（Manager）**：该组件作为系统对外的首要接口，承担作业的接入与解析职责。它接收用户提交的分布式训练作业描述，解析其中的任务、通信拓扑及资源需求。解析完成后，Manager 将结构化的作业信息传递给调度器（Scheduler），并负责将作业状态、执行结果等反馈给用户，管理作业的生命周期。
- **控制器（Controller）**：Controller 是资源管理的执行层，包含两个专用于处理不同资源维度的子控制器：
  - 计算资源控制器（Compute Controller）：该控制器基于 Docker 容器技术实现对底层异构计算资源的统一虚拟化抽象与管理。它负责根据调度器的指令，在指定的边缘节点上创建、启动、暂停或销毁 Worker 容器，从而为每个任务提供计算隔离的运行环境。同时，Compute Controller 持续从各边缘节点收集细粒度的资源监控数据（包括 CPU 利用率、内存占用等），不仅为 Scheduler 的决策提供实时、准确的依据，也确保了资源分配的公平性和隔离性。
  - 网络资源控制器（Network Controller）：为满足分布式训练中频繁的参数同步与数据交换对网络性能的严格要求，该控制器负责管理边缘节点间的网络资源。其功能涵盖带宽的分配、数据流的路由策略制定以及跨节点的流量转发控制。通过与计算资源的协同管理，Network Controller 旨在减少通信瓶颈，保障作业的网络服务质量（QoS）。
- **调度器（Scheduler）**：作为 Master 节点的核心决策组件，Scheduler 负责协调 Compute Controller 与 Network Controller，实施跨资源维度的联合调度。它持续收集来自各控制器的系统全局信息，包括但不限于所有边缘设备的实时资源利用率、网络链路状态与拓扑、以及排队等待作业的详细配置。基于这些信息，Scheduler 运行内嵌的任务调度算法，生成最优的作业调度策略与资源分配方案。其调度决策直接决定了任务被派往哪个边缘节点、获得多少资源以及使用怎样的网络路径，最终目标是优化系统整体的负载均衡度等关键性能指标。

3.2.2 Edge Node 节点设计

Edge Node 是部署在边缘侧的物理或虚拟设备，负责提供具体的计算和网络能力。每个 Edge Node 在设计上需具备自治的资源管理能力和与 Master 协同工作的通信能力，其内部组件如下：

- **通信器（Messenger）**：该组件是 Edge Node 与外界通信的枢纽。它一方面与 Master 节点保持持久的心跳连接，定期上报本节点的资源状态（通过 Manager 收集），并接收来自 Master 的调度指令与控制命令（如创建 Worker）；另一方面，在需要节点间直接通信的任务中（如分布式训练中的参数服务器与 Worker 之间），Messenger 也负责与其他 Edge Node 建立点对点的数据通信通道，高效传输中间数据或模型参数。
- **管理器（Manager）**：每个 Edge Node 拥有一个本地的管理器，它是 Master 端 Controller 在边缘侧的代理和执行延伸，也包含两个部分：
  - 计算资源管理器（Compute Manager）：负责管理本节点所有计算资源的具体分配。它接收并执行来自 Master Compute Controller 的容器操作指令，通过调用本地的 Docker 来实际创建和管理 Worker 容器。同时，它持续监控本节点各容器的资源消耗情况，并将聚合后的数据通过 Messenger 上报。
  - 网络资源管理器（Network Manager）：负责执行 Master Network Controller 下发的网络策略。它在本地通过流量控制工具 Traffic Control 来实施具体的带宽限制、流量整形或路由规则，确保节点出/入口的网络流量符合全局调度方案的要求。
- **工作器（Worker）**：Worker 是由 Docker 容器实例化的任务执行单元，是实际执行用户作业中具体任务的实体。每个 Worker 运行在资源隔离的容器环境中，内部包含用户指定的训练框架、应用程序代码及依赖库。Worker 根据作业逻辑进行本地计算，并通过 Messenger 组件与其他 Worker 或参数服务器进行通信，共同完成分布式训练任务。

综上所述，本系统通过 Master 节点的集中式智能调度与 Edge Node 节点的分布式高效执行相结合，构建了一个层次化、松耦合的协同计算框架。各组件各司其职又紧密联动，为在复杂边缘环境下开展资源敏感的分布式训练任务提供了坚实的系统基础。

3.3 任务调度流程

该系统的任务调度流程遵循从作业描述到资源分配、最终至任务执行的清晰逻辑过程，旨在实现跨资源维度的协同优化。整个过程可划分为三个阶段，如图 3 所示，其详细流程如下：

首先，在作业提交与解析阶段，用户通过客户端向 Master 节点的 Manager 组件提交分布式训练作业。该作业通常以资源描述文件进行定义，其中完整定义了作业的规格，包括：任务类型、待执行的程序实体（如镜像名称或脚本路径）、计算资源需求（例如，所需的 CPU 核数及内存大小）、任务间的网络带宽需求及通信拓扑。Manager 接收并解析此作业描述，将其转化为系统内部可识别的结构化任务元数据，为后续的智能调度提供精确的决策依据。

随后，进入资源联合调度与决策阶段，此为系统的核心环节。Master 中的 Scheduler（即联合调度器）被触发，它综合多源信息进行全局决策：

- **任务侧信息**：来自 Manager 的作业元数据，特别是任务资源需求与通信拓扑。
- **系统侧信息**：Controller 持续收集并维护的全局资源视图，包括各 Edge Node 的实时计算资源利用率、可用网络带宽以及链路拓扑。

基于上述信息，Scheduler 运行其内嵌的协同调度算法。该算法旨在平衡边缘节点的负载、最小化通信开销并满足资源约束，最终生成一个细粒度的调度策略。此策略具体规定了：1) 每个任务实例被分配到的目标 Edge Node；2) 为任务间关键数据流分配的带宽配额；3) 高效的数据传输路径或路由方案。此策略体现了系统“数据与计算协同感知”的核心设计思想。

最后，在任务分发、执行与监控阶段，调度策略被下发并执行。Master 的 Controller 将任务启动指令及资源配置要求下发至目标 Edge Node。各 Edge Node 的本地 Manager 协同工作：Compute Manager 根据指令通过 Docker 创建指定规格的 Worker 容器；Network Manager 则配置本地的流量控制规则以实施分配的带宽方案。任务开始在隔离的容器中执行，期间所有 Worker 的状态（如运行、完成、失败）及各节点的资源消耗数据通过 Messenger 组件实时反馈至 Master 的 Controller。这些动态信息构成了一个闭环反馈，使得 Scheduler 能够进行潜在的动态任务迁移或资源再分配，以应对负载波动或节点故障，从而保障系统的整体鲁棒性与执行效率。

综上所述，该工作流程通过“解析-决策-执行-反馈”的过程，实现了对边缘异构资

源的精细化管理与自适应调度，确保了分布式训练作业在复杂边缘环境中的高效、可靠运行。

3.4 本章小结

本章详细阐述了协同边缘计算任务调度系统 CES 的设计与实现。首先明确了系统面向弹性环境中具有通信依赖的分布式训练任务，确立了计算与网络资源联合调度的核心设计目标，具体包括提升真实环境调度可信度、资源利用均衡度与作业性能隔离。随后，系统采用主从架构，详细设计了中心化 Master 节点与分布式 Edge Node 节点的内部结构及交互机制，实现了全局资源统一视图与协同决策。最后，系统描述了从作业提交、资源联合调度到任务分发执行的完整调度流程，通过“解析-决策-执行-反馈”闭环机制保障了动态环境下的调度鲁棒性。本章为后续算法设计与实验验证提供了完整的系统支撑。

第四章　基于深度强化学习的联合任务调度算法

本章针对协同边缘计算中计算与网络资源联合调度的多目标优化问题，提出基于深度强化学习的求解方案 CES-PPO。首先对问题进行了建模，定义负载均衡度与带宽满足度两个核心目标及内在权衡。其次将调度问题转化为马尔可夫决策过程，设计状态、动作空间与分层奖励函数。提出基于近端策略优化的双分支策略网络的算法 CES-PPO，实现任务映射与带宽分配的协同决策，并引入动作掩码、启发式奖励等机制提升学习效率。通过仿真与真实环境实验，验证算法在综合性能上优于对比方法，消融实验证实启发式奖励的关键作用。

4.1　问题建模

我们为系统接收到的每个作业调度制定了一个联合调度问题，单作业调度的目标是通过决定在哪个物理节点分配作业的每个任务，以及任务间数据传输引起的每个数据流的带宽分配，在尽可能满足带宽需求的情况下使得系统更加负载均衡。Scheduler 维护两个作业队列：1) 正在运行的作业队列，记为 Q_{run}；2) 等待调度的作业队列，记为 Q_{wait}。在线调度算法对 Q_{wait} 中的作业进行逐个调度，完成调度的作业放入队列 Q_{run} 进行执行。

4.1.1　边缘环境

边缘环境包含物理节点集 P 与物理链路集 R。一个边缘环境包含多个物理节点和多条物理链路。每个物理节点 p 有固定的 CPU 资源 $CPU^P(p)$ 和内存资源 $RAM^P(p)$。链接两个物理节点 p_i 和 p_j 的物理链路集合记为 $R_{p_j}^{p_i} = \{r_a, r_b, \}$。每条物理链路是有向的，且有固定的带宽资源 $BW^R(r)$。

4.1.2　作业

作业节点集 N、作业链路集 V。一个作业中包含多个任务节点和多条任务链路。每个任务节点 n 有所需的 CPU 资源 $CPU^N(n)$ 和内存资源 $RAM^N(n)$。任务网络建模成 P2P 网络，相连的两个任务节点 n_i 和 n_j 之间仅存在一条链路，记为 $V_{n_j}^{n_i} = v$。每条链路是有向的，其所需的带宽资源是弹性的，最小带宽为 $BW_{min}^V(v)$，最大带宽为 $BW_{max}^V(v)$，动态分配变量 $B^V(v)$ 表示实际分配的带宽，且 $BW_{min}^V(v) \leq B^V(v) \leq BW_{max}^V(v)$。当两个任务节点映射到同一个物理节点时，他们之间的链路将不再消耗物理带宽资源，即可以最大地满足用户的带宽请求。带宽分配变量 $B(v)$ 由调度器动态决策，在满足

$BW^V_{min}(v) \leq B^V(v) \leq BW^V_{max}(v)$ 前提下优化全局目标。

4.1.3 任务映射与带宽分配的关系与条件

联合调度问题是在遵循特定约束条件的情况下，将作业的任务节点映射到物理节点上，并且为作业中的每一条任务网络边分配带宽。该过程可以分解为两个阶段，即任务节点映射和带宽分配。任务节点映射是指在资源约束条件下将每个工作的任务节点映射到物理节点，并通过 Dijkstra 算法为每条任务链路寻找两个任务节点映射到的物理节点之间的一条最短路径。不同的任务节点可以共享同一个物理节点，即任务节点到物理节点的映射关系：

$$\forall n \in N, \forall p \in P, X^n_p = \begin{cases} 1，\text{任务节点 } n \text{ 映射到物理节点 } p \text{ 上} \\ 0，\text{其他情况} \end{cases} \tag{4-1}$$

任务链路到物理链路的映射关系：

$$\forall n \in V, \forall r \in R, Y^v_r = \begin{cases} 1，\text{任务链路 } v \text{ 流经物理链路 } r \\ 0，\text{其他情况} \end{cases} \tag{4-2}$$

每个任务节点必须且仅能部署在一个物理节点上：

$$\forall n \in N, \sum_{p \in P} X^n_p = 1. \tag{4-3}$$

每个物理节点拥有的 CPU 资源、内存资源能够满足任务节点：

$$\forall p \in P, \sum_{n \in N} (X^n_p \times CPU^N(n)) \leq CPU^P(p). \tag{4-4}$$

$$\forall p \in P, \sum_{n \in N} (X^n_p \times RAM^N(n)) \leq RAM^P(p). \tag{4-5}$$

带宽分配是指为每条任务链路分配具体的带宽数值，若两个任务节点映射到同一个物理节点上，则该任务链路不消耗物理链路的带宽。任务链路带宽的分配范围：

$$BW^V_{min}(v) \leq B(v) \leq BW^V_{max}(v) \tag{4-6}$$

每条物理链路拥有的带宽资源能够满足任务链路：

$$\forall r \in R, \sum_{v \in V,\ n_i \neq n_j} (Y^v_r \times B^V(v)) \leq BW^R(r). \tag{4-7}$$

4.1.4 优化目标

联合调度问题的优化目标是在满足所有约束条件（式 (1)-(7)）的前提下，通过协同优化任务映射与带宽分配，实现系统整体效能的帕累托改进。我们主要关注两个关键且相互制衡的性能维度：系统负载均衡度与作业带宽需求满足度。这两个目标共同构成了联合调度问题的多目标优化框架。

4.1.4.1 负载均衡度

物理资源负载均衡度旨在衡量 CPU、内存等节点资源在物理基础设施中的利用均匀性。一个优秀的调度方案应避免将过多的任务节点堆积在少数物理节点上，导致“热点”产生，而应尽可能地将负载分散到多个节点上。该指标通过计算所有物理节点资源利用率的标准差来实现，其值越低，代表均衡性越好，系统整体可靠性和未来请求的接纳能力越强。

首先定义三类资源的利用率：

$$U^{\text{CPU}}(p) = \frac{\sum_{n \in N} \left(X_p^n \cdot \text{CPU}^N(n) \right)}{\text{CPU}^P(p)}, \tag{4-8}$$

$$U^{\text{RAM}}(p) = \frac{\sum_{n \in N} \left(X_p^n \cdot \text{RAM}^N(n) \right)}{\text{RAM}^P(p)}, \tag{4-9}$$

$$U^{\text{BW}}(r) = \frac{\sum_{v \in V} \left(Y_r^v \cdot B(v) \right)}{\text{BW}^R(r)} \tag{4-10}$$

对应的平均利用率为：

$$\overline{U^{CPU}} = \frac{1}{|P|} \sum_{p \in P} U^{CPU}(p), \tag{4-11}$$

$$\overline{U^{RAM}} = \frac{1}{|P|} \sum_{p \in P} U^{RAM}(p), \tag{4-12}$$

$$\overline{U^{BW}} = \frac{1}{|R|} \sum_{r \in R} U^{BW}(r) \tag{4-13}$$

最终，负载均衡度通过三个维度的标准差加权和来综合衡量：

$$L^{CPU} = \sqrt{\frac{1}{|P|} \sum_{p \in P} (U^{CPU}(p) - \overline{U^{CPU}})^2}, \tag{4-14}$$

$$L^{RAM} = \sqrt{\frac{1}{|P|}\sum_{p\in P}(U^{RAM}(p) - \overline{U^{RAM}})^2}, \tag{4-15}$$

$$L^{BW} = \sqrt{\frac{1}{|R|}\sum_{r\in R}(U^{BW}(r) - \overline{U^{BW}})^2} \tag{4-16}$$

系统的整体负载均衡度定义为：

$$L = w_1 \cdot L^{CPU} + w_2 \cdot L^{RAM} + w_3 \cdot L^{BW} \tag{4-17}$$

其中 w_1, w_2, w_3 为权重系数，满足 $w_1 + w_2 + w_3 = 1$。优化目标为最小化 L。

4.1.4.2 带宽需求满足度

带宽需求满足度用于评估调度方案对作业中任务链路服务质量 (QoS) 的满足程度。在资源允许的条件下，调度器应尽可能为任务链路分配接近其最大需求 $BW^V_{max}(v)$ 的带宽，以提供更优的网络性能。该指标反映了系统中所有任务链路的带宽需求得到满足的平均程度，其值越高代表链路服务质量越好。

对于每条任务链路 $v \in V$，其带宽需求满足度定义为：

$$\delta(v) = \begin{cases} 1 & \text{if } BW^V_{\max}(v) = BW^V_{\min}(v) \\ \frac{B(v) - BW^V_{\min}(v)}{BW^V_{\max}(v) - BW^V_{\min}(v)} & \text{otherwise} \end{cases} \tag{4-18}$$

整个作业的带宽需求满足度则为所有任务链路满足度的平均值：

$$D_{BW} = \frac{1}{|V|}\sum_{v\in V}\delta(v) \tag{4-19}$$

优化目标为最大化 D_{BW}。

4.1.4.3 多目标权衡关系

这两个优化目标之间存在内在的权衡关系。当多个任务节点被映射到同一物理节点时，它们之间的任务链路将不再消耗物理带宽资源，这可以显著提高带宽需求满足度。然而，这种做法同时会加剧该物理节点上计算资源的负载集中度，从而降低负载均衡指标。反之，将任务节点分散部署到不同物理节点虽有利于负载均衡，却可能因任务链路跨越物理链路而消耗带宽资源，降低带宽需求满足度。

因此，联合调度问题的本质是在这两个竞争性目标之间寻找帕累托最优解集，即在满足系统资源约束的前提下，寻求负载均衡度 L 与带宽需求满足度 D_{BW} 的最佳权衡。

该问题可形式化为如下多目标优化问题：

$$\begin{aligned} &\text{Minimize} && L \\ &\text{Maximize} && D_{BW} \\ &\text{s.t.} && \text{约束条件 (1)-(7)} \end{aligned} \tag{4-20}$$

后续的调度算法设计将围绕如何有效求解这一多目标优化问题展开。

4.2 基于 PPO 的联合资源调度算法设计

该联合调度问题是一个具有 NP-hard 复杂性的多目标组合优化问题，需要在动态边缘环境中，为在线到达的作业实时协同决策任务映射与带宽分配。问题的挑战在于，负载均衡与带宽需求满足这两个竞争目标之间存在内在权衡，且决策空间混合了离散映射与连续带宽变量。传统优化方法难以在满足实时性约束的同时有效处理这一复杂权衡。因此，本文提出了一种基于近端策略优化（PPO）的深度强化学习求解框架，其能够通过端到端的训练，学习在动态环境下实时输出联合调度策略，并自适应地平衡多目标优化与硬资源约束，为求解该问题提供了高效且自适应的新途径。

4.2.1 问题转化与 MDP 建模

针对协同边缘计算环境中资源调度这一 NP 难组合优化问题，本文将复杂的联合优化问题建模为马尔可夫决策过程（MDP），以便利用深度强化学习进行高效求解。MDP 由元组 $\langle S, A, P, R, \gamma \rangle$ 定义，其中状态空间 S 全面表征环境信息，动作空间 A 对应调度决策，状态转移概率 P 描述环境动态，奖励函数 R 引导多目标优化，折扣因子 γ 平衡短期与长期收益。算法采用集中式智能体架构，通过与环境交互学习最大化累积奖励的策略，在满足资源约束的前提下实现负载均衡与带宽需求满足度的协同优化。

4.2.1.1 状态空间设计

状态空间需要同时捕捉物理环境、作业需求与决策进度三方面信息：

边缘环境状态：每个物理节点的可用 CPU、可用内存以及关联链路的平均可用带宽，构成维度为 $[|P|, 3]$ 的矩阵。

作业状态：每个任务节点的 CPU 需求、内存需求及其出链路的平均带宽需求，构成维度为 $[|N|, 3]$ 的矩阵。

决策过程状态：包括当前决策阶段（任务节点映射或带宽分配）、当前处理的任务

节点/链路索引、任务节点/链路总数以及已完成映射的比例。

为处理可变规模的作业，采用零填充将特征矩阵统一到最大维度，并进行归一化以提升训练稳定性。完整状态 $s \in S$ 可表示为：

$$s = [s_p; s_v; s_d] \tag{4-21}$$

其中 s_p 为物理环境状态矩阵，s_v 为作业状态矩阵，s_d 为决策过程状态向量，总维度为 $3|P| + 3|N| + 5$。

4.2.1.2 动作空间设计

动作空间采用两阶段离散化设计，分别对应任务映射与带宽分配：

任务节点映射阶段：从物理节点集合中选择一个节点部署当前任务节点，动作空间大小为 $|P|$，动作表示为：

$$a_{map} \in \{0, 1, \ldots, |P| - 1\} \tag{4-22}$$

带宽分配阶段：虽然原始问题中带宽需求为连续区间 $[BW^V_{min}(v), BW^V_{max}(v)]$，但为提升算法训练稳定性和计算效率，采用离散化处理。该设计将连续带宽需求均匀划分为 L 个等级，从预设离散带宽等级中选择分配等级，动作空间大小为 L，动作表示为：

$$a_{bw} \in \{0, 1, \ldots, L - 1\} \tag{4-23}$$

实际分配的带宽值为：

$$B(v) = BW^V_{min}(v) + \frac{a_{bw}}{L - 1} \times (BW^V_{max}(v) - BW^V_{min}(v)) \tag{4-24}$$

此离散化设计将无限连续动作空间压缩为有限离散集合，显著降低搜索复杂度，同时通过调整等级数量 L 可在精度与效率间取得平衡。两个阶段共享状态编码，但通过独立的策略头生成动作概率分布，实现阶段专业化决策。

4.2.1.3 奖励函数框架

奖励函数是引导智能体学习的关键。本文设计了一个分层、多目标的奖励框架，将最终的系统优化目标（负载均衡度 L 与带宽满足度 D_{BW}）分解为可微分的即时信号。总奖励由步骤奖励、启发式奖励和最终奖励三部分构成：

$$R_{\text{total}} = R_{\text{step}} + \lambda_{\text{heuristic}} \cdot R_{\text{heuristic}} + R_{\text{final}} \tag{4-25}$$

其中，$\lambda_{\text{heuristic}}$ 是一个采用线性衰减调度的权重系数，在训练初期设定较高（如 0.8），以注入强领域先验知识，引导智能体快速避开无效搜索空间；随着训练进行，其权重逐步降低（至 0.1），促使智能体更多依赖从环境交互中学到的长期策略，从而平衡” 引导学习” 与” 自主探索”。

4.2.2 整体算法框架

本文提出的基于 PPO 的联合资源调度算法框架如图4-1所示。算法核心是通过深度强化学习实现计算资源与网络资源的协同调度，将传统 NP 难的组合优化问题转化为序列决策过程。

图 4-1 基于 PPO 的联合资源调度算法框架

算法4-1展示了整体调度流程：

4.2.3 双分支策略网络架构设计

为有效处理本问题中混合（离散映射与连续分配）且分阶段的复杂决策过程，本文设计了一种双分支策略网络架构，其结构如图4-2所示。该网络的核心设计理念是“共享表征、分头决策”，通过一个共享的编码器提取状态的高层特征，随后由两个独立的策略头分别负责节点映射与带宽分配的决策生成。这种设计兼顾了两种决策间的关联性学习与各自动作空间的特殊性。

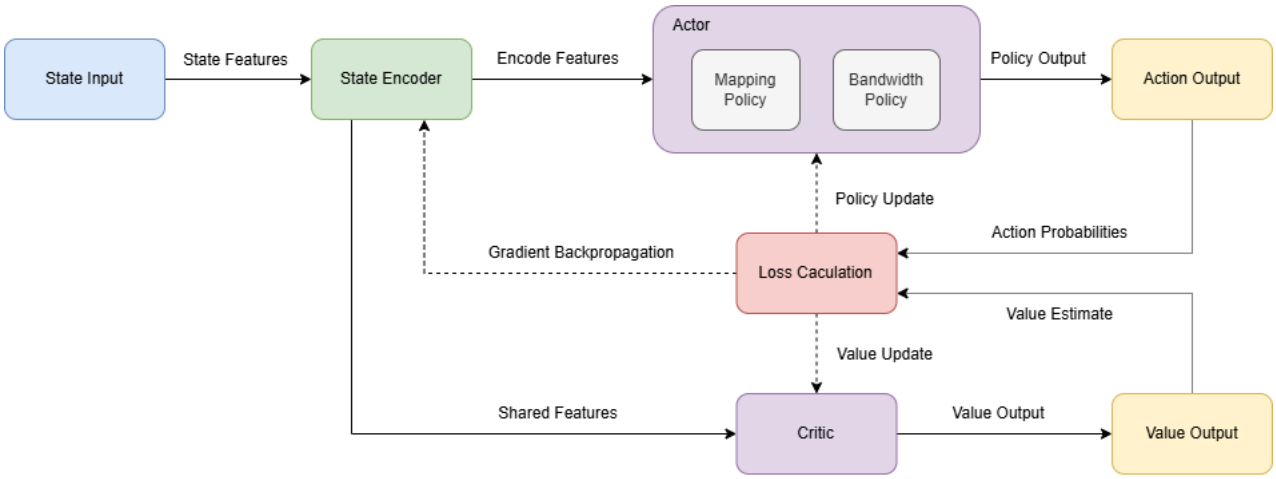

图 4-2 双分支策略网络架构

Algorithm 4-1 基于 PPO 的联合资源调度算法

Input: 物理网络拓扑G_P, 虚拟网络请求G_V
初始化π_θ, V_φ
for $episode = 1$ **to** N **do**
    // 步骤 1：环境初始化
    $state \leftarrow env.reset(G_P, G_V)$
    // 步骤 2：序列决策过程
    for $step = 1$ **to** M **do**
        // 2.1 状态编码与动作选择
        if 当前阶段 = 节点映射阶段 **then**
            $action \leftarrow \pi_\theta(state)$ ▷ 选择物理节点
            执行节点映射，更新资源约束
        else
            $action \leftarrow \pi_\theta(state)$ ▷ 选择带宽等级
            执行带宽分配，更新链路约束
        end if
        // 2.2 接收奖励与环境转移
        $reward \leftarrow calculate\_reward(state, action)$
        $next\_state \leftarrow env.step(action)$
        // 2.3 存储经验数据
        存储 $(state, action, reward, next\_state)$
    end for
    // 步骤 3：PPO 策略更新
    计算优势函数 $A_t = R_t - V_\varphi(s_t)$
    计算策略损失 $L^{CLIP}(\theta)$
    更新 π_θ 与 V_φ
end for
Output: 节点映射X, 带宽分配B

编码器模块：作为网络的公共特征提取器，编码器 $\phi(\cdot)$ 接收并融合来自物理网络、虚拟作业及决策过程的多维异构状态信息 s。它通过多层感知机将高维原始状态映射为一个紧凑且信息丰富的隐藏表示 h，该过程可表示为：

$$h = \text{ReLU}(W_2 \cdot \text{ReLU}(W_1 \cdot s + b_1) + b_2) \tag{4-26}$$

其中，W_1, b_1, W_2, b_2 为可学习参数。编码器模块的作用在于从复杂的状态中学习并抽象出对下游两个决策任务均有价值的通用特征，从而避免了两任务分别进行特征学习的参数冗余与信息割裂。

策略头分支：在获得共享的隐藏表示 h 后，网络分为两个独立的策略头，分别生成节点映射和带宽分配的动作概率分布。

- **映射策略头** π_{map}：负责在任务节点映射阶段，基于当前状态 s 的隐藏表示 h，计算将任务节点部署到各个物理节点 $p \in P$ 的概率。其输出维度为 $|P|$，即物理节点的数量：

$$\pi_{map}(a|s) = \text{Softmax}(W_{map} \cdot h + b_{map}) \tag{4-27}$$

- **带宽策略头** π_{bw}：负责在带宽分配阶段，基于相同的 h，计算为当前任务链路选择不同带宽等级 $l \in 0, 1, \ldots, L-1$ 的概率。其输出维度为 L：

$$\pi_{bw}(a|s) = \text{Softmax}(W_{bw} \cdot h + b_{bw}) \tag{4-28}$$

两个策略头共享编码器的输出 h，确保了节点映射决策所产生的资源占用、负载分布等信息能直接影响后续带宽分配的决策，实现了两阶段决策的隐式协同与信息贯通。同时，独立的参数（W_{map}, b_{map} 与 W_{bw}, b_{bw}）使得每个头可以专注于学习其特定动作空间的最优策略。

价值网络：在演员-评论家框架中，价值网络用于评估当前状态 s 的长期期望回报，为策略更新提供基线（Baseline），以降低梯度估计的方差并加速训练。本算法中的价值网络与策略网络共享编码器，同样以隐藏表示 h 作为输入，通过一个独立的回归头预测状态价值 $V(s)$：

$$V(s) = W_{val} \cdot \text{ReLU}(W_{val_hidden} \cdot h + b_{val_hidden}) + b_{val} \tag{4-29}$$

共享编码器的设计使得价值估计能够建立在与策略决策相同的特征理解之上，提高了价值预测的准确性和与策略学习的一致性。

动作掩码机制：为确保智能体的所有决策均满足实时资源约束，本文引入了动作掩码机制。该机制在智能体进行决策前，根据当前物理节点的可用资源（CPU、内存）或物理链路的剩余带宽，动态地为每个可能动作计算一个二进制掩码 $mask(a)$：若动作 a 对应的资源需求超出可用资源，则 $mask(a) = 0$，否则 $mask(a) = 1$。在策略网络输出动作概率前，通过将无效动作（$mask(a) = 0$）对应的 logits 设置为负无穷，确保其在 Softmax 后的概率为零，从而引导智能体仅从可行的动作空间中采样，有效避免了违反资源约束的无效探索，提升了训练效率与策略的实用性。

4.2.4 奖励函数设计与启发式奖励

奖励函数 $R(s,a)$ 设计为多目标加权和，引导策略同时优化负载均衡和带宽满足度。启发式奖励 $R_{\text{heuristic}}$ 并非单一的奖励项，而是一个根据当前决策阶段（映射或分配）动态计算的、融合了多种领域知识的复合奖励，旨在直接而高效地促进两个核心优化目标。

4.2.4.1 基础奖励

基础奖励 R_{base} 为智能体的单步决策提供即时的可行性反馈与质量引导。具体而言，节点映射奖励 R_{map} 由成功指示函数、CPU 利用率与理想值（$U_{ideal}=0.625$）的接近度、以及内存利用率与理想值的接近度三部分加权构成，旨在鼓励成功映射的同时引导资源使用趋向均衡高效；带宽分配奖励 R_{bw} 则定义为实际分配带宽 $B(v)$ 在其需求区间 $[BW^V_{min}(v), BW^V_{max}(v)]$ 内的归一化满足度，直接反映了当前链路服务质量需求的达成情况。

4.2.4.2 启发式奖励设计

启发式奖励 $R_{\text{heuristic}}$ 旨在将领域知识编码为可学习的梯度信号，通过更精细化的引导加速智能体对关键调度目标的掌握。该奖励根据决策阶段动态计算，分为节点映射与带宽分配两个部分。

任务节点映射阶段：当为任务节点 n 选择物理节点 p 时，启发式奖励 $R^{\text{map}}_{\text{heuristic}}$ 从负载均衡、资源匹配与拓扑优化三个维度提供引导，其计算方式为：

$$R^{\text{map}}_{\text{heuristic}} = \eta_1 \cdot R_{\text{load}} + \eta_2 \cdot R_{\text{affinity}} + \eta_3 \cdot R_{\text{topo}} \tag{4-30}$$

其中各项含义如下：

- **负载均衡奖励** R_{load}：鼓励将任务部署到当前负载较低的物理节点，通过惩罚选择高负载节点的行为，促进节点间资源利用的均衡分布。
- **资源亲和奖励** R_{affinity}：通过计算任务资源需求向量与节点剩余资源向量的余弦相似度，鼓励大任务优先匹配资源充足的节点，提高资源分配效率。
- **拓扑优化奖励** R_{topo}：针对已存在高带宽需求链路的任务对，鼓励将其映射到相同或邻近节点，从而减少后续带宽分配阶段对物理链路资源的消耗。

带宽分配阶段：当为任务链路 v 分配带宽等级时，启发式奖励 $R^{\text{bw}}_{\text{heuristic}}$ 侧重于带宽

资源的优化配置，其计算方式为：

$$R^{\text{bw}}_{\text{heuristic}} = \eta_4 \cdot R_{\text{importance}} + \eta_5 \cdot R_{\text{compete}} \tag{4-31}$$

其中各项含义如下：

- **链路重要性奖励** $R_{\text{importance}}$：根据链路最大带宽需求占作业总带宽需求的比例定义其重要性，为核心链路分配更高带宽时给予更高奖励，优先保障关键数据流的服务质量。
- **路径竞争减免奖励** R_{compete}：通过检查所选物理路径上各链路的剩余带宽比例，鼓励选择竞争程度较低的路径，为后续链路预留资源，提升系统整体的可扩展性。

通过上述模块化的启发式奖励设计，算法将高层优化目标（负载均衡与带宽满足）转化为与具体决策上下文紧密相关的即时学习信号，结合自适应权重衰减策略，在训练初期有效引导智能体避开无效搜索空间，后期则鼓励其基于环境反馈自主优化，最终实现高效、均衡的联合调度策略。

4.2.4.3 最终奖励

回合结束时的最终奖励 R_{final} 评估整体调度质量：

$$R_{final} = \gamma_1 \cdot L_{balance} + \gamma_2 \cdot D_{BW} + \gamma_3 \cdot U_{avg} \tag{4-32}$$

其中，$L_{balance}$ 为负载均衡度，D_{BW} 为带宽满足度，U_{avg} 为平均资源利用率。

4.2.5 训练优化策略

为提升算法训练效率并确保最终策略的质量，本文设计了多项训练优化策略，包括自适应权重调度、课程学习机制和温度调度策略。

4.2.5.1 自适应权重调度

在奖励函数设计中，启发式奖励的权重 $\lambda_{\text{heuristic}}$ 并非固定不变，而是采用线性衰减策略进行动态调整。具体而言，在训练初期设定较高的初始权重 λ_{initial}，以强化领域知识的引导作用，帮助智能体快速建立有效的决策模式；随着训练步数 t 的增加，权重逐渐线性下降，直至达到预设的最终权重 λ_{final}。这一过程使得智能体在训练后期能够减少对启发式规则的依赖，更多地依据环境反馈进行自主探索与优化，从而在引导学习与自主探索之间取得良好平衡。

4.2.5.2 PPO 优化目标与优势估计

本文采用近端策略优化（PPO）算法的裁剪目标函数来更新策略网络参数。PPO 通过限制每次策略更新的幅度来确保训练稳定性，其核心优化目标函数为：

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{4-33}$$

其中，$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ 为新旧策略的概率比，$\hat{A}_t$ 为优势函数估计，$\epsilon = 0.2$ 为裁剪参数。该目标函数通过裁剪机制限制概率比 $r_t(\theta)$ 在区间 $[1-\epsilon, 1+\epsilon]$ 内，防止因策略更新步幅过大而导致性能震荡，从而在保证学习效率的同时维持训练稳定性。

为了准确评估动作的长期价值，算法采用广义优势估计（GAE）计算优势函数 $\hat{A}_t$：

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{4-34}$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{4-35}$$

其中，δ_t 为时序差分误差，γ 为折扣因子，$\lambda = 0.95$ 为权衡参数。GAE 通过指数加权多步时序差分误差，在偏差与方差之间取得平衡，为策略梯度提供了低方差、低偏差的优势估计，从而引导策略向更高长期回报的方向更新。

4.2.6 算法复杂度分析

本节从时间和空间两个维度分析所提出算法的计算复杂度。时间复杂度方面，训练阶段的前向传播和反向传播复杂度分别为 $O(d_{\text{state}} \times d_{\text{hidden}} + d_{\text{hidden}}^2)$ 和 $O(B \times d_{\text{hidden}}^2)$，其中 B 为批次大小；推理阶段的单次决策复杂度为 $O(d_{\text{state}} \times d_{\text{hidden}})$，完成一个作业的完整调度复杂度为 $O((|N|+|V|) \times d_{\text{state}} \times d_{\text{hidden}})$，可实现毫秒级实时决策。空间复杂度主要包括模型参数存储 $O(d_{\text{state}} \times d_{\text{hidden}} + d_{\text{hidden}}^2)$ 以及训练时的经验缓冲区 $O(B \times T_{\max} \times d_{\text{state}})$。尽管训练阶段需要较多计算资源，但推理阶段的高效性使得本算法在实际部署中相比传统启发式算法具有显著优势。

4.3 实验结果与分析

本节通过仿真实验与真实环境场景实验，对 CES-PPO 算法进行验证。

4.3.1 对比方法

为全面评估本文提出的算法性能，本文选取以下 4 种对比算法：

（1）PPO Balance 算法：基于 A2C-DRL[28]算法思想实现的 PPO 调度算法。其策略

网络将节点资源状态与任务需求编码为输入，输出各节点的选择概率，并采用贪心策略完成映射，训练目标已融合负载均衡与网络效率优化。带宽分配阶段则采用启发式规则，依据虚拟节点在请求拓扑中的重要性（如节点度）为其物理链路分配带宽，重要性高的链路获得更高带宽。

（2）FlexiTask 算法[22]：模仿 Kubernetes 的两阶段调度框架。预选阶段过滤出满足资源需求与负载阈值（如短期平均与长期峰值利用率）的候选节点；优选阶段综合资源空闲度、均衡度及节点选择热度等因素进行评分，选择得分最高的节点进行映射。

（3）Smart 算法：基于多因素加权评分的决策方法。对每个物理节点分别计算资源效率、负载均衡性、网络连接性与资源余量四个维度的分数，按预设权重加权求和，选择总分最高的节点；带宽分配阶段则依据虚拟节点的重要性进行差异化分配。

（4）Random 算法：作为基准对比方法，完全不进行优化。节点映射阶段在所有物理节点中随机选择；带宽分配阶段在可用带宽等级中随机分配，体现最基础的随机调度行为。

4.3.2 仿真实验

4.3.2.1 实验环境

实验构建了基于 Python 的协同边缘调度仿真环境。物理网络由 10 个节点组成，节点间拓扑通过随机连接生成（连通概率为 0.3），并确保整个网络连通。各物理节点的计算资源与内存资源总量在 [50, 100] 范围内随机分配，且在仿真初始化时，每个节点已有部分资源被占用，其已使用量占该节点资源总量的比例随机处于 [10%, 50%] 之间。物理链路的总带宽在 [100, 1000] 范围内随机设定。每个作业包含 3 至 8 个任务节点（数量随机确定），每个任务节点对 CPU 和内存的资源请求均为 [10, 50]。任务之间的通信链路设有最低与最高带宽请求，分别为 [10, 50] 和 [50, 100]。任务节点之间以概率 0.4 随机建立连接，并确保任务子图连通。

实验以任务节点数量和物理环境负载程度作为变量，在每个规模下对比多种方法。使用基础随机种子 seed=42，并通过调整每轮仿真的内部种子进行多次独立运行。每种算法均执行 30 轮仿真，最终对负载均衡度、带宽满意度及综合指标取 30 次结果的平均值与标准差，作为各任务规模下的性能评估依据。

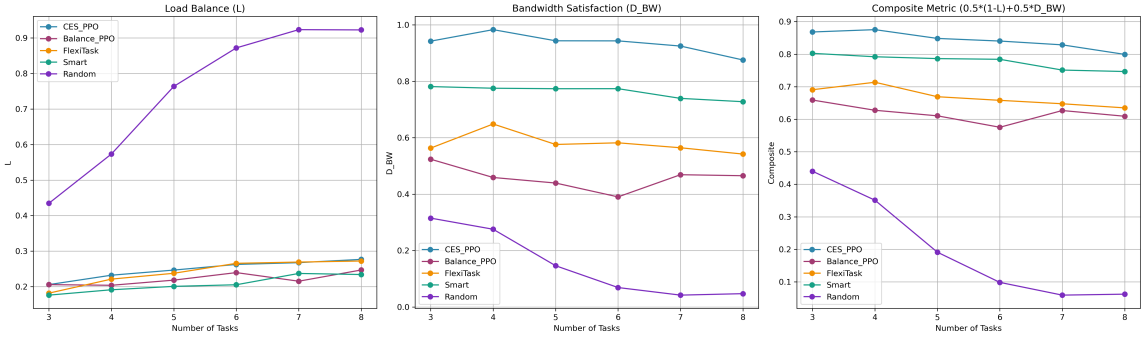

图 4-3 不同任务节点数量下的仿真实验结果

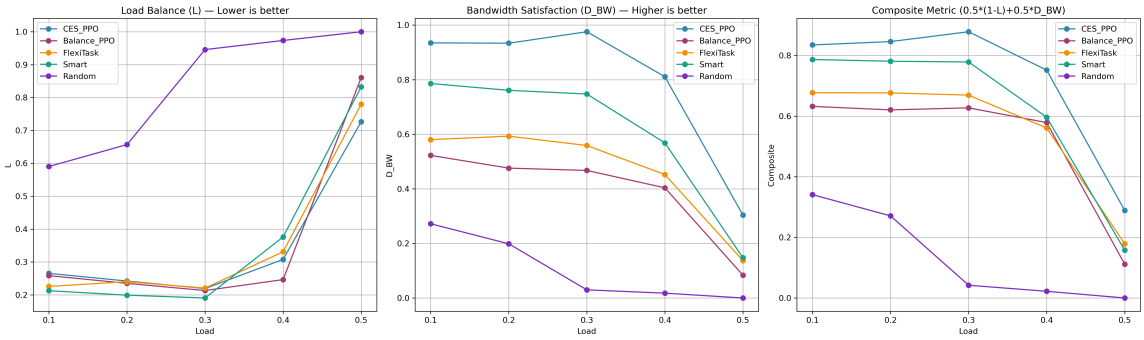

图 4-4 不同物理环境负载程度下的仿真实验结果

4.3.2.2 实验分析

仿真实验结果如图4-3和图4-4所示。综合分析表明，本文提出的计算与网络资源联合调度算法（CES-PPO）在不同任务节点规模以及不同物理环境负载程度下，其综合性能均展现出显著且稳定的优势。这一优势源于联合调度框架能够在对节点计算资源进行映射决策的同时，一体化评估并优化物理网络的链路带宽分配，从而更精准地满足通信约束。在带宽满足度方面，CES-PPO 在所有实验场景下均显著优于所有对比基线。这得益于算法在奖励函数与决策过程中对网络链路状态的显式建模与优化，使其能够精准适配动态复杂的带宽约束。从图4-3可见，随着任务节点数增加，CES-PPO 的带宽满足度保持领先；从图4-4可见，在不同物理环境负载程度下，CES-PPO 的带宽满足度同样最优，Smart 次之，而 FlexiTask 与 PPO-Balance 表现相近且位于第三梯队。在负载均衡度方面，各算法的表现整体较为接近，且均随任务规模增大或物理负载升高而有所下降。CES-PPO 的负载均衡度与 PPO-Balance、FlexiTask 及 Smart 等优化算法平均水平相当，但略低于其中表现最好的算法。这反映了 CES-PPO 在多目标优化中的权衡策略：为保障对通信性能更为关键的带宽需求，在绝对意义上的节点间负载均衡性上做出了有意识、可控的妥协。从综合指标（$0.5 \times (1 - L) + 0.5 \times D_{BW}$）来看，CES-PPO 在所有

实验场景下均取得最优值，Smart 次之，FlexiTask 与 PPO-Balance 则表现相近且处于第三梯队。这一结果充分验证了联合调度机制的有效性：通过在一个决策循环内协同处理计算与网络资源，算法能够实现整体系统效能的帕累托改进，避免传统分阶段调度可能带来的目标冲突与次优解。

4.3.3 真实环境实验

4.3.3.1 真实环境实验

在真实部署实验中，我们以协作训练 Fashion-MNIST 图像分类模型为目标，采用了 Gossip Learning、Ring All-Reduce 与 E-Tree Learning 三种不同通信架构的分布式学习任务；硬件方面，实验由一个配备 AMD Ryzen Threadripper 3990X 64 核处理器的 Master 服务器以及 10 台通过有线千兆局域网全连接的异构 Jetson 设备（4 台 Nano 与 6 台 Xavier NX）组成，并使用 Tailscale 进行组网；作业则被设置为固定包含 5 个任务节点，其资源请求与仿真环境一致，任务节点间的连接拓扑根据所采用的训练架构随机生成。负载以 Docker 容器（stress:latest）形式运行在边缘节点上，用于模拟后台计算与网络负载。

4.3.3.2 实验分析

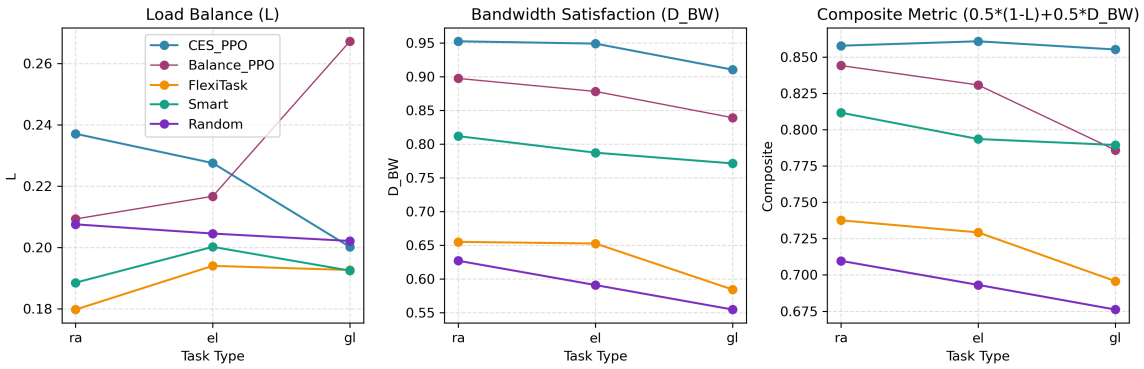

图 4-5 真实环境实验

真实环境下的实验结果如图4-5所示。总体来看，本文算法在三种分布式学习架构下的综合指标均保持领先，验证了其在实际异构边缘环境中的有效性与鲁棒性，这尤其突显了算法应对异构资源需求与复杂通信拓扑的实用性。具体分析如下：在带宽满足度上，本文算法相较于对比方法有明显提升，这与仿真实验结论一致，凸显了其网络优化能力的泛化性。在负载均衡度上，本文算法表现处于可接受范围，虽未达到最优，但通过与高带宽满足度的结合，最终在综合指标上取得了最佳平衡。值得注意的是，算法在不同通信拓扑下展现了差异化的特性：在 Gossip Learning 架构下，其负载均衡度优于

PPO-Balance 算法；而在 E-Tree Learning 架构下，其综合性能与 PPO-Balance 相当。这些结果表明，本文算法能够自适应地处理不同通信模式带来的调度挑战，尤其擅长在存在大量随机或复杂通信需求的场景中协调资源。实验证实，本文算法能够很好地集成到 CES 系统中，并有效支撑真实的分布式学习任务。

4.3.4 消融实验

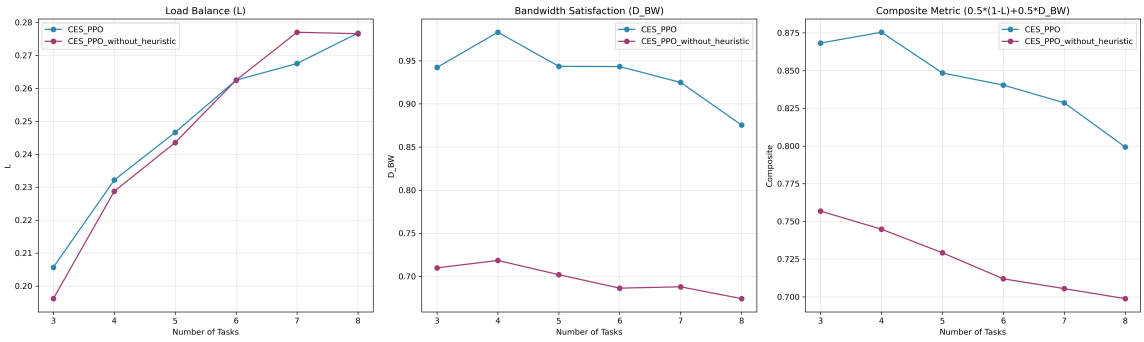

图 4-6 消融实验

为验证算法中启发式奖励机制的核心作用，我们进行了消融实验，结果如图4-6所示。实验完整对比了引入启发式奖励的算法与将其移除后的基线版本。实验结果表明，启发式奖励对算法性能起到了关键的促进作用。具体而言：在带宽满足度上，完整算法相比基线有显著提升，这直接证明了该奖励项能有效引导智能体在策略探索中，优先关注网络链路带宽的分配效率与约束满足。与此同时，在负载均衡度方面，完整算法与基线模型仍保持相近水平，说明其性能增益并未以严重牺牲负载均衡为代价。以上结果充分证实，我们所设计的启发式奖励机制成功地在“负载均衡”与“带宽满足”这两个存在一定冲突的目标之间实现了高效权衡。更进一步，该奖励函数的核心价值在于引导调度器将计算资源与网络资源视为相互关联的联合决策变量进行整体评估，而非彼此独立的优化维度，从而驱动智能体学习到整体更优的联合调度策略。

4.4 本章小结

本章围绕协同边缘计算中计算与网络资源的联合调度问题展开研究。首先对问题进行严格形式化定义，明确多目标优化框架。基于深度强化学习将调度问题建模为马尔可夫决策过程，设计了双分支策略网络的 PPO 求解算法 CES-PPO，实现任务映射与带宽分配的联合优化。引入启发式奖励、动作掩码等机制，有效引导智能体学习高效可行策略。仿真与真实环境实验表明，所提算法在负载均衡度与带宽满足度综合指标上显著优于 PPO-Balance、FlexiTask 等对比方法，验证了联合调度机制在处理通信依赖任务时的

优越性。消融实验进一步证实启发式奖励设计的关键贡献。

第五章　多任务队列调度算法

针对协同边缘计算环境中多分布式学习作业的在线调度问题，本章提出一种融合滚动窗口、资源感知贪心排序与动态饥饿保护的混合启发式调度算法。该算法在 CPU、内存、链路带宽等多维资源约束下，联合优化带宽满足度、系统负载均衡度与作业平均等待时间三个相互冲突的指标。首先，基于系统负载动态确定候选作业窗口，平衡决策复杂度与调度机会；其次，通过资源适配度、业务优先级与等待因子计算综合评分，优先调度与剩余资源最匹配的作业；然后，引入跳过计数阈值与强制预留机制防止低优先级作业饥饿。在本章的最后，我们通过仿真实验验证算法在不同环境下的性能优势，并与多种基线调度策略进行对比分析。该算法在 CES 单作业调度器上进行了多作业的拓展，具有良好的可扩展性与实用性。

5.1　问题建模

随着深度学习模型在边缘智能场景中的广泛应用，越来越多的分布式学习作业需要部署在资源受限的边缘节点上。这些作业通常包含多个协同计算的任务节点及任务间的通信链路，对计算资源（CPU、内存）与网络资源（带宽）均有需求。如何在满足物理资源约束的前提下，高效、公平地调度多批作业，已成为边缘计算资源管理领域的关键挑战。

现有研究往往将作业调度分解为两个独立阶段：先决定作业的执行顺序，再为单个作业分配计算节点与带宽资源。这种分阶段方法容易造成目标冲突——例如，仅按优先级排序可能忽略资源匹配度，导致资源碎片化；仅按资源需求大小排序则可能使大作业长期阻塞小作业。此外，多数调度器未显式考虑作业间的带宽竞争，或仅以最小带宽保障为目标，难以逼近最大带宽需求。因此，我们在第四章提出的单作业调度算法基础上，设计了一种全新的多作业在线调度算法，旨在同时优化带宽满足度、负载均衡度与平均等待时间三个核心指标。

为解决上述问题，本文提出一种多作业队列在线调度算法 CES-Multi-Job，其核心内容包括：

- **联合调度框架**：将计算资源与网络资源统一纳入调度决策，避免分阶段优化的次优性；
- **动态滚动窗口**：根据当前运行队列的负载情况自适应调整候选作业窗口大小，在

决策开销与调度性能间取得平衡；

- **资源感知评分**：从 CPU、内存、带宽三个维度计算作业需求与系统剩余资源的匹配程度，优先调度最能填补资源空缺的作业；
- **饥饿保护机制**：通过跳过计数与等待时间监控，对长期未获调度的作业实施优先级提升或强制资源预留，保障公平性。

本文所述算法与现有的单作业调度算法（如第四章中的基于强化学习的 PPO 调度算法）完全兼容，可作为上层调度器集成至 CES 等边缘计算平台。

5.1.1 系统模型

考虑一个协同边缘计算系统，包含一个中心调度器及若干边缘节点。系统维护两个队列：等待调度队列 Q_{wait} 存放已到达但尚未获得资源的作业；运行队列 Q_{run} 存放正在执行的作业。调度器在每个离散时间步 t 观察系统状态，从 Q_{wait} 中选择若干作业尝试分配资源，成功则移入 Q_{run}。每个作业 j 包含一个任务节点集合 N_j 和一个通信链路集合 V_j，分别描述作业内部的计算与通信需求。调度器需在满足物理资源约束的前提下，为每个任务节点分配一个物理节点，并为每条虚拟链路分配物理链路上的带宽资源。

5.1.1.1 物理网络资源

本节定义物理网络资源的描述方式。物理网络由节点和链路构成，分别以集合 $P = \{p_1, p_2, \ldots, p_M\}$ 和 $R = \{r_1, r_2, \ldots, r_L\}$ 表示，其中 $M = |P|$ 为节点总数，$L = |R|$ 为链路总数。所有物理链路均为双向传输，且上下行带宽可不对称。对于任意物理节点 $p \in P$，其可用计算资源包括 CPU 核心数和内存容量，分别记为 C_p^{CPU} 和 R_p^{RAM}。在系统运行过程中，部分资源已被分配给当前正在执行的作业任务，因此节点的剩余空闲资源随时间动态变化，分别用 $A_p^{\text{CPU}}(t)$ 和 $A_p^{\text{RAM}}(t)$ 表示时刻 t 的剩余 CPU 核数和内存大小。类似地，对于任意物理链路 $r \in R$，其总带宽记为 BW_r^{total}，已分配带宽为 $U_r(t)$，则时刻 t 的剩余带宽为 $A_r^{\text{BW}}(t) = BW_r^{\text{total}} - U_r(t)$。

5.1.1.2 作业模型

每个作业 $j \in J$ 具有一系列属性，用于刻画其资源需求和执行特性。作业 j 包含一个任务节点集合 $N_j = \{n_{j1}, \ldots, n_{jK_j}\}$，其中 $K_j = |N_j|$ 表示该作业包含的任务数量。任务节点之间通过通信链路相连，构成作业内部的通信拓扑，记链路集合为 $V_j = \{v_{j1}, \ldots, v_{jE_j}\}$，$E_j = |V_j|$ 为链路数量，每条链路连接两个不同的任务节点。对于

每个任务节点 $n \in N_j$，其资源需求由 CPU 核数 $\text{CPU}^N(n)$ 和内存大小 $\text{RAM}^N(n)$ 描述。在实际部署时，每个任务必须独占式地分配满足其需求的完整资源量。对于每条通信链路 $v \in V_j$，其带宽需求为一个区间 $[b_v^{\min}, b_v^{\max}]$，调度器必须至少分配 $b_v^{\min}$ 的带宽以保证基本通信性能，同时应尽可能接近 $b_v^{\max}$ 以提升作业执行效率。此外，每个作业 j 具有优先级权重 $w_j \in [1,5]$（整数），数值越大表示业务紧急程度越高。作业的到达时间记为 t_j^{arr}，实际开始执行时间为 t_j^{start}，由此可定义其等待时间 $T_j^{\text{wait}} = t_j^{\text{start}} - t_j^{\text{arr}}$。若作业尚未开始执行，则当前等待时间表示为 $t - t_j^{\text{arr}}$，其中 t 为当前时刻。这些属性共同构成了作业调度的输入信息，后续的调度决策需在满足物理资源约束的前提下，综合考虑作业优先级、等待时间及带宽分配的优化目标。

5.1.1.3 映射与资源分配约束

设二元决策变量 $X_p^n(j) \in \{0,1\}$ 表示作业 j 的任务 n 是否放置在物理节点 p；$Y_r^v(j) \in \{0,1\}$ 表示作业 j 的虚拟链路 v 是否占用物理链路 r。则约束如下：

1. **节点资源约束**（任一时刻 t）：

$$\forall p \in P: \sum_{j \in Q_{\text{run}}(t)} \sum_{n \in N_j} X_p^n(j) \cdot \text{CPU}^N(n) \le C_p^{\text{CPU}}, \tag{5-1}$$

$$\forall p \in P: \sum_{j \in Q_{\text{run}}(t)} \sum_{n \in N_j} X_p^n(j) \cdot \text{RAM}^N(n) \le R_p^{\text{RAM}}. \tag{5-2}$$

2. **链路带宽约束**（任一时刻 t）：

$$\forall r \in R: \sum_{j \in Q_{\text{run}}(t)} \sum_{v \in V_j} Y_r^v(j) \cdot B_v^{\text{alloc}} \le BW_r^{\text{total}}, \tag{5-3}$$

其中 $B_v^{\text{alloc}} \in [BW_v^{\min}, BW_v^{\max}]$ 为实际分配给链路 v 的带宽。

3. **任务映射唯一性**：

$$\forall j, \forall n \in N_j: \sum_{p \in P} X_p^n(j) = 1. \tag{5-4}$$

4. **链路路径约束**：若任务 n_i 映射至 p_a，任务 n_k 映射至 p_b，且虚拟链路 $v = (n_i, n_k)$，则 $Y_r^v(j) = 1$ 当且仅当物理链路 r 位于从 p_a 到 p_b 的最短路径上。本文假设采用静态最短路径路由。

5. **作业非抢占**：作业一旦开始执行，在完成前不中断，不释放资源。

5.1.2 性能指标

为量化调度质量，定义三个核心指标，分别为带宽带满足度、负载均衡与作业平均等待时间。

5.1.2.1 带宽需求满足度

对于运行中的作业 j，定义其带宽满足度为所有虚拟链路实际分配带宽与最大需求比值的平均值：

$$D_{\mathrm{BW}}(j) = \frac{1}{|\mathcal{V}_j|} \sum_{v \in \mathcal{V}_j} \frac{B_v^{\mathrm{alloc}}}{BW_v^{\max}}. \tag{5-5}$$

若作业所有任务映射至同一节点（无通信），则定义 $D_{\mathrm{BW}}(j) = 1$。系统整体带宽满足度为运行队列中作业的平均值：

$$\bar{D}_{\mathrm{BW}}(t) = \frac{1}{|Q_{\mathrm{run}}(t)|} \sum_{j \in Q_{\mathrm{run}}(t)} D_{\mathrm{BW}}(j). \tag{5-6}$$

5.1.2.2 系统负载均衡度

负载均衡度是衡量系统资源利用均匀性的重要指标，本文分别从节点和链路两个维度，考察 CPU、内存和带宽三类资源的利用分布情况。首先，定义物理节点 p 在时刻 t 的 CPU 利用率为其已占用资源与总资源的比值：

$$\rho_p^{\mathrm{CPU}}(t) = 1 - \frac{A_p^{\mathrm{CPU}}(t)}{C_p^{\mathrm{CPU}}}, \tag{5-7}$$

其中 $A_p^{\mathrm{CPU}}(t)$ 为剩余 CPU 核数，C_p^{CPU} 为总 CPU 核数。

类似地，内存利用率定义为：

$$\rho_p^{\mathrm{RAM}}(t) = 1 - \frac{A_p^{\mathrm{RAM}}(t)}{R_p^{\mathrm{RAM}}}. \tag{5-8}$$

对于物理链路 r，其带宽利用率则为已分配带宽占总带宽的比例：

$$\rho_r^{\mathrm{BW}}(t) = \frac{U_r(t)}{BW_r^{\mathrm{total}}}, \tag{5-9}$$

其中 $U_r(t)$ 为时刻 t 已分配的带宽，BW_r^{total} 为链路总带宽。

为了量化每类资源在不同实体（节点或链路）上利用率的离散程度，引入不均衡系数，采用变异系数（标准差与均值的比值）进行度量。设某类资源 $x \in \{\mathrm{CPU}, \mathrm{RAM}, \mathrm{BW}\}$ 对应的实体数量为 M_x（对于 CPU 和内存，$M_x = |P|$；对于带宽，$M_x = |R|$），利用率

的均值为：

$$\bar{\rho}^x = \frac{1}{M_x} \sum_{i=1}^{M_x} \rho_i^x. \tag{5-10}$$

则资源 x 的不均衡系数定义为：

$$L_t^x = \frac{\sqrt{\frac{1}{M_x} \sum_{i=1}^{M_x} (\rho_i^x - \bar{\rho}^x)^2}}{\bar{\rho}^x + \epsilon}, \tag{5-11}$$

其中分子为标准差，分母添加一个极小正数 ϵ（如 10^{-6}）以避免除零情形。该系数反映了该类资源利用率的波动程度：系数越大，表示不同实体间的利用率差异越明显，即负载分布越不均衡。

为综合评估整体系统的负载均衡程度，对三类资源的不均衡系数进行加权组合，得到整体负载均衡度：

$$L_t = w_1 L_t^{\text{CPU}} + w_2 L_t^{\text{RAM}} + w_3 L_t^{\text{BW}}, \quad w_1 + w_2 + w_3 = 1. \tag{5-12}$$

权重反映了不同资源在均衡性评价中的重要性，可根据实际场景调整。本文默认取 $w_1 = 0.4$，$w_2 = 0.4$，$w_3 = 0.2$，即优先考虑计算资源的均衡性，同时兼顾网络带宽的分布。L_t 的值越小，表明系统整体资源利用越均匀，负载均衡性越好。该指标将在后续调度决策中作为优化目标之一，引导调度器合理分配任务和通信资源，避免局部热点。

5.1.2.3 作业平均等待时间

在时刻 t，所有**已到达**作业的平均等待时间：

$$\bar{T}_{\text{wait}}(t) = \frac{1}{|Q_{\text{wait}}(t) \cup Q_{\text{run}}(t)|} \sum_{j \in Q_{\text{wait}} \cup Q_{\text{run}}} (t - t_j^{\text{arr}}). \tag{5-13}$$

对于已完成作业，其等待时间固定为 $t_j^{\text{start}} - t_j^{\text{arr}}$。

5.1.3 优化目标

本文采用加权和法将多目标问题转化为单目标优化。在每个调度时刻 t，选择待调度作业子集 $S_t \subseteq Q_{\text{wait}}(t)$，使得调度后的系统综合性能最大：

$$\max_{S_t} \ \alpha_1 \bar{D}_{\text{BW}}(t + |S_t|) + \alpha_2 \left(1 - L_{t+|S_t|}\right) + \alpha_3 \left(1 - \frac{\bar{T}_{\text{wait}}(t + |S_t|)}{T_{\max}}\right), \tag{5-14}$$

其中 $\alpha_1 = 0.5$，$\alpha_2 = 0.3$，$\alpha_3 = 0.2$，$T_{\max}$ 为最大容忍等待时间（设为仿真长度或经验值）。

由于未来状态依赖于当前调度决策且作业到达随机，精确在线优化极为困难。因此，本文设计一种启发式算法，在每个时刻仅调度至多一个作业，以贪心方式逼近上述目标。

5.2 算法设计

5.2.1 总体框架

本文提出的多作业在线调度算法 CES-Multi-Job 由四个核心模块构成：

1. **滚动窗口选择**：根据当前系统负载，从 Q_{wait} 中选出有限个作业作为候选集 C，避免每步遍历整个队列。
2. **综合评分计算**：对每个候选作业，计算资源适配度、优先级得分与等待因子，加权得到总评分。
3. **贪心调度尝试**：按评分降序依次调用单作业调度器尝试分配资源，成功则更新系统状态并移动作业。
4. **饥饿保护**：监控长期未获调度的作业，通过提升优先级或强制预留方式提高其调度机会。

算法5-2给出了 CES-Multi-Job 的伪代码。

5.2.2 滚动窗口机制

窗口大小 W_t 动态计算，其基本思想是：当系统剩余资源较多时，应扩大候选窗口，增加调度机会；当系统资源紧张时，缩小窗口，避免频繁失败尝试。具体公式：

$$W_t = \min\left(\left\lfloor \beta \cdot \frac{\sum_{p\in P} A_p^{\text{CPU}}(t)}{\max_{p\in P} C_p^{\text{CPU}}} \cdot |P| \right\rfloor,\ |Q_{\text{wait}}(t)|\right), \tag{5-15}$$

其中 $\beta \in [1,3]$ 为窗口扩展因子，默认 $\beta = 2$。分母使用最大节点 CPU 容量以归一化。若全部节点 CPU 空闲率均为 100%，则 $W_t = \beta|P|$；若系统满载，则 $W_t = 0$，不调度新作业。窗口内作业按**先入先出**顺序选取，即最早到达的 W_t 个作业。这种机制能自适应调整调度范围，兼顾调度效率与成功率，避免在高负载时过多尝试导致资源碎片化。

5.2.3 资源适配度计算

资源适配度 $R_{\text{match}}(j)$ 衡量作业 j 的总资源需求与当前网络剩余资源的匹配程度。本文采用最大空闲资源归一化方法：分别找出 CPU、内存、带宽在各自实体中的最大空闲

Algorithm 5-2 多作业在线调度算法 CES-Multi-Job

Require: 等待队列 $\mathcal{Q}_{\text{wait}}$，运行队列 $\mathcal{Q}_{\text{run}}$，物理网络状态 $(\mathcal{P}, \mathcal{R})$，参数 $\beta, \tau, \theta, K, T_{\text{th}}$

for 每个调度周期 **do**

  $W \leftarrow \min\left(\left\lfloor \beta \cdot \dfrac{\sum_p A_p^{\text{CPU}}}{\max_p C_p^{\text{CPU}}} \cdot |\mathcal{P}| \right\rfloor, |\mathcal{Q}_{\text{wait}}|\right)$ ▷ 步骤 1：滚动窗口选择

  $C \leftarrow \{j_1, j_2, \ldots, j_W\}$，即 $\mathcal{Q}_{\text{wait}}$ 的前 W 个作业（按到达时间排序）

  for each $j \in C$ **do** ▷ 步骤 2：为每个候选作业计算综合评分

    $R_{\text{match}}(j) \leftarrow \text{ComputeResourceMatch}(j)$ ▷ 式 (5-16)

    $S_{\text{wait}}(j) \leftarrow 1 - \exp\left(-\dfrac{t - t_j^{\text{arr}}}{\tau}\right)$

    $S_{\text{total}}(j) \leftarrow 0.6 \cdot R_{\text{match}}(j) + 0.3 \cdot \dfrac{w_j}{5} + 0.1 \cdot S_{\text{wait}}(j)$

  end for

  将 C 按 S_{total} 降序排列 ▷ 步骤 3：按评分降序尝试调度

  for each j in sorted C **do**

    $success \leftarrow \text{TryMapJob}(j)$ ▷ 调用单作业调度器

    if $success$ **then**

      将 j 从 $\mathcal{Q}_{\text{wait}}$ 移至 $\mathcal{Q}_{\text{run}}$，更新资源状态

      重置 j 的跳过计数 $\text{skip}(j) \leftarrow 0$

      break

    else

      $\text{skip}(j) \leftarrow \text{skip}(j) + 1$

    end if

  end for

  for each $j \in \mathcal{Q}_{\text{wait}}$ **do** ▷ 步骤 4：饥饿保护处理

    if $t - t_j^{\text{arr}} > T_{\text{th}}$ **then**

      临时提升 $w_j \leftarrow \min(w_j + 1, 5)$ ▷ 优先级提升

    end if

    if $\text{skip}(j) \geq K$ **then**

      $\text{TryForceMap}(j)$ ▷ 尝试预留最小资源包

    end if

  end for

end for

Ensure: 更新后的 $\mathcal{Q}_{\text{wait}}$，$\mathcal{Q}_{\text{run}}$ 及资源状态

量，然后计算需求与该最大值的比值，并截断至 1。数学表达式为：

$$R_{\text{match}}(j) = \frac{1}{3}\left[\min\left(1, \frac{\text{CPU}^{\text{dem}}(j)}{\max_{p\in P} A_p^{\text{CPU}}}\right) + \min\left(1, \frac{\text{RAM}^{\text{dem}}(j)}{\max_{p\in P} A_p^{\text{RAM}}}\right) + \min\left(1, \frac{\text{BW}^{\text{dem}}(j)}{\max_{r\in R} A_r^{\text{BW}}}\right)\right], \tag{5-16}$$

其中：

- $\text{CPU}^{\text{dem}}(j) = \sum_{n\in N_j} \text{CPU}^N(n)$，作业总 CPU 需求；
- $\text{RAM}^{\text{dem}}(j) = \sum_{n\in N_j} \text{RAM}^N(n)$，作业总内存需求；
- $\text{BW}^{\text{dem}}(j) = \sum_{v\in V_j} BW_v^{\min}$（此处采用 $BW_v^{\min}$ 作为保守估计）。

该归一化方式计算简单，且能反映作业需求是否超过当前网络中任何单节点和链路的剩余能力。若所有空闲资源均大于需求，则 $R_{\text{match}} = 1$；若需求超过最大空闲资源，则比值小于 1，鼓励调度需求较小的作业。

5.2.4 综合评分

每个候选作业的综合评分 $S_{\text{total}}(j)$ 由三部分加权组成：

- **资源适配度** $R_{\text{match}}(j)$，权重 0.6 ——反映即时资源匹配程度，是调度决策的主要依据。
- **优先级得分** $\frac{w_j}{5}$，权重 0.3 ——归一化至 $[0.2, 1]$，体现业务紧急程度。
- **等待时间因子** $S_{\text{wait}}(j)$，权重 0.1 ——采用指数函数 $S_{\text{wait}}(j) = 1 - \exp\left(-\frac{t - t_j^{\text{arr}}}{\tau}\right)$，其中 τ 为等待敏感参数（默认 $\tau = 50$）。等待时间越长，该因子越接近 1，防止作业饥饿。

因此：

$$S_{\text{total}}(j) = 0.6 \cdot R_{\text{match}}(j) + 0.3 \cdot \frac{w_j}{5} + 0.1 \cdot S_{\text{wait}}(j). \tag{5-17}$$

5.2.5 饥饿保护策略

为保障长期公平性，CES-Multi-Job 引入双重饥饿防护：

1. **等待时间阈值提升**：若作业等待时间超过 T_{th}，则将其优先级临时提升 1 级（不超过最大值 5），使其在后续评分中获得更高优先级权重。
2. **强制预留调度**：若作业连续被跳过 K 次，则触发强制调度尝试：系统为该作业预留一份“最小资源包”——例如，为每个任务预留最小需求（CPU/RAM），并为通信链路预留 $BW_v^{\min}$ 带宽。若预留成功，则立即调度该作业；否则保留其跳过计数，待后续资源充足时再尝试。

这两种保护机制协同工作：前者提高评分排名，后者通过资源预留保证极端情况下的调度机会。

5.2.6 算法复杂度分析

本节分析 CES-Multi-Job 调度算法的时空复杂度。设等待队列长度为 J，物理网络包含 P 个节点和 R 条链路。每个调度周期内，滚动窗口大小 W 满足 $W \leq J$，算法主要包含以下步骤：

- 计算 W 个作业的综合评分，耗时 $O(W)$；
- 对 W 个作业按评分排序，耗时 $O(W \log W)$；
- 依次尝试映射至多 W 个作业，每次映射调用单作业调度器，其复杂度为 $O(\Phi_{\max})$（$\Phi_{\max}$ 为单个作业调度开销的上界，与作业规模和神经网络结构有关，详见第四章），故映射阶段耗时 $O(W\Phi_{\max})$；
- 遍历整个等待队列进行饥饿保护，耗时 $O(J)$。

因此，每个调度周期的总时间复杂度为 $O(W + W\log W + W\Phi_{\max} + J)$。由于 $W \leq J$ 且 $\Phi_{\max}$ 在实际中为与作业规模相关的小常数，上式可简化为 $O(J\log J + J\Phi_{\max})$。在典型部署场景中，$J$ 通常为百数量级，且神经网络前向传播可在毫秒级完成，故算法能够满足在线调度的实时性要求。

空间复杂度主要包括：存储物理网络状态（$O(P+R)$）、维护等待队列与运行队列的作业信息（$O(J \cdot (\max K_j + \max E_j))$），以及单作业调度器所用神经网络模型参数（$O(d_{\text{state}} \times d_{\text{hidden}} + d_{\text{hidden}}^2)$）。整体空间开销与网络规模、作业数量和作业规模呈线性关系，在边缘计算平台上可轻松支撑。

5.3 实验设计

为验证 CES-Multi-Job 算法的有效性，本节设计了仿真实验框架，涵盖物理网络生成、作业负载生成、对比算法、评估指标及统计分析方法。

5.3.1 仿真环境设置

5.3.1.1 物理网络拓扑

仿真实验采用的物理网络包含 10 个节点，构成一个随机连通图。节点之间的链路生成方式如下：首先，以概率 $p_{\text{edge}} = 0.3$ 独立决定每一对节点之间是否存在链路，从而得到一个随机图；随后，检查该图的连通性，若存在多个连通分量，则随机添加边直至

整个图变为连通图，以确保任意两个物理节点之间均存在可达路径。每个物理节点的资源容量（CPU 核心数和内存大小）均在区间 $[50, 100]$ 内按均匀分布随机取值，其中 CPU 单位为核心数，内存单位为吉字节（GB）。每条物理链路的带宽容量则在 $[100, 1000]$ Mbps 范围内均匀随机采样，以模拟异构的网络环境。

5.3.1.2 作业到达过程

作业的到达过程服从泊松分布，其到达率 λ 取 3（单位：作业/时间单位）。每个作业 j 包含的任务节点数 K_j 从集合 $\{3, 4, 5, 6, 7, 8\}$ 中均匀随机选取。在任务节点之间，以概率 $p_{\text{comm}} = 0.4$ 独立添加虚拟通信链路，并确保生成的通信子图是连通的，从而构成作业的内部拓扑结构。若生成的图不连通，则随机添加边直至连通。每个任务节点 n 的 CPU 需求 $\text{CPU}^N(n)$ 和内存需求 $\text{RAM}^N(n)$ 均独立地在区间 $[10, 50]$ 内按均匀分布随机生成。对于每条虚拟链路 v，其最小带宽需求 $b_v^{\min}$ 在 $[10, 50]$ Mbps 内均匀随机采样，最大带宽需求则在此基础上增加一个 $[0, 50]$ Mbps 的随机增量，即 $b_v^{\max} = b_v^{\min} + \text{Uniform}[0, 50]$，以保证 $b_v^{\max} \geq b_v^{\min}$。作业的优先级权重 w_j 为整数，从 $\{1, 2, 3, 4, 5\}$ 中均匀随机选取，数值越大表示业务紧急程度越高。仿真总时长设定为时间单位，在此过程中动态生成作业，作业总数约为 $\lambda \cdot T_{\text{sim}}$，大致在 1200 个，具体数量服从泊松过程的随机性。仿真总时长设为 $T_{\text{sim}} = 400$ 个时间单位。作业到达过程服从参数 $\lambda = 3$ 的泊松分布，因此作业总数是一个期望值为 $\lambda \cdot T_{\text{sim}} = 3 \cdot 400 = 1200$ 的泊松随机变量，实际生成的数量将围绕该期望值随机波动。

5.3.2 对比算法

为评估本文提出的 CES-Multi-Job 调度算法的性能，我们选取四种具有代表性的调度策略作为对比基线，涵盖基于优先级的简单策略、经典作业调度策略以及源自现有研究的启发式方法。各对比算法的核心思想及其在本文仿真环境中的简化实现方式如下：

- **SWTS**：每个作业的综合得分由以下四个因素加权组合：关键路径长度（CPL，与作业预计运行时间成反比）、任务并行度（TPS，与任务节点数 K_j 正相关）、数据局部性评分（DLS，与总带宽需求 $\sum b_v^{\min}$ 成反比）以及资源匹配度（计算节点资源与任务需求的匹配程度）。此外，作业的优先级 w_j 也被纳入评分体系。每次调度时，选择综合得分最高的作业进行映射。
- **AdaEvo**：首先，结合作业的优先级和等待时间计算紧急度，然后根据紧急度将等待队列中的作业划分为 K 组（等数量分组），并从紧急度最高的组内选取作业。

组内评分综合考虑资源匹配度、CPL、TPS 和优先级，并以该组的紧急度加权，最终选择得分最高的作业进行调度。

- **FIFO（先入先出）**：最基础的调度策略，严格按照作业到达顺序进行调度。每次调度周期中，直接选择等待队列中等待时间最长的作业尝试映射，不涉及任何评分或资源匹配计算。
- **SJF（短作业优先）**：经典的作业调度算法，优先选择执行时间最短的作业以降低平均等待时间。在本文实现中，作业的执行时间由任务节点数 K_j 近似表征，即每次遍历等待队列，选择 K_j 最小的作业进行调度。该算法能够反映短作业对系统吞吐量的影响。

5.3.3 实验结果分析

本节通过仿真实验对比 CES-Multi-Job 算法与四种基线算法（SWTS、AdaEvo、FIFO、SJF）的性能表现。实验围绕三个核心指标展开：作业平均等待时间、带宽需求满足度以及系统负载均衡度。所有算法均采用相同的单作业调度器（CES-PPO）以确保公平性，并在相同仿真环境下独立运行 10 次，取平均值作为最终结果。

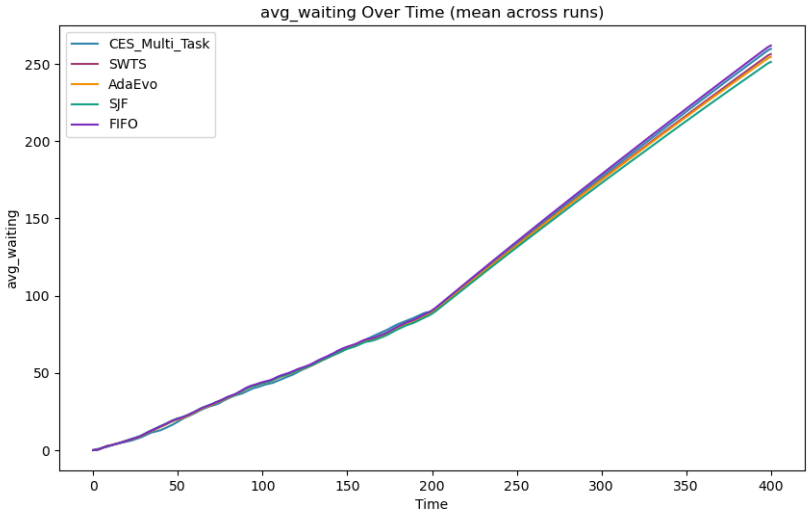

图 5-1 等待时间统计图

等待时间分析：图5-1展示了各算法下作业平均等待时间随仿真时间的变化趋势。观察可知，五种算法的等待时间曲线高度重合，均随仿真进程推进而逐渐增长，且在仿真结束时数值十分接近。这一现象表明，在本文设定的作业到达率（λ=3）和资源规模下，作业等待时间主要由系统负载和单作业调度器的执行效率主导，而非多作业选择策略。由于所有算法使用相同的底层调度器，作业一旦被选中，其执行时间分布基本一致，

因此队列积压程度相近，导致等待时间无明显差异。

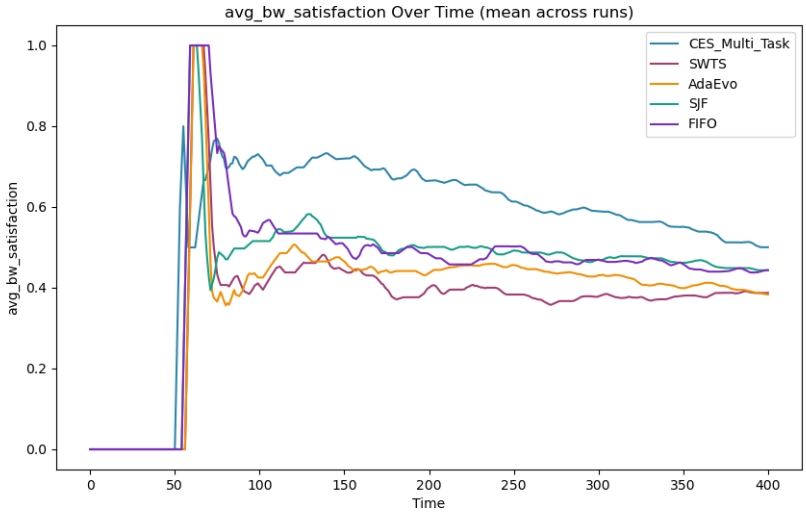

图 5-2 带宽满足度统计图

带宽满足度分析：带宽满足度定义为作业实际分配带宽与需求带宽的比值，反映调度器对通信资源的保障能力。如图5-2所示，CES-Multi-Job 算法的带宽满足度显著高于其他算法，而 SJF 和 FIFO 次之，SWTS 和 AdaEvo 相对较低。这一结果源于 CES-Multi-Job 在作业选择阶段显式考虑了带宽需求匹配：其综合评分中包含资源匹配度分量，倾向于选择通信需求与当前链路剩余带宽更匹配的作业。相比之下，SWTS 虽引入数据局部性评分，但该指标仅与总带宽需求成反比，未与实时可用带宽联动；AdaEvo 则侧重于紧急程度分组，带宽因素未占主导；FIFO 和 SJF 完全忽略带宽，仅靠映射器在单作业层面尽力满足，故带宽满足度居中。

负载均衡度分析：负载均衡度采用三类资源（CPU、内存、带宽）利用率的变异系数加权和（L_t）度量，值越小表示资源分布越均匀。实验结果如图5-3显示，SWTS 和 AdaEvo 的负载均衡度最佳（L_t 值最低），CES-Multi-Job 与 SJF、FIFO 水平相当，略逊于前两者。SWTS 得益于其节点感知策略和动态调整机制，在任务分配时显式考虑节点负载，从而有效避免热点；AdaEvo 通过紧急程度分组和组内资源匹配，间接促进了资源均衡。CES-Multi-Job 虽在单作业映射层采用了负载最轻节点优先的贪心策略，但在多作业选择阶段以带宽满足度为主要优化目标，可能优先调度带宽需求高的作业，导致部分节点或链路负载集中，因此均衡性未达最优。SJF 和 FIFO 由于选择策略简单，依赖底层映射器的负载均衡能力，表现与 CES 相近。

综合性能评估：为综合比较各算法的整体表现，构造加权性能指标 $P = \alpha \cdot B_{\text{sat}} -$

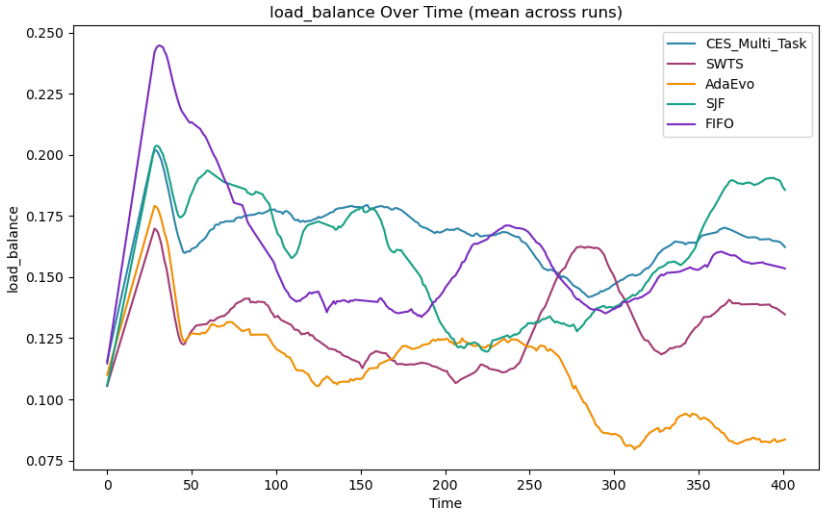

图 5-3 负载均衡度统计图

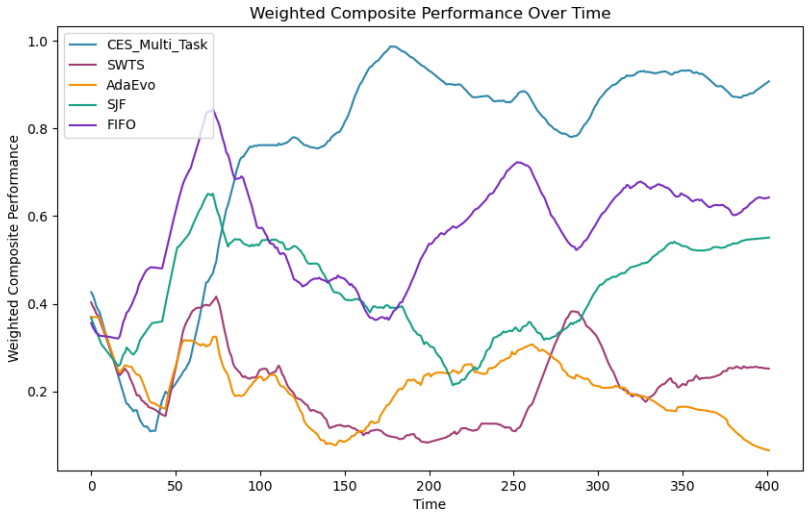

图 5-4 加权综合性能统计图

$\beta \cdot L_t + \gamma \cdot (1 - T_{\text{wait}}/T_{\max})$，其中 B_{sat} 为带宽满足度，L_t 为负载均衡度，T_{wait} 为平均等待时间，权重根据系统设计目标设定为 $\alpha = 0.5, \beta = 0.3, \gamma = 0.2$。如图5-4结果表明，CES-Multi-Job 的综合得分最高，SJF 和 FIFO 次之，SWTS 和 AdaEvo 相对较低。这说明尽管 CES-Multi-Job 在负载均衡方面略有不足，但其在带宽满足度上的显著优势足以弥补这一短板，并在整体上取得最佳调度效果。SJF 和 FIFO 因实现简单且依赖底层映射器，在等待时间和带宽满足度上表现中等，综合性能尚可。SWTS 和 AdaEvo 虽负载均衡出色，但带宽满足度偏低，拉低了综合评分。

综上，CES-Multi-Job 算法在保障作业通信需求方面具有明显优势，同时维持可接

受的等待时间和负载均衡水平，整体调度性能优于对比算法。未来工作可进一步引入动态负载均衡机制，在带宽优化与资源均匀分布之间寻求更优平衡。

5.4 本章小结

本文面向协同边缘计算环境下的多分布式学习作业调度问题，提出了一种在线启发式调度算法 CES-Multi-Job。该算法通过动态滚动窗口控制决策规模，利用资源感知评分引导作业调度顺序，并引入等待时间阈值与强制预留机制防止饥饿，综合优化了带宽满足度、负载均衡度与平均等待时间三个相互冲突的指标。与现有分阶段调度方法相比，CES-Multi-Job 将计算与网络资源联合考虑，且与任意单作业调度器兼容，具有良好的可扩展性与实用性。

总结与展望

文章完成了问题形式化建模、算法详细设计及完整的仿真实验方案。后续工作将探索以下方向：（1）将窗口自适应因子 β 与等待敏感参数 τ 动态在线调优；（2）引入多作业并行调度能力，进一步提升资源利用率；（3）将 CES-Multi-Job 扩展至异构资源（如 GPU、NPU）场景。

参考文献

[1] Chiang M, Zhang T. Fog and IoT: An Overview of Research Opportunities[J]. IEEE Internet of Things Journal, 2016, 3(6): 854-864. DOI: 10.1109/JIOT.2016.2584538.

[2] Raeisi-Varzaneh M, Dakkak O, Habbal A, et al. Resource Scheduling in Edge Computing: Architecture, Taxonomy, Open Issues and Future Research Directions[J]. IEEE Access, 2023, 11: 25329-25350. DOI: 10.1109/ACCESS.2023.3256522.

[3] Zhang M, Cao J, Sahni Y, et al. EaaS: A Service-Oriented Edge Computing Framework Towards Distributed Intelligence[C]//2022 IEEE International Conference on Service-Oriented System Engineering (SOSE). 2022: 165-175. DOI: 10.1109/SOSE55356.2022.00026.

[4] Luo Q, Hu S, Li C, et al. Resource Scheduling in Edge Computing: A Survey[J]. IEEE Communications Surveys & Tutorials, 2021, 23(4): 2131-2165. DOI: 10.1109/COMST.2021.3106401.

[5] Lin L, Liao X, Jin H, et al. Computation Offloading Toward Edge Computing[J]. Proceedings of the IEEE, 2019, 107(8): 1584-1607. DOI: 10.1109/JPROC.2019.2922285.

[6] Konečný J, McMahan H B, Yu F X, et al. Federated Learning: Strategies for Improving Communication Efficiency[EB/OL]. 2017. https://arxiv.org/abs/1610.05492. arXiv: 1610.05492 [cs.LG].

[7] Zhang M, Cao J, Yang L, et al. ENTS: An Edge-native Task Scheduling System for Collaborative Edge Computing[C]//2022 IEEE/ACM 7th Symposium on Edge Computing (SEC). 2022: 149-161. DOI: 10.1109/SEC54971.2022.00019.

[8] Kairouz P, McMahan H B, Avent B, et al. Advances and Open Problems in Federated Learning[EB/OL]. 2021. https://arxiv.org/abs/1912.04977. arXiv: 1912.04977 [cs.LG].

[9] Kurniawan A. Learning AWS IoT: Effectively manage connected devices on the AWS cloud using services such as AWS Greengrass, AWS button, predictive analytics and machine learning[M/OL]. Packt Publishing, 2018. https://books.google.com.hk/books?id=7NRJDwAAQBAJ.

[10] Jensen D. Beginning Azure IoT Edge Computing: Extending the Cloud to the Intelligent Edge[M/OL]. Apress, 2019. https://books.google.com.hk/books?id=SeuVDwAAQBAJ.

[11] Xiong Y, Sun Y, Xing L, et al. Extend Cloud to Edge with KubeEdge[C]//2018 IEEE/ACM Symposium on Edge Computing (SEC). 2018: 373-377. DOI: 10.1109/SEC.2018.00048.

[12] Kjorveziroski V, et al. Kubernetes distributions for the edge: serverless performance evaluation[J/OL]. The Journal of Supercomputing, 2022, 78: 13728-13755. https://link.springer.com/article/10.1007/s11227-022-04430-6. DOI: 10.1007/s11227-022-04430-6.

[13] Li M, Ren Y, Guo Y, et al. KubeMSEN: Extend KubeEdge to Support Metadata Sharing among Edge Nodes[C]//2023 5th International Conference on Frontiers Technology of Information and Computer (ICFTIC). 2023: 45-50. DOI: 10.1109/ICFTIC59930.2023.10456315.

[14] Santos J, Wauters T, Volckaert B, et al. Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing Applications[C]//2019 IEEE Conference on Network Softwarization (NetSoft). 2019: 351-359. DOI: 10.1109/NETSOFT.2019.8806671.

[15] Rossi F, Cardellini V, Lo Presti F, et al. Geo-distributed efficient deployment of containers with Kubernetes[J/OL]. Computer Communications, 2020, 159: 161-174. https://www.sciencedirect.com/science/article/pii/S0140366419317931. DOI: https://doi.org/10.1016/j.comcom.2020.04.061.

[16] Wojciechowski Ł, Opasiak K, Latusek J, et al. NetMARKS: Network Metrics-AwaRe Kubernetes Scheduler Powered by Service Mesh[C]//IEEE INFOCOM 2021 - IEEE Conference on Computer Communications. 2021: 1-9. DOI: 10.1109/INFOCOM42981.2021.9488670.

[17] Sundar S, Liang B. Offloading Dependent Tasks with Communication Delay and Deadline Constraint[C]//IEEE INFOCOM 2018 - IEEE Conference on Computer Communications. 2018: 37-45. DOI: 10.1109/INFOCOM.2018.8486305.

[18] Wang J, Hu J, Min G, et al. Dependent Task Offloading for Edge Computing based on Deep Reinforcement Learning[J]. IEEE Transactions on Computers, 2022, 71(10): 2449-2461. DOI: 10.1109/TC.2021.3131040.

[19] Sahni Y, Cao J, Yang L. Data-Aware Task Allocation for Achieving Low Latency in Collaborative Edge Computing[J]. IEEE Internet of Things Journal, 2019, 6(2): 3512-3524. DOI: 10.1109/JIOT.2018.2886757.

[20] Shen S, Ren Y, Ju Y, et al. EdgeMatrix: A Resource-Redefined Scheduling Framework for SLA-Guaranteed Multi-Tier Edge-Cloud Computing Systems[J]. IEEE Journal on Selected Areas in Communications, 2023, 41(3): 820-834. DOI: 10.1109/JSAC.2022.3229444.

[21] Tang J, Jalalzai M M, Feng C, et al. Latency-Aware Task Scheduling in Software-Defined Edge and Cloud Computing With Erasure-Coded Storage Systems[J]. IEEE Transactions on Cloud Computing, 2023, 11(2): 1575-1590. DOI: 10.1109/TCC.2022.3149963.

[22] Liu T, Zeng F, Xia P. Dynamic Kubernetes Scheduling: Load and Resource Optimization in Collaborative Edge Computing[C]//2024 10th International Conference on Big Data Computing and Communications (BigCom). 2024: 9-17. DOI: 10.1109/BIGCOM65357.2024.00011.

[23] Cámara-Miró J, Costero L, Igual F D. QoS-aware workload scheduling on heterogeneous and dynamic edge-to-cloud deployments[C]//2025 33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP). 2025: 321-328. DOI: 10.1109/PDP66500.2025.00052.

[24] Chaalal E, et al. Functionality-aware offloading technique for scheduling containerized edge applications in IoT edge computing[J/OL]. Journal of Cloud Computing, 2025, 14(13). https://link.springer.com/article/10.1186/s13677-025-00737-w. DOI: 10.1186/s13677-025-00737-w.

[25] Wang Z, Goudarzi M, Buyya R. TF-DDRL: A Transformer-Enhanced Distributed DRL Technique for Scheduling IoT Applications in Edge and Cloud Computing Environments [J]. IEEE Transactions on Services Computing, 2025, 18(2): 1039-1053. DOI: 10.1109/TSC.2025.3528346.

[26] Jiang F, Peng Y, Wang K, et al. MARS: A DRL-Based Multi-Task Resource Scheduling Framework for UAV With IRS-Assisted Mobile Edge Computing System[J]. IEEE Transactions on Cloud Computing, 2023, 11(4): 3700-3712. DOI: 10.1109/TCC.2023.3307582.

[27] Li J, Zhang H, Li S, et al. BD-TTS: A blockchain and DRL-based framework for trusted task scheduling in edge computing[J/OL]. Computer Networks, 2024, 251: 110609. https://www.sciencedirect.com/science/article/pii/S1389128624004419. DOI: https://doi.or

g/10.1016/j.comnet.2024.110609.

[28] Lu J, Yang J, Li S, et al. A2C-DRL: Dynamic Scheduling for Stochastic Edge–Cloud Environments Using A2C and Deep Reinforcement Learning[J]. IEEE Internet of Things Journal, 2024, 11(9): 16915-16927. DOI: 10.1109/JIOT.2024.3366252.

[29] Zhang P, Luo Z, Kumar N, et al. CE-VNE: Constraint escalation virtual network embedding algorithm assisted by graph convolutional networks[J/OL]. Journal of Network and Computer Applications, 2024, 221: 103736. https://www.sciencedirect.com/science/article/pii/S1084804523001558. DOI: https://doi.org/10.1016/j.jnca.2023.103736.

[30] Duan Z, Wang T. Towards learning-based energy-efficient online coordinated virtual network embedding framework[J/OL]. Computer Networks, 2024, 239: 110139. https://www.sciencedirect.com/science/article/pii/S1389128623005844. DOI: https://doi.org/10.1016/j.comnet.2023.110139.

[31] Zhang P, Gan P, Kumar N, et al. RKD-VNE: Virtual network embedding algorithm assisted by resource knowledge description and deep reinforcement learning in IIoT scenario[J/OL]. Future Generation Computer Systems, 2022, 135: 426-437. https://www.sciencedirect.com/science/article/pii/S0167739X2200173X. DOI: https://doi.org/10.1016/j.future.2022.05.008.

[32] Zhang M, Shen X, Cao J, et al. EdgeShard: Efficient LLM Inference via Collaborative Edge Computing[J]. IEEE Internet of Things Journal, 2025, 12(10): 13119-13131. DOI: 10.1109/JIOT.2024.3524255.

[33] Dong P, Ge J, Wang X, et al. Collaborative Edge Computing for Social Internet of Things: Applications, Solutions, and Challenges[J]. IEEE Transactions on Computational Social Systems, 2022, 9(1): 291-301. DOI: 10.1109/TCSS.2021.3072693.

[34] Davy S, Famaey J, Serrat J, et al. Challenges to support edge-as-a-service[J]. IEEE Communications Magazine, 2014, 52(1): 132-139. DOI: 10.1109/MCOM.2014.6710075.

[35] Varghese B, Wang N, Li J, et al. Edge-as-a-Service: Towards Distributed Cloud Architectures[EB/OL]. 2017. https://arxiv.org/abs/1710.10090. arXiv: 1710.10090 [cs.DC].

[36] Sahni Y, Cao J, Yang L, et al. Multihop Offloading of Multiple DAG Tasks in Collaborative Edge Computing[J]. IEEE Internet of Things Journal, 2021, 8(6): 4893-4905. DOI: 10.1109/JIOT.2020.3030926.

[37] Liu X, Fang D, Xu P. Automated Performance Benchmarking Platform of IaaS Cloud[C]//2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). 2021: 1402-1405. DOI: 10.1109/TrustCom53373.2021.00197.

[38] Wen Z, Liang Y, Li G. Design and Implementation of High-availability PaaS Platform Based on Virtualization Platform[C]//2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC). 2020: 1571-1575. DOI: 10.1109/ITOEC49072.2020.9141564.

[39] Aleem S, Ahmed F, Batool R, et al. Empirical Investigation of Key Factors for SaaS Architecture[J]. IEEE Transactions on Cloud Computing, 2021, 9(3): 1037-1049. DOI: 10.1109/TCC.2019.2906299.

[40] Liang Y, Huang F, Mei Y, et al. Distributed and Efficient Request Scheduling in Collaborative Edge Computing[C]//2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS). 2024: 1458-1459. DOI: 10.1109/ICDCS60910.2024.00150.

[41] Mach P, Becvar Z. Mobile Edge Computing: A Survey on Architecture and Computation Offloading[J]. IEEE Communications Surveys & Tutorials, 2017, 19(3): 1628-1656. DOI: 10.1109/COMST.2017.2682318.

[42] Zhao M, Li W, Bao L, et al. Fairness-Aware Task Scheduling and Resource Allocation in UAV-Enabled Mobile Edge Computing Networks[J]. IEEE Transactions on Green Communications and Networking, 2021, 5(4): 2174-2187. DOI: 10.1109/TGCN.2021.3095070.

[43] Materwala H, Ismail L, Shubair R M, et al. Energy-SLA-aware genetic algorithm for edge–cloud integrated computation offloading in vehicular networks[J/OL]. Future Generation Computer Systems, 2022, 135: 205-222. https://www.sciencedirect.com/science/article/pii/S0167739X22001327. DOI: https://doi.org/10.1016/j.future.2022.04.009.

[44] Wang J, Zhang J, Liu L, et al. Utility Maximization for Splittable Task Offloading in IoT Edge Network[J/OL]. Computer Networks, 2022, 214: 109164. https://www.sciencedirect.com/science/article/pii/S1389128622002705. DOI: https://doi.org/10.1016/j.comnet.2022.109164.

[45] Xu X, Zhang X, Gao H, et al. BeCome: Blockchain-Enabled Computation Offloading for IoT in Mobile Edge Computing[J]. IEEE Transactions on Industrial Informatics, 2020, 16(6): 4187-4195. DOI: 10.1109/TII.2019.2936869.

[46] Yang L, Yao H, Wang J, et al. Multi-UAV-Enabled Load-Balance Mobile-Edge Computing for IoT Networks[J]. IEEE Internet of Things Journal, 2020, 7(8): 6898-6908. DOI: 10.1109/JIOT.2020.2971645.

[47] Xu F, Xie Y, Sun Y, et al. Two-stage computing offloading algorithm in cloud-edge collaborative scenarios based on game theory[J/OL]. Computers and Electrical Engineering, 2022, 97: 107624. https://www.sciencedirect.com/science/article/pii/S0045790621005541. DOI: https://doi.org/10.1016/j.compeleceng.2021.107624.

[48] Zhang J, Guo H, Liu J, et al. Task Offloading in Vehicular Edge Computing Networks: A Load-Balancing Solution[J]. IEEE Transactions on Vehicular Technology, 2020, 69(2): 2092-2104. DOI: 10.1109/TVT.2019.2959410.

[49] Tang M, Wong V W. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems[J]. IEEE Transactions on Mobile Computing, 2022, 21(6): 1985-1997. DOI: 10.1109/TMC.2020.3036871.

[50] Wang J, Zhao L, Liu J, et al. Smart Resource Allocation for Mobile Edge Computing: A Deep Reinforcement Learning Approach[J]. IEEE Transactions on Emerging Topics in Computing, 2021, 9(3): 1529-1541. DOI: 10.1109/TETC.2019.2902661.

[51] Chen Y, Zhang N, Zhang Y, et al. Energy Efficient Dynamic Offloading in Mobile Edge Computing for Internet of Things[J]. IEEE Transactions on Cloud Computing, 2021, 9(3): 1050-1060. DOI: 10.1109/TCC.2019.2898657.

[52] Wu H, Wolter K, Jiao P, et al. EEDTO: An Energy-Efficient Dynamic Task Offloading Algorithm for Blockchain-Enabled IoT-Edge-Cloud Orchestrated Computing[J]. IEEE Internet of Things Journal, 2021, 8(4): 2163-2176. DOI: 10.1109/JIOT.2020.3033521.

[53] Grislin-Le Strugeon E, Ouarnoughi H, Niar S. A Multi-Agent Approach for Vehicle-to-Fog Fair Computation Offloading[C]//2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA). 2020: 1-8. DOI: 10.1109/AICCSA50499.2020.9316512.

[54] Verbraeken J, Wolting M, Katzy J, et al. A Survey on Distributed Machine Learning

[J/OL]. ACM Computing Surveys, 2020, 53(2): 1-33. http://dx.doi.org/10.1145/3377454. DOI: 10.1145/3377454.

[55] Xing E P, Ho Q, Xie P, et al. Strategies and Principles of Distributed Machine Learning on Big Data[EB/OL]. 2015. https://arxiv.org/abs/1512.09295. arXiv: 1512.09295 [stat.ML].

[56] Schulman J, Wolski F, Dhariwal P, et al. Proximal Policy Optimization Algorithms [EB/OL]. 2017. https://arxiv.org/abs/1707.06347. arXiv: 1707.06347 [cs.LG].

[57] Rio A d, Jimenez D, Serrano J. Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments[J]. IEEE Access, 2024, 12: 146795-146806. DOI: 10.1109/ACCESS.2024.3472473.

攻读博士/硕士学位期间取得的研究成果

一、已发表（包括已接受待发表）的论文，以及已投稿、或已成文打算投稿、或拟成文投稿的论文情况**（只填写与学位论文内容相关的部分）：**

序号	作者（全体作者，按顺序排列）	题目	发表或投稿刊物名称、级别	发表的卷期、年月、页码	与学位论文哪一部分（章、节）相关	被索引收录情况
1						
2						

注：在“发表的卷期、年月、页码”栏：

1. 如果论文已发表，请填写发表的卷期、年月、页码；

2. 如果论文已被接受，填写将要发表的卷期、年月；

3. 以上都不是，请据实填写“已投稿”，“拟投稿”。

不够请另加页。

二、与学位内容相关的其它成果（包括专利、著作、获奖项目等）

致　谢

活着不容易啊，感谢所有帮助过我的人，感谢我的父母，感谢我的导师，感谢我的同学，感谢我的朋友们。谢谢你们！