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 SCES（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

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

目　录

插图目录

表格目录

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。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。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。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。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。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。

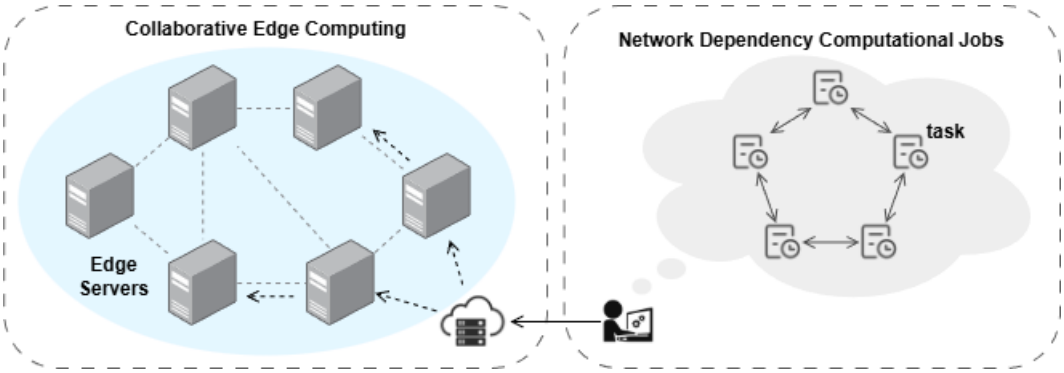

图 1-1 协同边缘任务调度场景图

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。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。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、Azure IoT Edge 和 KubeEdge 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）或扩展其能力到边缘（如 KubeEdge、OpenYurt）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 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 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 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 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 框架针对物联网应用的任务依赖建模，以及 MARS 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。现有研究即便考虑了任务依赖（如 TF-DDRL），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 中的物联网独立任务、A2C-DRL 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。以 CE-VNE、PPO-VNE 等为代表的先进算法，虽引入了图卷积网络（GCN）自动提取拓扑特征（CE-VNE, PPO-VNE），并设计了多目标奖励函数以同时优化资源收益与能耗（PPO-VNE），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 包含带宽资源，PPO-VNE 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 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，未能从根本上实现计算、带宽、安全等多维资源的深度协同。例如，其通用的两阶段式“节点-链路”映射动作（CE-VNE、Energy Allocation for V2G 等均采用节点选择后接 BFS/Dijkstra 路径搜索），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

2.2　分布式训练

2.3　任务调度

2.4　深度强化学习

2.5　本章小结

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（Master-Edge Node）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。

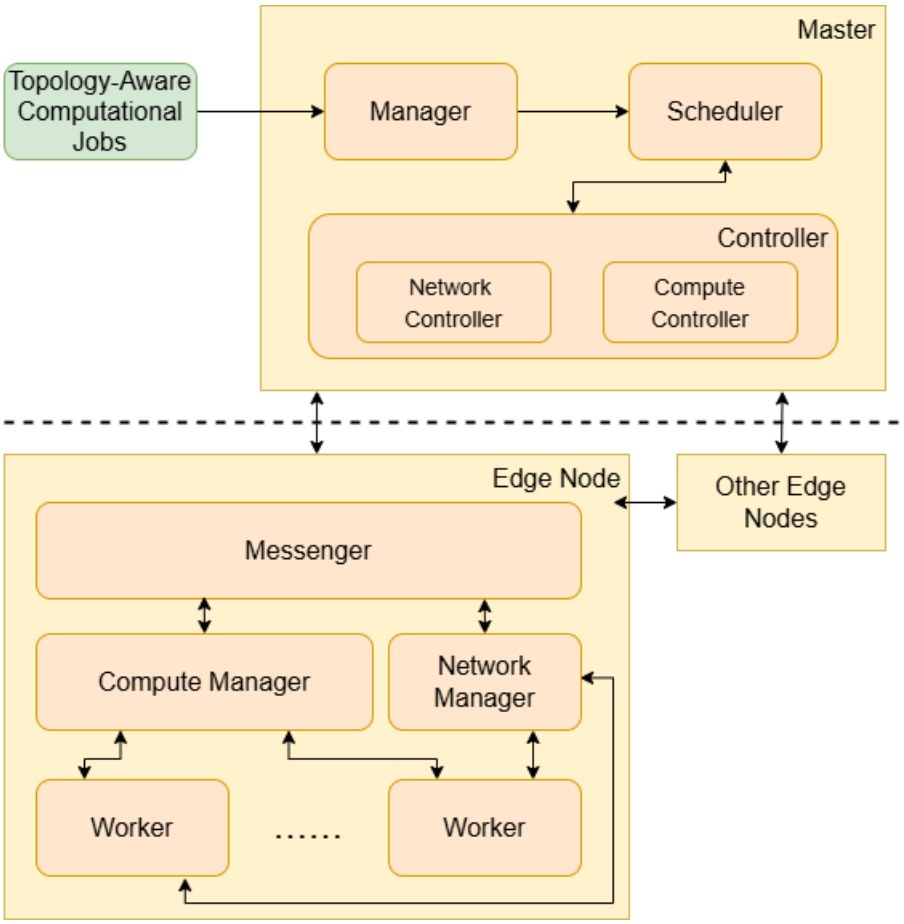

图 3-1 协同边缘任务调度系统 CES 架构图

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

第四章　基于深度强化学习的联合任务调度算法

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 $BW_{min}^V(v) \leq B^V(v) \leq BW_{max}^V(v)$ 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：

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

任务链路到物理链路的映射关系：

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

每个任务节点必须且仅能部署在一个物理节点上：

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

每个物理节点拥有的 CPU 资源、内存资源能够满足任务节点：

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

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

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：

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

每条物理链路拥有的带宽资源能够满足任务链路：

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

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：

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

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

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

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：

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

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。

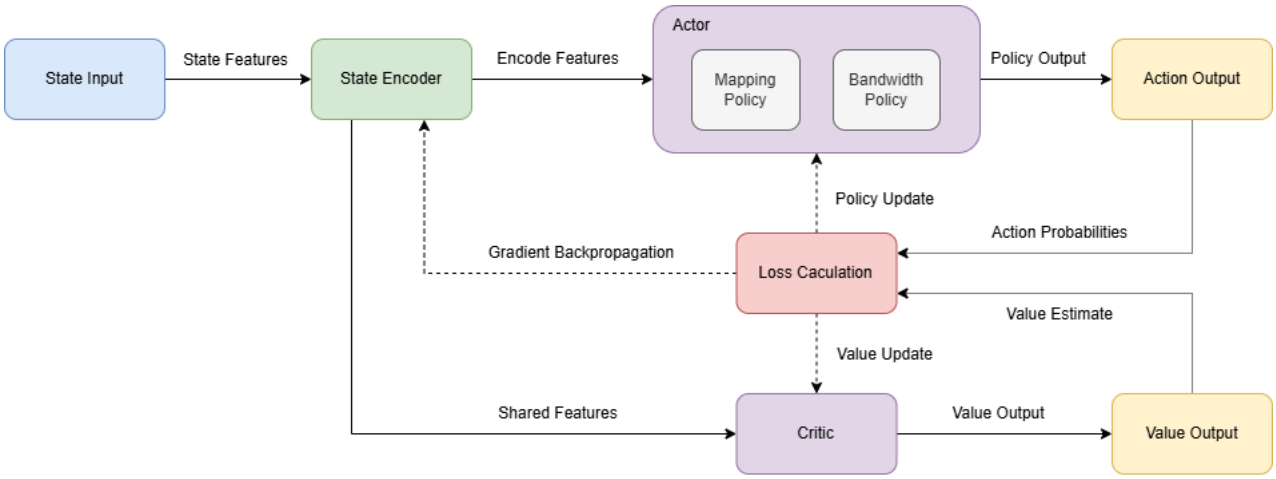

图 4-2 双分支策略网络架构

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 系统及算法进行验证。

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：基于强化学习的端到端调度框架。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：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。

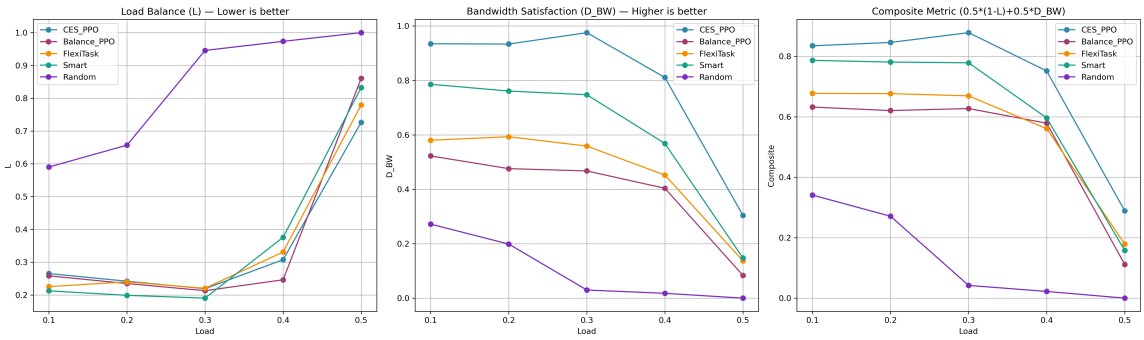

图 4-3 仿真实验结果

4.3.2.2 实验分析

仿真实验结果如图4-3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在综合性能上具有显著且稳定的优势。这直接得益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，一体化地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。具体而言：在带宽满足度方面，本文算法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，使其能够更精准地适配动态且复杂的带宽约束。在负载均衡度方面，本文算法与 PPO Balance、FlexiTask 及 Smart 等优化算法的平均水平相当，但略低于它们。这反映了算法在多目标优化中的权衡策略，即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的、可控的妥协。最终，从融合负载均衡度与带宽满足度的综合指标 $(0.5 \times (1 - L) + 0.5 \times D_{BW})$ 来看，本文算法在所有任务规模下均取得了最优值。这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

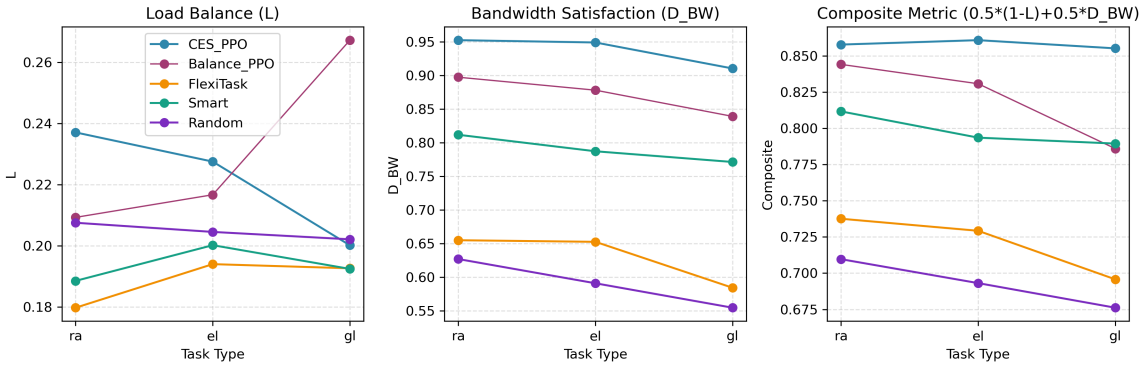

图 4-4 真实环境实验

真实环境下的实验结果如图4-4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

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4-5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

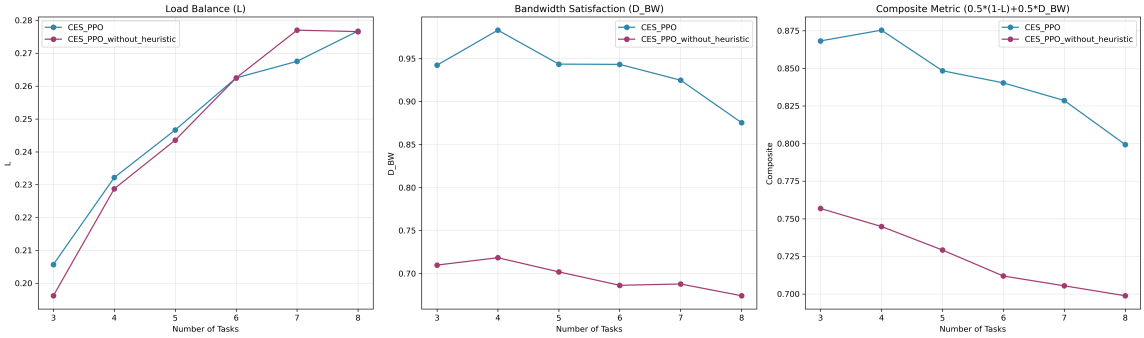

图 4-5 消融实验

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

第五章　多任务队列调度算法

5.1　问题建模

5.2　算法设计

5.3　实验结果与分析

总结与展望

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

这次你离开了没有像以前那样说再见, 再见也他妈的只是再见
我们之间从来没有想象的那么接近, 只是两棵树的距离
你是否还记得山阴路我八楼的房间, 房间里唱歌的日日夜夜
那么热的夏天你看着外面, 看着你在消逝的容颜
我多么想念你走在我身边的样子, 想起来我的爱就不能停止
南京的雨不停地下不停地下, 就像你沉默的委屈
一转眼, 我们的城市又到了夏天, 对面走来的人都眯着眼
人们不敢说话不敢停下脚步, 因为心动常常带来危险
我多么想念你走在我身边的样子, 想起来我的爱就不能停止
南京的雨不停地下不停地下, 有些人却注定要相遇
你是一片光荣的叶子，落在我卑贱的心
像往常一样我为自己生气并且歌唱
那么乏力, 爱也吹不动的叶子

作者姓名

2020 年 7 月 10 日

于华南理工大学