一种去中心化的在线边缘任务调度与资源分配方法

彭青蓝[1] 夏云霓[1] 郑万波[2] 吴春蓉[1] 庞善臣[3] 龙 梅[4] 蒋 宁[5]

[1](重庆大学计算机学院软件与理论重庆市重点实验室 重庆 400044)

[2](昆明理工大学数据科学研究中心 昆明 650500)

[3](中国石油大学计算机科学与技术学院 山东 青岛 266510)

[4](重庆猪八戒网络科技有限公司 重庆 401123)

[5](马上消费金融股份有限公司 重庆 401121)

摘 要 创新移动应用迅速发展和5G通讯技术的成熟落地引发了终端用户对计算资源下沉至边缘的巨大需求，从而推动了多接入边缘计算(Multi-access Edge Computing，MEC)概念的提出和发展.在MEC环境中，用户可以将移动任务卸载到周围部署的边缘服务器上来加速移动应用.尽管边缘服务器可以在近用户端提供低时延、高响应性的服务，但其仍面临计算资源有限和用户请求多样带来的挑战，因此需要效率高、实时性强的任务调度与资源分配策略来合理地利用边缘资源.然而，目前针对MEC环境下任务调度和资源分配的方法大多基于中心化架构，并以批处理的方式对某个时间段到达任务进行统一调度与资源分配，因此在面对异构复杂的MEC资源网络和高时延敏感用户需求时具有一定的局限性，此外传统方法还缺少对任务多样性和优先级的考量.针对上述问题，本文提出了一种去中心化的在线任务调度与资源分配方法D-TSRA，该方法以任务优先级加权的卸载响应时间为评价指标，由实时任务调度策略、边缘资源分配策略、和运行时任务迁移策略组成.基于真实边缘环境下数据集的实验表明本文所提出的D-TSRA方法与传统任务调度与资源分配方法相比最多能够减少34.21%的加权任务卸载响应时间.

关键词 多接入边缘计算；移动任务卸载；去中心化调度；在线资源分配；边缘任务迁移

中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2022.01462

A Decentralized Online Edge Task Scheduling and Resource Allocation Approach

PENG Qing-Lan[1] XIA Yun-Ni[1] ZHENG Wan-Bo[2] WU Chun-Rong[1] PANG Shan-Chen[3]
LONG Mei[4] JIANG Ning[5]

[1](*Software Theory and Technology Chongqing Key Lab, College of Computer Science, Chongqing University, Chongqing* 400044)

[2](*Data Science Research Center, Kunming University of Science Technology, Kunming* 650500)

[3](*College of Computer Science and Technology, China University of Petroleum, Qingdao, Shandong* 266510)

[4](*ZBJ Networks Co., Ltd, Chongqing* 401123)

[5](*MSXF Networks Co., Ltd, Chongqing* 401121)

Abstract The rapid development of novel mobile applications and advanced communication technologies have created a huge demand for mobile computing resources, which promotes the emergence of Multi-access Edge Computing (MEC) paradigm. In MEC environments, edge users are allowed to offload their mobile tasks to nearby edge servers to relieve the shortage of mobile resources. Through edge servers could provide low-latency services with high-responsible computing capabilities, they are still faced with the challenge of limited computing resources and the massive

收稿日期：2020-09-15；在线发布日期：2021-03-10.本课题得到国家自然科学基金项目(62172062，62162036)和重庆市重点研发计划项目(cstc2019jscx-xyd0385)资助.**彭青蓝**，博士研究生，主要研究方向为云计算、边缘计算、服务计算.E-mail：qinglan.peng@cqu.edu.cn.**夏云霓**(通信作者)，博士，教授，主要研究领域为服务计算、云计算、边缘计算和随机Petri网络.E-mail：xiayunni@hotmail.com.**郑万波**(通信作者)，博士，副教授，主要研究方向为云计算、灾害应急、可信软件.E-mail：zwanbo2008@163.com.**吴春蓉**，博士研究生，主要研究方向为数据挖掘和服务计算.**庞善臣**，博士，教授，主要研究领域为可信计算、服务计算、云计算、Petri网.**龙 梅**，硕士，高级工程师，主要研究方向为众包系统、服务计算.**蒋 宁**，硕士，首席信息官，主要研究方向为服务计算和云计算.

offloading requests. Thus, smart task scheduling and resource provision strategies with high real-time property are in high demand for better resource utilization. Traditional approaches are usually based on the centralized architecture and batch-processing scheduling mode, which might suffer from low efficiency and high communication overhead. In this paper, we target at the online edge task scheduling and resource allocation problem, propose a decentralized approach, and prove it is $O(1/\epsilon^2)$-competitive when augmented with ϵ additional computing power. Experiments based on real-world edge environments have demonstrated that the proposed approach could achieve at most 34.21% reduction on the average weighted offloading response time.

Keywords　multi-access edge computing; mobile task offloading; decentralized scheduling; online resource allocation; edge task migration

1　引　言

近年来，随着以 5G 为代表的先进通讯技术和移动应用生态环境的不断完善和发展，移动应用愈发呈现出延迟敏感、资源密集、和数据量大的发展势态（如虚拟现实、虚拟增强、沉浸式在线游戏等）. 面对终端用户和移动应用日益增长的性能需求与有限的移动设备资源（计算能力、电池续航、存储空间等）之间的矛盾，作为一个新兴的计算模式，多接入边缘计算[1]（Multi-access Edge Computing，MEC）通过将边缘服务器部署在 5G 基站、智能网关、无线中继等近客户端设备[2-3]上来协助处理移动任务与数据，从而在支撑与强化这些计算资源密集型移动应用上展现出巨大潜力[4-5]. 在这种模式下，边缘用户、设备可以将计算密集型的移动任务卸载至周边部署的可用边缘服务器执行以缓解本地计算资源的短缺[6].

在 MEC 环境下的实际应用场景中，边缘用户可以在任意时刻任意地点提出任务卸载请求，考虑到移动任务普遍对于延迟具有较高的敏感性，因此边缘任务调度器和资源分配中心需要对卸载的任务进行实时调度并分配合理的资源[7]. 不同任务的类型、优先级、数据量等特征具有高度的差异性，而这些信息只有卸载请求到达后才会被揭晓，且存在任务计算量无法进行精确地预估的现实问题[8]. 同时，规格不同的边缘设备也决定了边缘服务器普遍存在的异构性，如有的边缘服务器通过部署定制的 FPGA 或 ASIC 处理器来获得在特定任务（如人脸识别、语音识别、实时美颜等）上的加速效果[9-11]. 因此，针对移动应用对资源和时延日益严苛的要求，面对信息不完整的到达任务流、异构的边服务器配置和复杂的网络拓扑，如何综合考虑不同任务的优先级、类型等信息，作出实时调度并为其分配适当的边缘资源来实现优化的任务优先级加权的卸载响应时间成为一个亟待解决的热点问题.

如图 1(a)所示，目前大多数边缘任务调度与资源分配解决方案[6,12-13]都是基于中心化的架构，依赖一个全局任务调度与资源分配中心来实现资源感知、请求接受、任务调度，与资源分配等功能. 然而这种中心化的架构易受单点故障的侵扰[14]，且

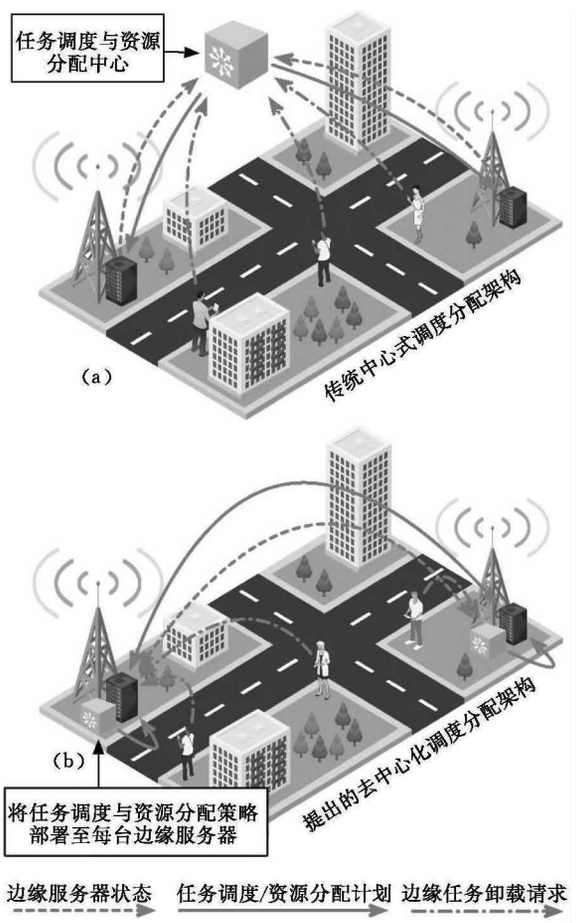

图 1　架构比较：传统中心化的调度分配方法与所提出的去中心化方法

在边缘资源异构性强、用户请求量高的情况下会因求解问题规模爆炸导致调度效率低下、调度方案时效性差等后果.同时这种架构的调度方法通常以批处理的方式运作,即同时完成多个在一个时间窗口内到达任务的调度与资源分配方案.由于用户任务卸载请求的到达具有先后次序,因此这种批处理式的调度方案还会造成额外的任务调度等待时间,从而降低了用户感知的服务质量(QoS)和服务体验(QoE).针对上述问题,本文提出了一种去中心化的实时边缘任务调度与资源分配方法 D-TSRA(Decentralized online edge Task Scheduling and Resource Allocation).如图1(b)所示,在D-TSRA中,边缘任务调度、迁移,与资源分配策略均部署在边缘资源网络中的每一个边缘服务器上,每个边缘节点只需以异步的形式与周围节点完成信息沟通与共享,整体架构没有中心化的调度模块和集中式的调度流程.当用户需要卸载移动任务至边缘资源时,其只需将其传递给周围通讯质量最好的边缘服务器,然后由部署在各个边缘服务器上的 D-TSRA 方法以协作的方式控制任务卸载生命周期中的所有活动(如调度、迁移和资源分配等).

本文的主要贡献体现在以下三个方面:(1)为了更贴合真实边缘应用环境,本文摒弃了任务卸载请求到达时间服从某个已知分布的先验假设,所提出 D-TSRA 方法适用于任意请求到达流.同时,本文还摒弃了任务的计算量可以精确估计且已知的假设.在 D-TSRA 中,任务的计算量在其卸载生命周期中对调度和分配决策全程透明,且任务属性(如类型、优先级和数据量等)只有在该任务到达时刻才被揭示;(2)在系统模型中,本文充分考虑了边缘设备的异构性和移动任务的多样性,即不同边缘服务器对于不同类型任务有不同的处理速度.此外,本文还考虑了由不同用户群体和不同类型任务具备的任务优先级差异情形,并将平均优先级加权的任务卸载响应时间作为调度目标;(3)针对 MEC 环境下的在线任务调度与资源分配问题,本文提出了一种去中心化的在线任务调度与资源分配算法(D-TSRA),并对其竞争比进行了分析.

本文第2节回顾相关工作并对其局限性进行分析;第3节给出系统模型和问题描述;第4节详细描述所提出的去中心化在线解决方案并对其竞争比进行分析;第5节描述基于真实边缘环境数据集的实验并对结果进行分析;最后,第6节总结全文并对未来工作做出展望.

2 相关工作分析

移动任务卸载(Mobile Task Offloading)旨在借助于外部的计算资源(如远程云、移动云、机会云等[15])来执行计算密集型与延迟敏感型任务,从而缓解移动设备资源有限与用户日益增长的性能需求之间的矛盾[16-17].近年来,随着边缘计算模式及其相应标准技术的不断发展成熟,面向边缘计算平台的移动任务卸载成为一个热点问题,涌现出了许多代表性的工作.

一些工作致力于同时调度在一个时间窗口中到达边缘任务卸载请求,通过求解多任务到多服务器之间调度映射的静态优化问题(通常为 NP 难)来获取最终的调度与资源分配方案.如 Chen 等人[6]将超密度网络下某个时间点的多任务卸载与资源分配问题描述为一个混合整数规划(Mixed Integer Programming,MIP)问题,然后将其分解、转化为两个凸优化问题并提出了一个基于拉格朗日乘子的方法来进行求解.Alameddine 等人[18]将任务卸载成功率作为调度目标并提出了一个基于逻辑的 Benders 分解方法来减少相应 MIP 问题的规模.Yang 等人[19]将卸载能耗与时延作为调度目标,将多用户对多服务器的卸载建模为一个博弈问题并提出了一个基于势博弈的分布式求解方法.Dai 等人[12]提出了一个两层任务卸载框架来解决高度异构网络环境下具有依赖关系的移动任务的卸载问题,然后分别提出了一个基于半定规划与线性凸优化的方法来求解相应的边缘用户安置和任务卸载决策.Yang 等人[13]将生成卸载方案的过程看做一个分类问题,并提出了一个前馈神经网络模型来支撑卸载调度决策.

同时也有一些工作致力于动态边缘任务调度与资源分配,并将长期的卸载 QoS 作为调度指标.如 Liu[20]和 Du[21]等人假设移动任务卸载请求的到达时间独立同分布且平均到达率提前已知,而 Zhao 等人[22]则假设用户的任务卸载需求服从一个概率已知的二项分布.此外还有 Chen[23]和 Xu[24]等人认为卸载请求到达强度服从泊松分布且其都用一个随机的到达强度来描述不同时刻这些请求到达模式的波动.基于这些假设,这些工作建立起了相应的队列模型并运用李雅普诺夫优化(Lyapunov Optimization)的方法通过维持当前系统的稳定来实现在线调度.Huang 等人[25]提出了一种基于深度强化学习

的方法来获取实时调度决策，然而，他们在系统模型中仅考虑了单台边缘服务器下的决策问题. 武汉理工大学的刘伟等人[26]针对边缘服务器资源受限下的任务卸载问题，提出了一种面向多用户的串行任务动态卸载策略，并提出了一种基于化学反应优化算法的求解方法.

经过对相关研究的分析，得出目前相关工作仍在以下三个方面存在局限：(1) 大多数边缘任务调度与资源分配算法是为中心化的调度器设计，而这种中心化的架构易受单点故障的侵扰，还会造成为保证全局信息一致性而带来的额外通信开销；(2) 一些工作[6,12,19,27]采用批处理的方式调度在一个时间窗口内到达的所有任务，而这会造成额外的等待调度时间，同时还面临着请求高峰期由某时刻任务高强度到达导致问题规模爆炸而带来的性能下降；(3) 一些工作[20,22-23]假设用户卸载请求到达服从某个已知的分布且任务大小已知. 然而在真实应用场景中，边缘任务的到达时间随机性较大，且存在到达地域上分布不均的特点，无法用某个分布来精确描述. 任务的属性(如类型和优先级等)信息也只有在其真正到达时才揭示给调度算法，同时任务的具体计算量也无法在运行时精确估计. 这些假设可能导致据此设计的相应算法和策略在真实环境下鲁棒性差和泛化性弱的后果.

综上所述，尽管当前工作在中心化的边缘任务调度与资源分配上取得了突破性的进展，但其缺少了对如何在复杂环境下面对不完整信息如何做出实时高效调度决策的思考，且忽略了对任务多样性和一些任务属性无法预知的考量，因而在真实应用环境下具有一定的局限性. 因此如何设计面向异构边缘服务器和多样化用户请求的、去中心化的、在线任务调度与资源分配方法成为解决 MEC 环境下计算卸载问题的关键挑战.

3　系统建模与问题描述

本节首先对 MEC 环境下的任务卸载系统进行建模，然后给出了要解决的在线边缘任务调度与资源分配问题的详细描述.

3.1　系统建模

如图 1 所示，在 MEC 环境下，边缘服务器可以被部署在近用户侧的信号基站上来承载用户的计算卸载. 本文用集合 $E=\{e_1,e_2,\cdots,e_m\}$ 表示一个区域内部署的 m 台边缘服务器. 每个边缘服务器可以用一个三元组 $e_i=(S_i,L_i,f_i)$ 来表示，其中 $S_i=(s_i^1,s_i^2,\cdots,s_i^o)$ 代表边缘服务器 e_i 针对不同类型的任务的处理速度，单位为每秒浮点运算次数(FLOPS)，$L_i=(lot,lat)$ 代表 e_i 的地理坐标(经纬度)，f_i 表示边缘服务器 e_i 的信号覆盖半径. 本文考虑 SA 或 NSA 组网下的 5G 网络，其中基站之间可以通过 X2 或 Xn 接口进行通信[28]，用边缘资源网络 $N=(E,B)$ 来表示某区域内部署的 m 台边缘服务器属性和其网络拓扑信息，其中 $B=\{b_{(ij)}\mid i,j\in[1,m],i\neq j\}$，$b_{(ij)}\neq 0$ 表示边缘服务器 e_i 和 e_j 之间存在直接通信链路且带宽为 b_{ij}.

本文用 $R=\{r_1,r_2,\cdots,r_n\}$ 表示在该区域内活动的所有移动用户在某时间段内卸载至边缘资源网络的任务的集合. 其中每个任务可以用一个六元组 $r_j=(a_j,L_j,c_j,w_j,d_j,p_j)$ 表示，a_j 代表任务 r_j 到达的时间，$L_j=(lot,lat)$ 表示发出该任务卸载请求用户的地理位置，c_j 代表该任务的类型，$w_j\in N^+$ 表示该任务的优先级，d_j 与 d_j' 分别表示卸载该任务所需要传输的上行和下行数据量，p_j 为完成该任务所需要的计算量，单位为浮点运算数(FLOPs).

多核心通用处理器、超线程技术，与特定任务定制化芯片(如 GPU、FPGA、ASIC 等)等技术是提高处理机效率、满足不同任务需求的有效方式. 因此本文考虑现实中边缘服务的异构性和移动任务普遍存在的多样性，即不同边缘服务器具有不同的资源配置，同一边缘服务器处理不同任务的能力也不相同. 此外，本文还考虑一个边缘服务器可以同时并行处理多个移动任务，并通过为其分配不同的处理机时间来实现边缘计算资源的分配[29-30]. 本文用 $J_i(t)$ 表示在 t 时刻边缘服务器 e_i 上正在处理的任务集合，$v_j(t)$ 表示 t 时刻服务器 e_i 分配给任务 $r_j\in J_i(t)$ 计算资源的比例，其中 $\sum_{r_j\in J_i(t)}v_j(t)\leq 1$. 令 $l_{ij}=s_i^{c_j}$ 表示任务 r_j 在边缘服务器 e_i 上资源独占情形下的执行速度，t 时刻任务 r_j 在服务器 e_i 上的实际执行的速度可以被计算为 $q_j(t)=l_{ij}\cdot v_j(t)$，则任务 r_j 的完成时间 C_j 可被计算为

$$C_j=\arg\min_t\left\{\int_{a_j}^{t}q_j(t)\mathrm{d}t\geq p_j\right\},$$

因此，任务 r_j 的实际执行用时为

$$F_j=C_j-a_j.$$

为了减少卸载延迟和减轻边缘网络内通信开销，与相关工作一致[6,27,31]，本文考虑移动用户只能将任务卸载至其信号直接可达的边缘服务器和这些

服务器 k 跳通讯距离内的服务器上. 本文用 $E_j(k)$ 表示可供任务 r_j 卸载的边缘服务器构成的资源池，$E_j(k)$可以被表示为

$$E_j(k)=\{e_x \mid H(e_x,e_i)<k, e_i\in E, e_x\in E_j\} \quad (1)$$

其中函数 $H(e_x,e_i)$表示边缘服务器 e_x 与 e_j 之间的通讯跳数，E_j 为发出请求 r_j 的用户周围信号直接可达边缘服务器的集合：

$$E_j=\{e_i \mid D(e_i,r_j)<f_i, e_i\in E\} \quad (2)$$

其中函数 $D(e_i,r_j)$表示发送任务 r_j 卸载请求的用户与边缘服务器 e_i 之间的距离.

任务 r_j 的卸载响应时间 T_j 由其任务数据的传输时间 D_j 和任务的执行时间 F_j 组成：

$$T_j=D_j+F_j,$$

其中 D_j 又由任务数据从用户设备至边缘服务器的上行传输时间、由调度或迁移造成的任务数据在边缘服务器之间传输时间和最终计算结果由边缘服务器返回用户设备的下行传输时间三部分构成：

$$D_j=\frac{d_j}{\beta\log_2[1+\rho_j\tau_{ij}/\lambda^2]}+\sum_{(x,y)\in M_j}\frac{d_j+d'_j}{b_{xy}}+\frac{d'_j}{\beta\log_2[1+\rho_j\tau_{i'j}/\lambda^2]},$$

其中 β 为基站的信道带宽，ρ_j 为移动设备的信号传输功率，M_j 为任务 r_j 的所有调度和迁移记录，$(x,y)\in M_j$ 表示 r_j 一次经由服务器 e_x 到达 e_y 的记录，τ_{ij} 和 $\tau_{i'j}$ 分别任务 r_j 的请求设备与 r_j 的首次到达和经由返回边缘服务器之间的信道增益，λ^2 为设备高斯噪音功率[6]. 注意，这里的经由返回服务器指的是任务结果回传至用户设备过程中经过的最后一台边缘服务器，由于用户和边缘服务器之间、边缘服务器内部之间的通信方式不同[32]，带宽不一致，因此需要分开计算.

然而在真实边缘应用环境中，卸载任务的计算量往往无法提前预知或精确估计[1,33]. 因此与目前假设任务计算量 p_j 提前已知的方法不同，本文中任务的计算量 p_j 在执行过程中对任务调度算法和资源分配策略保持透明. 即本文所提出的 D-TSRA 方法的决策依据仅需周边边缘服务器状态与不含 p_j 的运行时请求信息，不依赖于此处对 C_j、F_j、D_j 和 T_j 的精确估计. 这里对 C_j、F_j、D_j 和 T_j 进行建模和计算是仅用于在实验部分对调度结果指标进行评估，以验证所提出算法的性能和效率.

3.2 问题描述

基于以上系统模型，本文致力于解决如下问题：给定一个部署了 m 台边缘服务器 E 的区域，这 m 台边缘服务器和其网络拓扑 B 构成边缘资源网络 N，活动在该区域内的移动用户在一段时间内共发出 n 个边缘任务卸载请求 R，如何在每个请求到达的时刻实时调度其到恰当的边缘服务器上并为其分配合理的计算资源使得这些任务的平均优先级加权卸载响应时间最小. 该问题可以描述为

$$\text{Min}: \frac{1}{n}\left(\sum_{j-1}^{|R|-n} w_j\cdot T_j\right) \quad (3)$$

$$\text{s.t.}: g(r_j)\in E_j,\ \forall j \quad (4)$$

$$h(r_j)\in E_j,\ \forall j \quad (5)$$

$$e_x\in E_j(k),\ \forall j,(x,y)\in M_j \quad (6)$$

$$e_y\in E_j(k),\ \forall j,(x,y)\in M_j \quad (7)$$

其中函数 $g(r_j)$ 与 $h(r_j)$ 分别表示任务 r_j 的首次到达和最终结果经由返回边缘服务器. 如目标(3)所示，所提出的在线调度与资源分配问题致力于优化卸载移动任务的平均加权响应时间(Average Weighted Response Time，AWRT). 其中约束(4)和(5)将任务由用户设备与边缘服务器之间的数据传输限制在基站的信号覆盖范围之内. 约束(6)和(7)则表明了任务只能被调度或迁移至与其相对应的边缘资源池内，又如式(1)和(2)所示，这个资源池由发出该卸载任务请求的用户的地理位置决定.

如上文所述，该问题中用户的任务卸载请求随着时间的推移不断到达，任务到达前调度算法和相应分配策略无法预知任务的具体属性，且即使任务到达后也无法准确估计任务的计算量 p_j. 为满足移动应用对于低延迟的需求，调度器必须在任务到达后做出实时调度计划并为其分配合理的计算资源，并可以在运行时根据服务器状态和任务优先级信息等信息动态做出资源重分配、任务迁移等决策来实现长期的卸载响应时间优化. 这是一个典型的在线调度与分配问题，Feldman 等人[34] 和 Agrawal 等人[35] 已经证明没有在线算法可以得到最优解，因此针对该问题设计具有更低竞争比的在线近似算法成为需要解决的首要问题.

4 提出的 D-TSRA 方法

基于上一节提出的系统模型，针对要解决的在线边缘任务调度与资源分配问题，本文提出了一种去中心化的在线解决方法，简称 D-TSRA. 本节首先对该方法进行描述，然后对其竞争比进行分析.

4.1 方法概览

如图 2 所示，所提出的 D-TSRA 方法由实时任

务调度策略、边缘资源划分策略和实时任务迁移策略三部分组成. 在生产环境中，这三个策略均以去中心化的形式部署在边缘资源网络中每一台边缘服务器上，每台边缘服务器只需与自己周围 k 跳通讯内的服务器以异步的形式保持沟通并维持状态，无需一个随时掌握全局服务器状态与任务信息的调度中心与资源分配机构. 此外，D-TSRA 还摒弃了传统方法普遍采用的批处理任务调度与资源分配的模式，其调度与分配决策在每个任务到达时实时作出，避免了任务在待调度队列中的等待过程，减少了优化问题的规模，提高了整个任务卸载系统的响应性.

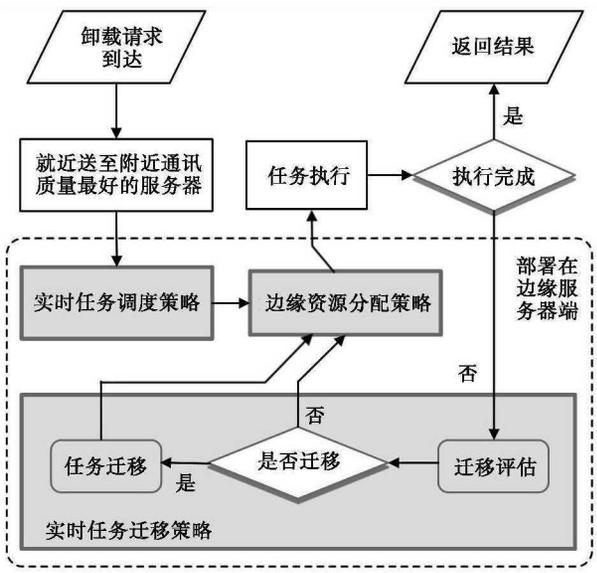

图 2　D-TSRA 方法的内部结构及一次任务卸载的流程

在 D-TSRA 中，一次完整的任务卸载流程如下：(1) 当用户需要卸载任务 r_j 至边缘资源池时，用户首先将任务直接交给周围信号可达的通讯质量最好的边缘服务器(即首次达到边缘服务器)；(2) 然后首次到达服务器根据自身掌握的边缘资源网络信息 N 和任务的 E_j 属性由根据式(1)为其计算其可用的规模为 k 的边缘计算资源池 $E_j(k)$；(3) 接着由部署的实时调度策略将任务指派到合适的目标服务器并由资源分配策略划分合适的计算资源；(4) 在任务执行过程中，边缘服务器上部署实时任务迁移策略会在每个时刻对任务队列中正在执行的任务进行迁移评估并据此制定迁移计划；(5) 任务执行完成返回结果.

4.2　实时任务调度与资源分配

算法 1 描述了 D-TSRA 方法中的实时任务调度策略. 可以看出当一个边缘服务器在 t 时刻收到用户的卸载请求后，其首先对任务的具体属性进行分析(如第 1 行所示)，接着为其构建可供 r_j 选择的、规模为 k 的边缘资源池 $E_j(k)$(如第 2 行所示). 为 r_j 构建边缘资源池是为其接下来的调度和以后的迁移评估做准备，作为 r_j 的一个固有属性，$E_j(k)$ 在构建完成后可以直接复用，无需重新计算. 接着由式(8)评估若将任务 r_j 调度到服务器 $e_i \in E_j(k)$ 上的效用值(如第 3～9 行所示)，最后将任务 r_j 调度至具有最高效用值服务器并触发该服务器的资源重分配(如第 10～12 行所示).

算法 1.　实时调度策略.

输入：当前时刻 t；移动任务卸载请求 r_j；资源池规模 k

1. $(L_j, c_j, w_j) \leftarrow r_j$
2. $E_j(k) \leftarrow$ 由式(1)构建 r_j 适用的规模为 k 的边缘资源池，并作为 r_j 的固有属性
3. $S \leftarrow \varnothing$
4. FOREACH $e_i \in E_j(k)$ DO
5. $\quad J_i(t)' \leftarrow J_i(t)$
6. $\quad J_i(t)' \leftarrow J_i(t)' \cup \{r_j\}$
7. $\quad s_i \leftarrow$ 基于 $J_i(t)'$ 由式(8)计算 $\phi(j,t)$
8. $\quad S \leftarrow S \cup \{s_i\}$
9. ENDFOR
10. $x \leftarrow \arg\max_{x} S$
11. 将 r_j 调度至 e_x 并追加到其任务队列 $J_x(t)$ 尾部
12. 由式(10)将 $J_x(t)$ 中的任务进行计算资源重分配

令 $J_i(t)$ 表示 t 时刻边缘服务器 e_i 的任务队列，本文考虑边缘服务器任务队列中任务完全并行执行[36-37]，任务在队列中的顺序表示其到达的先后次序. 若任务 $r_j \in J_i(t)$，则 r_j 在当前队列的效用值 $\phi(j,t)$ 可计算为

$$\phi(j,t)=\frac{l_{ij}}{\mathcal{W}_j(t)+w_j} \tag{8}$$

其中 $\mathcal{W}_j(i)$ 代表队列 $J_i(t)$ 中排在任务 r_j 之前所有任务的优先级之和，即

$$\mathcal{W}_j(t)=\sum_{r_y \in J_i(t)} w_y \cdot \pi(j,y) \tag{9}$$

其中布尔函数 $\pi(j,y) \in \{0,1\}$ 表示任务 r_y 是否比 r_j 先到达(即在队列 $J_i(t)$ 中 r_y 是否排在 r_j 之前)，$W_i(t)=\sum_{r_j \in J_i(t)} w_j$ 表示边缘服务器 e_i 的任务队列 $J_i(t)$ 中所有任务的优先级之和.

与操作系统中多进程的计算资源分配[29-30]类似，本文考虑边缘服务器的动态资源划分，即一个任务在其执行生命周期的不同时段可以被分配不同的资源. 每当一个边缘服务器的任务队列发生变化时，该服务器的资源分配策略就会被触发，新任务到达或旧任务离开都可以导致一个边缘服务器的任务队

列发生变动.其中新任务到达可能是任务的首次调度,也可能是执行中的任务被迁移到此,旧任务离开可能是任务完成返回,也可能是某执行中任务的迁出.在 D-TSRA 的边缘资源分配策略中,任意边缘服务器 e_i 在任意时刻 t 按以下规则将计算资源分配给当前队列 $J_i(t)$ 中的所有任务:

$$v_j(t)=\frac{[\mathcal{W}_j(t)+w_j]^{\kappa+1}-\mathcal{W}_j(t)^{\kappa+1}}{W_i(t)^{\kappa+1}},$$

其中 $\kappa\in N^+$ 为控制资源分配离散程度的一个参数.可以看出当 $\kappa=0$ 时,其完全退化为严格的加权轮询策略(Weighted Round Robin,WRR),但随着 κ 值的逐渐增大,队列尾部的具有较高优先级 w_j 的任务会逐渐获得更大的资源比例,而这个特性确保了迟到在尾部但具有较高优先级的任务可以得到及时的执行,从而避免了低优先级先到任务通过优先级放大作用对整体响应时间占比过高.此外,该策略还一定程度上保留了 WRR 的一个重要特点,若一个新的任务被调度到队列尾部,那么之前到达的具有相同权优先级任务会以相同比例降低分配到的资源,从而保证了资源分配的公平性.

由式(9)为一个边缘服务器任务队列中的所有任务计算 $\mathcal{W}_j(t)$ 的时间复杂度为 $O(|J|)$,计算 $W_i(t)$ 的时间复杂度也为 $O(|J|)$,其中 $|J|$ 表示边缘服务器的运行时平均队列长度.由式(8)为任务计算效用值的时间复杂度为 $O(1)$,所以为一个队列中所有任务计算其在当前队列中效用值的时间复杂度为 $O(|J|)$,因此进行一次在线调度扫描的时间复杂度为 $O(|E_k|\cdot(|J|+\log|E(k)|))$,其中 $|E(k)|$ 为规模为 k 的边缘资源池内边缘服务器的平均数量.由式(10)为一个任务计算资源分配比例的时间复杂度为 $O(1)$,因此为一个队列中所有任务计算资源分配比例的时间复杂度也为 $O(|J|)$.可以看出效应值计算与资源分配的时间复杂度均与队列中任务个数呈线性关系,具有良好的可扩展性,因此可以应对请求强度不断增加带来的计算开销.

4.3 实时任务迁移策略

在一个 MEC 任务卸载系统中,运行时每个时刻都有新任务到达、旧任务完成离开,因此系统中每个边缘服务器的任务队列、资源分配情况和任务的效应值都处于实时变化状态.为了捕捉这些变化并作出及时调整,D-TSRA 方法还包含了一个实时任务迁移策略.该策略可以以守护进程的形式部署在每一台边缘服务器上来不间断地对当前队列中的任务进行状态扫描、迁移评估和迁移决策,以确保不同优先级的任务始终在恰当的服务器上执行以实现长期优化的平均优先级加权响应时间.

算法 2 描述了 D-TSRA 方法中的实时任务迁移策略,可以看出该策略首先评估当前任务队列中每个任务的效用值(如第 3 行所示),接着对于每个任务评估将其迁移至其它可用边缘服务器上的效用值(如 4~10 行所示).若迁移后某任务的最大效用值高于当前效用值,则为该任务执行迁移操作并触发资源重分配(如 11~15 行).最后,在完成当前任务队列一轮迁移扫描后对队列中剩余的任务进行资源重分配.

算法 2. 运行时迁移策略.

输入:当前边缘服务器 e_i;服务器 e_i 的任务队列 $J_i(t)$

1. FOREACH $r_j\in J_i(t)$ DO
2. $E_j(k)\leftarrow$得到 r_j 适用的边缘资源池
3. $s_j\leftarrow$由式(8)计算 $\phi(j,t)$
4. $S\leftarrow\varnothing$
5. FOREACH $e_x\in E_j(k)$ DO
6. $J_x(t)'\leftarrow J_x(t)$
7. $J_x(t)'\leftarrow J_x(t)'\cup\{r_j\}$
8. $s_x\leftarrow$基于 $J_x(t)'$ 由式(8)计算 $\phi(x,t)$
9. $S\leftarrow S\cup\{s_x\}$
10. ENDFOR
11. $x\leftarrow\arg\max_x S$
12. IF $s_j<s_x$ THEN
13. 将 r_j 调度至 e_x 并追加到其任务队列 $J_x(t)$ 尾部
14. 触发服务器 e_x 的计算资源重分配
15. ENDIF
16. ENDFORE
17. 由式(10)将 $J_i(t)$ 中的任务进行计算资源重分配

边缘任务按照其对数据时效性的敏感程度可分为可断点执行类任务和非断点执行类任务.断点执行类任务在迁移时其下文数据可随任务本身一同迁移,迁移后已完成部分无需重新计算,使用新分到的资源从检查点继续执行[38-41](如医疗影像数据边缘处理、交通违法边缘识别、建筑结构安全边缘检测等应用).而非断点执行类任务在迁移后受数据时效性的影响,已完成部分失效丢弃,需要重新执行(如手机游戏计算卸载、车载实时障碍物检测、轨迹预测、路径生成、智慧园区实时入侵检测等).考虑到真实边缘应用环境中很难对不同边缘任务的计算量(即 p_j)做出实时、精准的预估,因此,所提出的 D-TSRA 方法在运行时不依赖任务的 p_j 属性,而这个特性也使得 D-TSRA 的迁移策略能够同时支持断点执行类任务和非断点执行类任务.基于本文任务计算量

p_j未知的考量，所提出的实时任务迁移策略可同时应用于断点执行任务和非断点执行任务. 这是由于非断点执行任务迁移后等价于任务的计算量 p_j 增加，而所提出的 D-TSRA 方法的任务调度和迁移不依赖于预估的任务预期完成时间(需要 p_j 参与估计). 因此，与传统的断点执行任务适用的迁移方法相比，D-TSRA 的实时任务迁移策略增加了对非断点执行类任务的支持.

由于为一个边缘服务器执行一次资源重分配时间复杂度为 $O(|J|)$，因此算法 2 所描述的一次运行时任务迁移扫描的时间复杂度为 $O(|E(k)|\cdot(|J|+\log|E(k)|))$.

定理 1.　在任意 t 时刻，给定边缘服务器任务队列的状态 $J_i(t), e_i\in E$，根据算法 2 所描述的迁移策略，系统中所有正在执行的任务必然在有限次迁移操作后收敛至稳态，即不再具备再次发生任务迁移的条件.

由算法 2 的 11～14 行可知每个任务的迁移是为了增加自身的效应值，由式(8)可知一个任务迁移后可能会对迁出队列中的其他任务的效应值造成影响. 因此可考虑将 t 时刻系统中每个任务都看作一个独立的个体，其做出的迁移操作看作该个体的策略，将所有个体在 t 时刻的策略选择看作一个势博弈(Potential Game)[42]，从而利用势博弈有限步改进(Finite Improvement Property)的性质证明该过程必然在有限步内收敛至纯纳什均衡(Pure Nash Equilibrium，PNE).

引理 1.　在任意 t 时刻，给定边缘服务器任务队列的状态 $J_i(t), e_i\in E$，系统中所有正在执行的任务在边缘服务器之间的迁移活动是一个势函数为 $\Phi(t)=\sum_{i-1}^{m}\sum_{r_j\in J_i(t)}\phi(j,t)$ 的势博弈.

证明.　要证明迁移过程是一个势博弈，只需证明势函数 $\Phi(t)$ 在迁移操作后单调递增即可，因为在势博弈中，每个个体收益增加(效应值)均能映射到一个全局势函数的增加上，即存在

$$\phi'(j,t)>\phi(j,t)\Rightarrow\Phi'(t)>\Phi(t),$$
$$\forall r_j,\forall r_y\neq r_j,\phi'(y,t)=\phi(y,t),$$

其中 $\phi'(j,t)$ 表示 t 时刻过后任务 r_j 的效应值，$\Phi'(t)$ 表示 t 时刻过后的势函数值.

任务 r_j 在 t 时刻有两种操作可供选择：(1) 不迁移维持原状；(2) 迁移至其他边缘服务器以获得更大的效应值. 下面分情况进行讨论.

(1) 当 r_j 选择维持原状时，$\phi'(j,t)=\phi(j,t)$，因为没有发生迁移操作，所以其对其他任务没有影响，因此 $\Phi'(t)=\Phi(t)$.

(2) 当 r_j 选择迁移至别的服务器时，令 e_b 和 e_a 分别表示 r_j 的迁出和迁入服务器，这里还要分情况进行讨论：

① 若任务 r_j 位于其迁出服务器 e_b 任务队列的末端，即

$$\pi(j,y)=1,\ \forall r_y\in J_b(t),$$

则由任务效应值计算式(8)可知，r_j 迁出后服务器 e_b 任务队列中剩余任务的效应值保持不变. 又因 r_j 迁入服务器 e_a 后位于队列末端，所以服务器 e_a 任务队列中原有任务效应值也保持不变. 如算法 2 中第 11～14 行所述，任务 r_j 迁移的触发条件是迁移后效应值增加，因此 $\phi'(j,t)>\phi(j,t)$. 由于在这种情况下除了任务 r_j 的效应值增加外，其他任务的效应值都保持不变，所以 $\Phi'(t)>\Phi(t)$.

② 若任务 r_j 不位于其迁出服务器 e_b 任务队列的末端，令 $\mathcal{J}_b(j,t)^*$ 表示排在其后的任务的集合：

$$\mathcal{J}_b(j,t)^*=\{r_j\mid\forall r_y\in J_b(t),\pi(j,y)=0\},$$

则除了 r_j 迁移后效应值增加，$\phi'(j,t)>\phi(j,t)$，根据式(8)，$\mathcal{J}_b(j,t)^*$ 中任务的效应值也会增加，即：

$$\phi'(y,t)>\phi(y,t),\ r_y\in\mathcal{J}_b(j,t)^*.$$

因此这种情况下，迁移后有多个位于迁出服务器 e_b 上任务的效应值增加，所以 $\Phi'(t)>\Phi(t)$.　　证毕.

由于任意势博弈一定存在纯纳什均衡解[43]，因此由引理 1 可知，算法 2 描述的迁移策略必然在经过有限步操作后收敛至纯纳什均衡，定理 1 得证. 最优反应动态(Best-Response Dynamics)是描述参与博弈的所有个体寻找纯纳什均衡过程的动力学模型，非常适用于分析势博弈收敛至纳什均衡的速度和效率[44]. 令 ϵ-PNE 表示一个 ϵ 近似的纯纳什均衡. 那么对于引理 1. 中所提出的势博弈模型，有

$$\phi'(j,t)\geq(1-\epsilon)\cdot\phi(j,t),$$

其中 $\epsilon\in[0,1]$. 由 Narahari[44] 的证明可知，该势博弈收敛至 ϵ-PNE 所需要的最大迭代次数为 $O\left(\frac{k\cdot\alpha}{\epsilon}\cdot\ln\frac{\phi(t)}{\phi_{\max}}\right)$，其中 k 为势博弈参与者的个数，$\alpha>1$ 是该势博弈满足成本函数边界跳跃条件的参数，$\phi_{\max}$ 为势函数的最终收敛值. 即在任意 t 时刻，给定边缘服务器任务队列的状态 $J_i(t), e_i\in E$，算法 2 所描述迁移策略的迁移次数近似上界为 $O\left(\frac{m\cdot\alpha}{\epsilon}\cdot\ln\frac{\phi(t)}{\phi_{\max}}\right)$.

4.4　竞争比分析

竞争比分析是评估在线算法性能的有效途径.

令 I 表示一个在线决策问题的一个实例，即一组到达事件构成的集合(本文问题中为 R)，$\Im$ 表示一个在线决策问题中所有实例组成的集合，ALG 和 OPT 分别表示针对该问题的在线算法和相应的最优离线算法. 对于算法 ALG，其在某个决策时刻 t 只知道已到达事件的集合 $I'\in I$(本文问题中为 $\{r_i \mid r_i\in R, a_i\leq t\}$)，而 OPT 则在任意时刻享有 I 的全部信息. 则算法 ALG 的竞争比定义为[45]

$$c=\sup_{I\in\Im}\left\{\frac{ALG(I)}{OPT(I)}\right\}.$$

也称算法 ALG 是 c-$competitive$ 的.

定理 2. 对于任意 $\epsilon>0$，假设所有边缘服务器的计算性能为原有的 $1+\epsilon$ 倍，针对于 3.2 小节提出的在线任务调度与资源分配问题，所提出的 D-TSRA 方法竞争比为 $O(1/\epsilon^2)$，即 D-TSRA 是 $O(1/\epsilon^2)$-$competitive$ 的.

为了分析所提出算法的竞争比，首先写出所求问题的标准线性规划形式：

$$\begin{aligned}&\text{Min}:\sum_{r_j\in T}\sum_{e_i\in E}\sum_{t>r_j}\left[\frac{l_{ij}(t-r_j)}{p_j}+1\right]\cdot w_j\cdot x_{ijt} \quad (11)\\&\text{s.t.}:\sum_{e_i\in E}\sum_{t>r_j}\frac{l_{ij}\cdot x_{ijt}}{p_j}\geq 1,\ \forall r_j\in T\\&\quad\sum_{r_{j,t}>r_j}x_{ijt}\leq 1,\ \forall e_i\in E,t\\&\quad x_{ijt}\geq 0,\ \forall e_i\in E,\ r_j\in T,\ t\geq r_j,\end{aligned}$$

其中布尔变量 $x_{ijt}\in\{0,1\}$ 代表任务 r_j 在时刻 t 是否在服务器 e_i 上处于运行状态. 由实验结果可知任务传输时间对整体卸载响应时间贡献较小，为方便分析，这里仅将任务优先级加权的总执行时间作为线性规划的目标函数. 接下来使用对偶拟合法(Dual-Fitting)来进行竞争比分析，问题(11)的对偶问题可以表述为

$$\begin{aligned}&\text{Max}:\sum_{r_j\in T}\alpha_j-\sum_{e_i\in E}\sum_{t}\beta_{it} \quad (12)\\&\text{s.t.}:\alpha_j\geq 0,\ \forall r_j\in T\\&\quad\beta_{it}\geq 0,\ \forall e_i\in E,t\\&\quad\frac{l_{ij}\cdot\alpha_j}{p_j}-\beta_{it}\leq\frac{w_j\cdot l_{ij}\cdot(t-r_j)}{p_j}+w_j,\\&\qquad\forall e_i\in E,r_j\in T,t\geq r_j\end{aligned}$$

针对此类问题，若不假设处理机享有额外的计算能力，无法得到有界的竞争比[8,46-47]. 因此，接下来的分析均建立在缘服务器计算能力为原始的 $\eta=1+3\epsilon$ 倍的假设之上.

令 $\epsilon=1/\kappa$，其中 κ 为 D-TSRA 方法中边缘资源分配策略的参数，$\mathcal{J}_j(t)$ 表示队列 $J_i(t)$ 中排在任务 r_j 之前的任务的集合. 因此在 t 时刻，对任务 r_j 而言，其对任务 $r_{j'}\in\mathcal{J}_j(t)$ 造成的瞬时加权延迟 $\delta_j(t)$ 可被计算为

$$\delta_j(t)=\frac{1}{\eta}\left[\sum_{r'_j\in\mathcal{J}_i(t)}[w_{j'}v_j(t)+w_jv_{j'}(t)]+w_jv_{j'}(t)\right],$$

可以看出 $\delta_j(t)$ 是 r_j 对 $r_{j'}\in\mathcal{J}_j(t)$ 造成的瞬时延迟和自身延迟的加权和，其也可表示为

$$\delta_j(t)=\frac{1}{\eta}\left[(\mathcal{W}_j(t)+w_j)\cdot v_j(t)+w_j\cdot\sum_{r'_j\in\mathcal{J}_i(t)}v_{j'}(t)\right],$$

令 Δ_j 表示 $\delta_j(t)$ 在任务 r_j 生命周期上的累加，则

$$\Delta_j=\int_{t-r_j}^{C_i}\delta_j(t)\mathrm{d}t,$$

易得 $\sum_{r_j}w_j\cdot F_j=\sum_{r_j}\Delta_j$.

令 $\beta_{it}=\frac{1}{\kappa+3}W_i(t)$，即 β_{it} 的值与 t 时刻边缘服务器 e_i 上正在执行的任务的优先级成正比，令 $\alpha_j=\frac{1}{\kappa+2}\Delta_j$，即 α_j 的值与任务 r_j 对其之前任务导致的累积加权延迟成正比. 将 α_j 和 β_{it} 代回对偶问题(12)得

$$\begin{aligned}\sum_{r_j\in T}\alpha-\sum_{e_i\in E,t}\beta_{it}&=\sum_{r_j\in T}\frac{\Delta_j}{\kappa+2}-\sum_{e_i\in E,t}\frac{W_i(t)}{\kappa+3}\\&=\epsilon\left[\sum_{r_j\in T}\frac{\Delta_j}{1+2\epsilon}-\sum_{e_i\in E,t}\frac{W_i(t)}{1+3\epsilon}\right]\\&=\epsilon\sum_{r_j\in T}w_jF_j\left(\frac{1}{1+2\epsilon}-\frac{1}{1+3\epsilon}\right)\\&=O(\epsilon^2)\sum_{r_j\in T}w_jF_j,\end{aligned}$$

因此对偶问题(12)存在界为 $O(\epsilon^2)$ 的可行解. 现在只需证明所设的 α_j 与 β_{ij} 值满足对偶问题的约束即可证明定理 2.

对偶问题(12)的约束与任务 r_j 在生命周期的任意时刻 $t\in[r_j,C_j]$ 导致的加权时延相关，为方便证明，将其生命周期划分为两段分别进行分析. 令 $t^*\in[r_j,C_j]$，则有

$$\Delta_j^1(t^*)=\int_{t-r_j}^{t^*}\delta_j(t)\mathrm{d}t,$$

$$\Delta_j^2(t^*)=\int_{t-t^*}^{C_j}\delta_j(t)\mathrm{d}t.$$

引理 2. 对于任意任务 r_j 在任意瞬间 $t^*\in[r_j,C_j]$，有

$$\Delta_j^1(t^*)\leq(\kappa+2)\cdot w_j\cdot(t^*-r_j).$$

$$\begin{aligned}\Delta_j^2(t^*)&\leq\frac{1}{\eta}\cdot\frac{\kappa+2}{\kappa+1}\cdot\frac{p_j(t^*)}{\phi(j,t^*)}\\&\leq\frac{1}{\eta}\cdot\frac{\kappa+2}{\kappa+1}\cdot p_j\cdot\frac{\mathcal{W}_j(t^*)+w_j}{l_{i^*j}}.\end{aligned}$$

证明详见附录.

引理 3.　令 $\sigma(j,t)$ 代表 t 时刻任务 r_j 被指派到的服务器，由引理 2 可知，对于任意瞬间 $t\in[r_j,C_j]$ 和被指派到服务器 $e_{i^*}=\sigma(j,t)$ 上的任务 r_j，有

$$\begin{aligned}\Delta_j&=\Delta_j^1(t)+\Delta_j^2(t)\\&\le(\kappa+2)(t-r_j)w_j+\frac{1}{\eta}\cdot\frac{\kappa+2}{\kappa+1}\cdot\\&\quad p_j\cdot\frac{\mathcal{W}_j(t^*)+w_j}{l_{i^*j}}.\end{aligned}$$

对于任意 r_j 和任意时刻 t，当 $x_{ijt}=1$ 时，由于 $\eta=1+3\epsilon$，$\kappa=1/\epsilon$，由引理 3 可得

$$\begin{aligned}\alpha_j-\frac{p_j}{l_{ij}}\beta_{it}&=\frac{\Delta_j}{\kappa+2}-\frac{p_j}{l_{ij}}\cdot\frac{W_i(t)}{\kappa+3}\\&\le w_j\cdot(t-r_j)+\frac{p_j}{\eta(\kappa+1)}\cdot\\&\quad\frac{\mathcal{W}_j(t)+w_j}{l_{ij}}-\frac{p_j}{l_{ij}}\cdot\frac{W_i(t)}{\kappa+3}\\&\le w_j\cdot(t-r_j).\end{aligned}$$

满足对偶问题(12)的约束.

当 $x_{ijt}=0$ 时，同理可得

$$\begin{aligned}\alpha_j-\frac{p_j}{l_{ij}}\beta_{it}&=\frac{\Delta_j}{\kappa+2}-\frac{p_j}{l_{ij}}\cdot\frac{W_i(t)}{\kappa+3}\\&\le w_j\cdot(t-r_j)+\frac{p_j}{\eta(\kappa+1)}\cdot\\&\quad\frac{\mathcal{W}_j(t)+w_j}{l_{ij}}-\frac{p_j}{l_{ij}}\cdot\frac{W_i(t)}{\kappa+3}\\&\le w_j\cdot(t-r_j)+\left[\frac{p_j}{\eta(\kappa-1)}\cdot\right.\\&\quad\left.\frac{\mathcal{W}_j(t)+w_j}{l_{ij}}-\frac{p_j}{\kappa+3}\cdot\frac{W_i(t)}{l_{ij}}\right]\\&\le w_j\cdot(t-r_j)+\frac{w_j\cdot p_j}{l_{ij}}.\end{aligned}$$

显然也满足对偶问题(12)的约束，定理 2 得证.

5　实验与分析

为了验证所提出方法的有效性，本文采用真实边缘环境数据集开展试验并进行案例分析. 本节首先描述了实验设置和实验场景，然后给出了实验结果并对其进行分析.

5.1　实验设置

本文实验基于真实边缘环境数据集 Telecom[48-51]，该数据集记录了上海市市区范围内 30 天内 1500 个基站的地理信息与 13 万条请求数据的到达时间. 为了比较不同算法在不同基站(边缘服务器)部署、请求数量和请求到达地理分布度与强度情况下的性能，本文选取浦东中心商务区(CBD)、黄埔旅游区和徐汇居民区这三个场景区域分别进行试验和案例分析. 图 3 展示了浦东 CBD 场景下边缘服务器部署和边缘用户请求的地域分布，图 4 展示了这三个场景边缘请求在不同时刻的到达强度，表 1 总结了这三个实验场景的代表性特征.

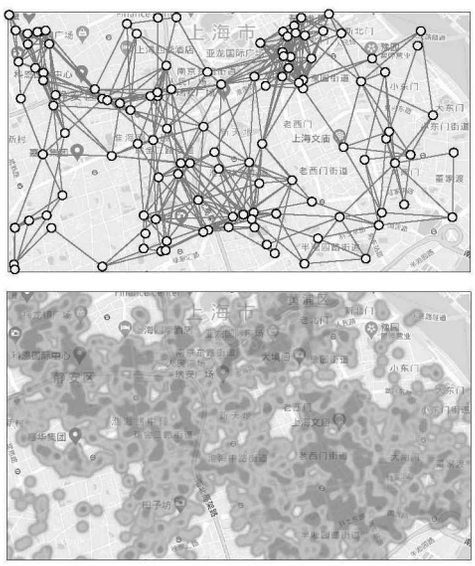

图 3　浦东 CBD 场景下的边缘服务器部署情况和边缘用户请求地域分布

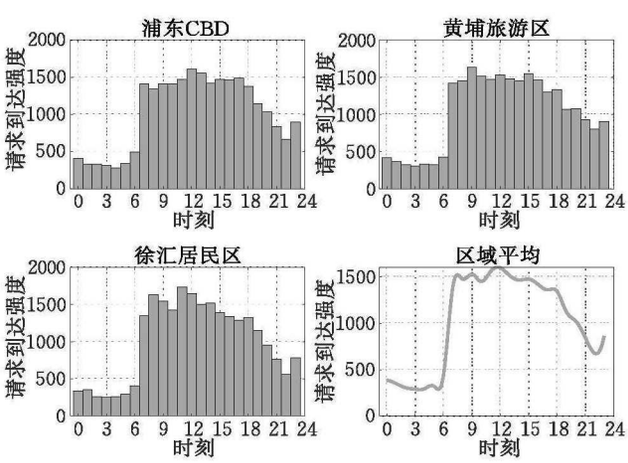

图 4　三个 MEC 场景中用户请求在一天内不同时刻的到达强度

表 1　三个实验场景的代表性特征

实验场景	服务器数量	服务器地理分布	请求变化强度	请求地理分布
浦东 CBD	116	整体密集	较缓和	局部密集
黄埔旅游区	118	局部密集	波动明显	较分散
徐汇居民区	77	分散	缓和	分散

本文考虑异构的边缘服务器和多样化的边缘任务请求，其中用户请求的任务种类 $o=10$，任务优先级 $w_j \in N^+$ 服从 1 到 5 的均匀分布，任务计算量 p_j 在 1.2 TFLOPs 与 3.6 TFLOPs 之间服从正态分布，任务上行数据量 d_j 在 0.5 Mb 与 50 Mb 之间服从正态分布，下行数据量 d'_j 在 0.01 Mb 与 5 Mb 之间服从正态分布，边缘服务器的计算能力 s_i 在 0.4 TFLOPS 和 1.2 TFLOPS 之间服从正态分布，基站信号覆盖面积为 250～500 m，基站之间的 X2 或 Xn 通信带宽 b_{ij} 在 512 Mbps 和 1 Gbps 之间服从正态分布，基站到用户设备的信道带宽 $\beta=20$ MHz，移动设备噪声功率 $\lambda^2=2\times10^{-13}$ W，移动设备信号传输功率 $\rho=0.5$ W[6]，信道增益 $\tau_{ij}=[D(e_i,r_j)]^{-\zeta}$，其中 $\zeta=4$ 为基站和用户设备之间的路径损耗因子[19]. 实验中通过控制变量法对参数 κ 进行评估，表 2 展示了当任务到达数量为 4000 时，所提出的 D-TSRA 方法在三个不同场景案例中不同参数 κ 下的得到的平均优先级加权响应时间，可以看出当 $\kappa=3$ 时 D-TSRA 方法性能最优，因此实验中将参数 κ 设置为 3.

表 2 三个场景中 D-TSRA 在不同参数 κ 下的 AWRT 值

实验场景	$\kappa=1$	$\kappa=2$	$\kappa=3$	$\kappa=4$	$\kappa=5$
浦东 CBD	55.87	57.41	**35.26**	51.17	46.78
黄埔旅游区	52.32	42.47	**33.71**	41.87	45.95
徐汇居民区	57.91	60.98	**33.87**	45.32	46.25

实验以一天内(24 h)不同场景下边缘用户任务卸载请求到达流为输入，模拟了任务到达、调度与迁移、资源分配与执行和完成离开的全过程. 实验还通过控制 24 h 内到达任务的数量(浦东 CBD 和黄埔旅游区为 4×10^3～2×10^4；徐汇居民区为 2×10^3～1×10^4)和边缘资源池的大小(即 $k\in\{1,2,3,4,5,\infty\}$，其中 $k=\infty$ 表示资源池为整个边缘资源网络)来评估不同算法在不同资源、不同负载情况下的性能表现.

实验将 BFD-WRR[52]、BFD-LZF[53]、SRA-noMG 和 OBS 四种方法作为基线来对所提出的 D-TSRA 进行评估. 其中 BFD-WRR 方法是一种中心化的调度方法，其中心调度器掌握区域内所有边缘服务器的配置信息 S_i 和当前任务队列状态 $J_i(t)$，$\forall e_i\in E$. 当用户请求 r_j 到达时，中心调度器采用最佳适应下降(Best-Fit-Decreasing，BFD)的思想，根据任务类型属性 c_j 将其调度到其适用的边缘资源池 $E_j(k)$ 中针对 c_j 任务处理速度最快的服务器上，并以严格加权轮询(WRR)策略为其分配计算资源；BFD-LZF 方法提出了任务当前满意度的概念，并采用最小满意度顺序优先(Least-Satisfaction-Index-First，LZF)方法来分配边缘资源，与 BFD-WRR 类似，BFD-LZF 算法也采用了 BFD 的思想来实时调度到达任务；为了评估本文所提出方法中任务在线迁移策略的有效性，实验中将 SRA-noMG(D-TSRA without online task migration)作为对比算法，SRA-noMG 算法采取了与所提出的 D-TSRA 方法类似的任务调度与资源分配策略，但屏蔽了实时任务迁移策略；为证明本文所提出的在线调度、迁移方法的优越性，实验中还实现了一个传统的基于批处理的离线调度算法 OBS(Optimal Batch Scheduling)作为基线算法之一. OBS 以中心化批处理的方式调度边缘任务请求，并使用整数规划求解器 Intprog 求解多任务到多服务器的调度映射方案.

5.2 优先级加权的移动任务卸载响应时间评估

图 5、图 6 和图 7 分别展示了不同算法在三个场景中不同任务数量和不同规模资源池情况下的平均优先级加权响应时间(AWRT). 整体而言，随着一天内到达任务数量的不断提高，各个算法导致的 AWRT 值均呈增长趋势，但随着资源池的不断变大，AWRT 值也呈现出明显的下降趋势. $k=\infty$(任务可以卸载至整个边缘资源网络上的任意服务器)时的 AWRT 值比 $k=1$(任务仅允许被卸载到信号可达的周围边缘服务)时小接近 2 个数量级，在所有的案例中，本文提出的 D-TSRA 方法均取得最低的 AWRT 值.

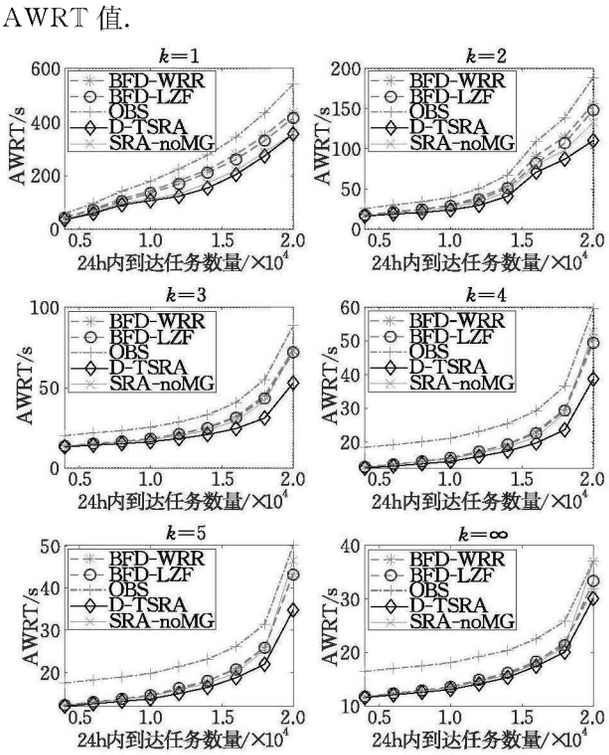

图 5 浦东 CBD 场景下 AWRT 值评估

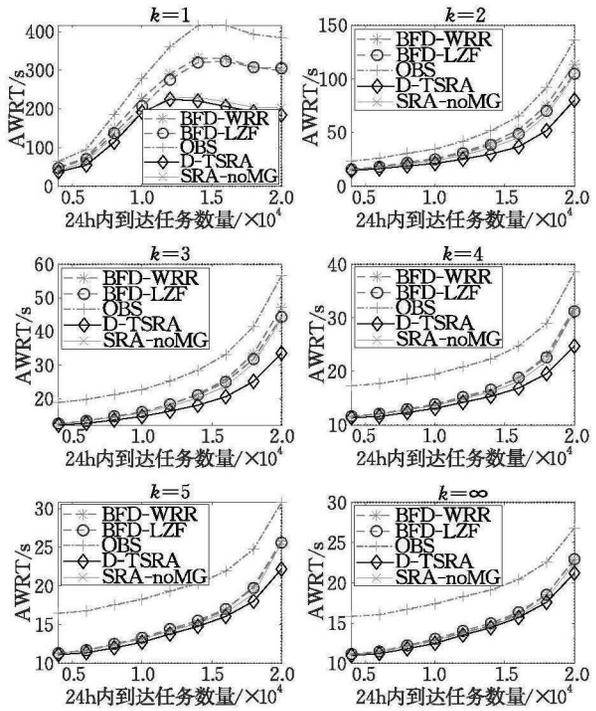

图 6　黄埔旅游区场景下 AWRT 值评估

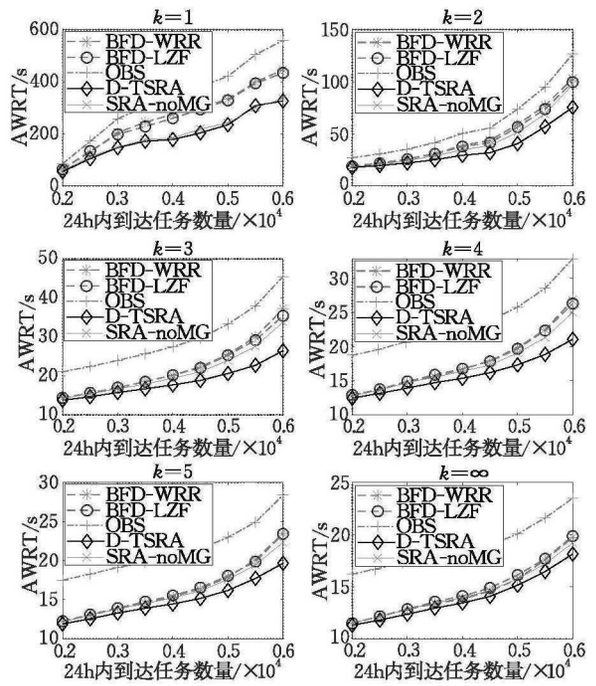

图 7　徐汇居民区场景下 AWRT 值评估

具体而言，与 BFD-WRR 算法相比 D-TSRA 方法在三个场景下 AWRT 值分别降低了 15.42%、14.57%和 13.84%；与 BFD-LZF 算法相比分别降低了 14.67%、14.21%和 14.06%；与 OBS 算法相比分别降低了 32.49%、33.71；与 SRA-noMG 算法相比分别降低了 11.93%、9.74%和 8.36%. 本文所提出的 D-TSRA 方法比 BFD-WRR 与 BFD-LZF 相比取得更低的 AWRT 值是因为其在选择调度与迁移目标与分配边缘资源时，不仅考虑了当前任务的执行速度和优先级，还将其对调入队列其他任务造成的加权时延考虑在内. 这样，D-TSRA 的调度策略在保留了一部分贪心性质的前提下可以以相互协作方式从对方的角度看待问题，从而获得优化的调度方案. 此外，还可以观察到当边缘资源池的规模越小（即 k 值越小），所提出的 D-TSRA 方法性能提升越明显. 这是因为当边缘资源紧张时，严格按照优先级分配资源的 WRR 策略会导致先到达的具有较低优先级的长作业占用过多资源，从而导致晚到的高优先级任务对最终的加权响应时间的贡献过高. 而所提出的 D-TSRA 方法中的资源调度策略会在运行时逐渐增加队列尾部具有较高优先级任务的资源分配比例，从而减轻了这些迟到高优先级任务带来的影响. OBS 方法的 AWRT 值在所有算法中最高，其主要原因在于 OBS 是一种基于批处理的离线调度算法，每隔一段时间统一处理期间到达的所有任务，触发频率由其求解速度决定. 这样任务到达后不可避免会有一段等待调度时间，该等待时间经过任务优先级杠杆放大后最终体现在较高的 AWRT 值上. D-TSRA 方法优于 SRA-noMG 方法得益于其拥有的运行时主动迁移策略. 对于一个边缘服务器来说，在运行时，其任务队列不断有新任务到达和旧任务完成离开，D-TSRA 中的在线迁移策略可以在运行时评估具有不同优先级的任务在不同的资源分配（即不同的边缘服务器上）情况下对队列中其他任务的延迟贡献，进而实现运行时迁移来达到优化平均加权响应时间的目的.

5.3　迁移开销评估

本文还对所提出 D-TSRA 方法的迁移开销进行了评估，图 8 展示了所提出的 D-TSRA 方法在浦东 CBD 场景中随着到达任务数量的增长在不同资源数目下的任务平均迁移次数和平均迁移时间. 可以看出，除了资源池规模最小（即 $k=1$）的情形，平均任务迁移次数随着总体到达任务数量的增加呈现出增长的趋势，且增长趋势逐渐变缓. 而每次迁移的平均时间开销则呈现出逐步下降的趋势，且 D-TSRA 方法的迁移花费时间对其加权响应时间相比影响较小（平均占比为 0.83%）. 当 $k\geq 2$ 时，随着到达任务数量的不断增加，任务到达事件与完成离开事件发生更加频繁，加快了边缘服务器任务队列状态的变化频率，因此会导致平均任务迁移次数的上升. 当 $k=1$ 时，由于边缘计算资源池规模较小，

可供任务迁移的选择较少，在这种情况下，当任务到达数量到达一定规模后，个别边缘节点实时处理能力小于实时到达任务的数量，造成一些边缘服务器任务队列持续增长，导致正在执行任务的降速从而降低了任务完成离开发生的频率，从而抑制了迁移行为的发生. 随着任务数的不断增加，平均迁移时间开销的降低是由平均迁移次数增长趋势下降所致. 这是因为所提出 D-TSRA 方法迁移策略中的任务效用值单调递增设定可以加速迁移稳态收敛，有效减少了迁移次数，从而降低迁移开销.

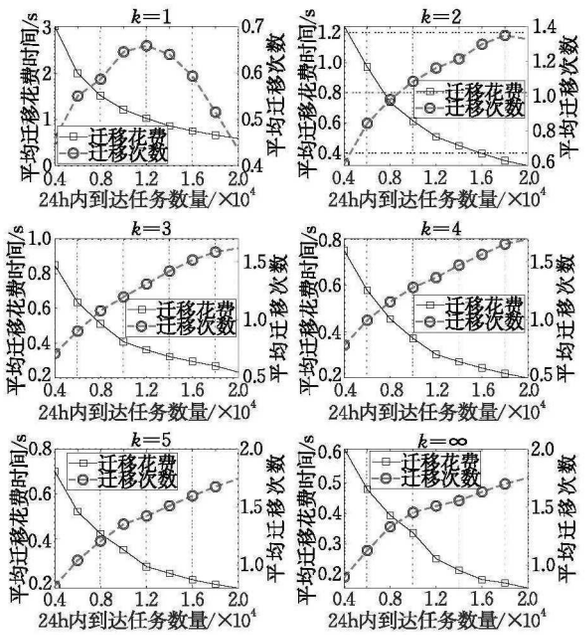

图 8 迁移开销评估

为验证所提出的迁移策略的收敛性和可控性，实验中还对所有到达任务卸载生命周期中发生的迁移次数进行了统计，图 9 展示了任务迁移次数的整体分布情况. 结果显示，平均有 45.22% 的任务在其卸载生命周期内没有发生过迁移；有超过 83.63% 的任务最多只进行了一次迁移；所有任务的最大迁移次数为 5 次，但这些任务占总任务数的比例小于 0.03%. 与 4.3 节迁移次数上界分析相结合，这些统计数据再次表明了所提出 D-TSRA 方法的任务运行时迁移策略是收敛可控的，不会因为过度迁移而带来额外的系统开销.

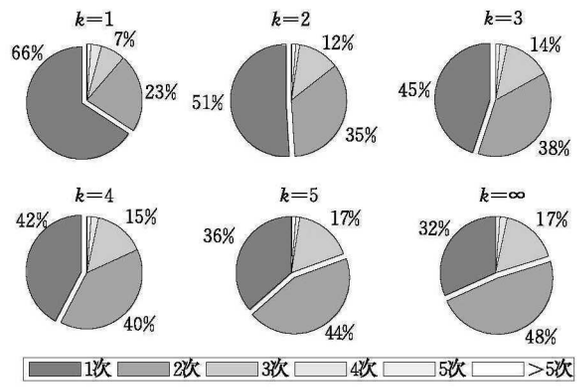

图 9 单个任务迁移次数分布

6 总结与展望

本文针对 MEC 环境下移动任务的调度与边缘资源分配问题提出了一种去中心化的在线解决方法(D-TSRA)，该方法由边缘任务实时调度策略、资源分配策略和任务迁移策略三部分构成. 与传统的中心化的调度方案不同，D-TSRA 摒弃了批处理的全局优化式的多任务同时调度与资源分配方案，将任务调度与资源分配看作为一个信息不完整的在线决策问题并提出了相应的去中心化的解决方案. D-TSRA 可以分布式的形式部署在每个边缘服务器上，每个边缘服务器只需与周围 k 跳通信距离内的服务器以异步通信的形式更新状态信息便可相互协作实现长期优化的任务卸载响应时间. 基于真实边缘环境下数据集的实验和案例分析证明了所提出算法与传统解决方案相比可以在较低的实时迁移成本下取得更加优化的长期平均加权任务响应时间.

随着边缘计算技术与相关标准不断落地与成熟，人工智能与数据协同化处理不断发展，下一步工作打算从以下几个方面开展：(1) 智能数据分析与数据挖掘技术可以被用来预测移动端用户的时空行为(如移动轨迹与请求趋势等)，并据此进行有针对性、预见性的调度、负载均衡和主动迁移，从而实现用户时空行为感知的智能调度；(2) 深度学习方法(如深度强化学习和生成对抗网络等)鲁棒性强、泛化性好的特性可以被用来训练不断演变的、可以自我纠正与更新的调度模型，产生基于知识的调度决策；(3) 未来会在系统模型中考虑更多的 QoS 指标(如任务卸载成功率、可靠性和成本等)来实现成本效益可控、目标质量高度定制化的调度决策.

参 考 文 献

[1] Wang X, IIan Y, Leung V C, et al. Convergence of edge computing and deep learning: A comprehensive survey. IEEE Communications Surveys & Tutorials, 2020, 22(2): 869-904

[2] IIu Y C, Patel M, Sabella D, et al. Mobile edge computing—A key technology towards 5G. ETSI White Paper, 2015, 11(11): 1-16

[3] Shi Wei-Song, Zhang Xing-Zhou, Wang Yi-Fan, et al. Edge computing: State-of-the-art and future directions. Journal of

Computer Research and Development, 2019, 56(1): 69-89 (in Chinese)
(施巍松, 张星洲, 王一帆等. 边缘计算：现状与展望. 计算机研究与发展, 2019, 56(1): 69-89)

[4] Deng S, Wu H, Yin J. Mobile Service Computing. Singapore: Springer, 2020

[5] Zhou Yue-Zhi, Zhang Di. Near-end cloud computing: Opportunities and challenges in the post-cloud computing era. Chinese Journal of Computers, 2019, 42(4): 677-700 (in Chinese)
(周悦芝, 张迪. 近端云计算：后云计算时代的机遇与挑战. 计算机学报, 2019, 42(4): 677-700)

[6] Chen M, Hao Y. Task offloading for mobile edge computing in software defined ultra-dense network. IEEE Journal on Selected Areas in Communications, 2018, 36(3): 587-597

[7] Xia Q, Liang W, Xu Z, et al. Online algorithms for location-aware task offloading in two-tiered mobile cloud environments //Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing. London, UK, 2014: 109-116

[8] Im S, Kulkarni J, Munagala K, et al. SelfishMigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors//Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science. Philadelphia, USA, 2014: 531-540

[9] Li W, Liewig M. A survey of AI accelerators for edge environment//Proceedings of the World Conference on Information Systems and Technologies. Budva, Montenegro, 2020: 35-44

[10] Véstias M. Convergence of Artificial Intelligence and the Internet of Things. Switzerland: Springer, 2020

[11] Chen J, Ran X. Deep learning with edge computing: A review. Proceedings of the IEEE, 2019, 107(8): 1655-1674

[12] Dai Y, Xu D, Maharjan S, et al. Joint computation offloading and user association in multi-task mobile edge computing. IEEE Transactions on Vehicular Technology, 2018, 67(12): 12313-12325

[13] Yang B, Cao X, Bassey J, et al. Computation offloading in multi-access edge computing: A multi-task learning approach. IEEE Transactions on Mobile Computing, 2020, 20(9): 2745-2762

[14] Rafique W, Qi L, Yaqoob I, et al. Complementing IoT services through software defined networking and edge computing: A comprehensive survey. IEEE Communications Surveys & Tutorials, 2020, 22(3): 1761-1804

[15] Peng Q, Xia Y, Zhou M, et al. Reliability-aware and deadline constrained mobile service composition over opportunistic networks. IEEE Transactions on Automation Science and Engineering, 2021, 18(3): 1012-1025

[16] Guo S, Dai Y, Guo S, et al. Blockchain meets edge computing: Stackelberg game and double auction based task offloading for mobile blockchain. IEEE Transactions on Vehicular Technology, 2020, 69(5): 5549-5561

[17] Liu B, Cao Y, Zhang Y, et al. A distributed framework for task offloading in edge computing networks of arbitrary topology. IEEE Transactions on Wireless Communications, 2020, 19(4): 2855-2867

[18] Alameddine H A, Sharafeddine S, Sebbah S, et al. Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. IEEE Journal on Selected Areas in Communications, 2019, 37(3): 668-682

[19] Yang L, Zhang H, Li X, et al. A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing. IEEE/ACM Transactions on Networking, 2018, 26(6): 2762-2773

[20] Liu C F, Bennis M, Debbah M, et al. Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. IEEE Transactions on Communications, 2019, 67(6): 4132-4150

[21] Du W, Lei T, He Q, et al. Service capacity enhanced task offloading and resource allocation in multi-server edge computing environment//Proceedings of the 2019 IEEE International Conference on Web Services. Milan, Italy, 2019: 83-90

[22] Zhao H, Deng S, Zhang C, et al. A mobility-aware cross-edge computation offloading framework for partitionable applications//2019 IEEE International Conference on Web Services. Milan, Italy, 2019: 193-200

[23] Chen L, Zhou S, Xu J. Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. IEEE/ACM Transactions on Networking, 2018, 26(4): 1619-1632

[24] Xu J, Chen L, Zhou P. Joint service caching and task offloading for mobile edge computing in dense networks// Proceedings of the IEEE INFOCOM 2018 IEEE Conference on Computer Communications. Honolulu, USA, 2018: 207-215

[25] Huang L, Bi S, Zhang Y J. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Transactions on Mobile Computing, 2019, 19(11): 2581-2593

[26] Liu Wei, Huang Yu-Cheng, Du Wei, et al. Resource-constrained serial task offload strategy in mobile edge computing. Journal of Software, 2020, 31(6): 1889-1908(in Chinese)
(刘伟, 黄宇成, 杜薇等. 移动边缘计算中资源受限的串行任务卸载策略. 软件学报, 2020, 31(6): 1889-1908)

[27] He Q, Cui G, Zhang X, et al. A game-theoretical approach for user allocation in edge computing environment. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(3): 515-529

[28] Lin X, Li J, Baldemair R, et al. 5G new radio: Unveiling the essentials of the next generation wireless access technology. IEEE Communications Standards Magazine, 2019, 3(3): 30-37

[29] Love R. Linux Kernel Development. London: Pearson Education, 2010

[30] Qin H, Li Q, Speiser J, et al. Arachne: Core-aware thread management//Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad, USA, 2018: 145-160

[31] Zhao H, Deng S, Liu Z, et al. Distributed redundancy scheduling for microservice-based applications at the edge. IEEE Transactions on Services Computing, 2020, 32(8): 1918-1932

[32] Wu C, Peng Q, Xia Y, et al. Online user allocation in mobile edge computing environments: A decentralized reactive approach. Journal of Systems Architecture, 2021, 113: 101904

[33] Peng Q, Xia Y, Wang Y, et al. Joint operator scaling and placement for distributed stream processing applications in edge computing//Proceedings of the International Conference on Service-Oriented Computing. Toulouse, France, 2019: 461-476

[34] Feldman J, Henzinger M, Korula N, et al. Online stochastic packing applied to display ad allocation//Proceedings of the European Symposium on Algorithms. Liverpool, UK, 2010: 182-194

[35] Agrawal S, Wang Z, Ye Y. A dynamic near-optimal algorithm for online linear programming. Operations Research, 2014, 62(4): 876-890

[36] Al-Habob A A, Dobre O A, Armada A G, et al. Task scheduling for mobile edge computing using genetic algorithm and conflict graphs. IEEE Transactions on Vehicular Technology, 2020, 69(8): 8805-8819

[37] Chen J, Li K, Deng Q, et al. Distributed deep learning model for intelligent video surveillance systems with edge computing. IEEE Transactions on Industrial Informatics, 2019, 31(4): 948-966

[38] Setlur A R, Nirmala S J, Singh H S, et al. An efficient fault tolerant workflow scheduling approach using replication heuristics and check pointing in the cloud. Journal of Parallel and Distributed Computing, 2020, 136: 14-28

[39] Behera S, Wan L, Mueller F, et al. Orchestrating fault prediction with live migration and checkpointing//Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing. Stockholm, Sweden, 2020: 167-171

[40] Rodríguez-Pascual M, Cao J, Moríñigo J A, et al. Job migration in HPC clusters by means of checkpoint/restart. The Journal of Supercomputing, 2019, 75(10): 6517-6541

[41] Lösch A, Beisel T, Kenter T, et al. Performance-centric scheduling with task migration for a heterogeneous compute node in the data center//Proceedings of the 2016 Conference on Design, Automation & Test in Europe Conference & Exhibition. Dresden, Germany, 2016: 912-917

[42] Monderer D, Shapley L S. Potential games. Games and Economic Behavior, 1996, 14(1): 124-143

[43] Marden J R, Arslan G, Shamma J S. Joint strategy fictitious play with inertia for potential games. IEEE Transactions on Automatic Control, 2009, 54(2): 208-220

[44] Narahari Y. Game Theory and Mechanism Design. Singapore: World Scientific, 2014

[45] Tao Ji-Ping, Xi Yu-Geng. A novel way to analyze competitive performance of online algorithms — Instance transformation based method. Journal of Systems Science and Mathematical Sciences, 2009, (10): 1381-1389(in Chinese)
(陶继平, 席裕庚. 一种新的在线调度算法竞争比分析方法—基于实例转换的方法. 系统科学与数学, 2009, (10): 1381-1389)

[46] Garg N, Kumar A. Minimizing average flow-time: Upper and lower bounds//Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science. Providence, USA, 2007: 603-613

[47] Chadha J S, Garg N, Kumar A, et al. A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation//Proceedings of the 41st Annual ACM Symposium on Theory of Computing. Bethesda, USA, 2009: 679-684

[48] Wang S, Zhao Y, Huang L, et al. QoS prediction for service recommendations in mobile edge computing. Journal of Parallel and Distributed Computing, 2019, 127: 134-144

[49] Wang S, Zhao Y, Xu J, et al. Edge server placement in mobile edge computing. Journal of Parallel and Distributed Computing, 2019, 127: 160-168

[50] Wang S, Guo Y, Zhang N, et al. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. IEEE Transactions on Mobile Computing, 2021, 20(3): 939-951

[51] Guo Y, Wang S, Zhou A, et al. User allocation-aware edge cloud placement in mobile edge computing. Software: Practice and Experience, 2020, 50(5): 489-502

[52] Peng Q, Xia Y, Feng Z, et al. Mobility-aware and migration-enabled online edge user allocation in mobile edge computing//Proceedings of the 2019 IEEE International Conference on Web Services. Milan, Italy, 2019: 91-98

[53] Jang J, Jung J, Hong J. K-LZF: An efficient and fair scheduling for edge computing servers. Future Generation Computer Systems, 2019, 98: 44-53

附 录. 引理 2 的证明.

证明. 由 $\Delta_j^1(t^*)$及 $v_j(t)$的定义可知:

$$\Delta_j^1(t^*)=\frac{1}{\eta}\int_{t-r_j}^{t^*}\left[v_j(t)(\mathcal{W}_j(t)+w_j)+w_j\sum_{r_{j'}\in\mathcal{J}_j(t)}v_{j'}(t)\right]\mathrm{d}t$$

$$\leq\frac{1}{\eta}\int_{t-r_j}^{t^*}\left[\eta\cdot\frac{(\mathcal{W}_j(t)+w_j)^{\kappa+1}-\mathcal{W}_j(t)^{\kappa+1}}{W_{\delta(j,t)}(t)^{\kappa+1}}\cdot[\mathcal{W}_j(t)+w_j]+w_j\cdot\eta\right]\mathrm{d}t,$$

令 $\sigma(j,t)$表示 t 时刻任务 r_j被调度到的边缘服务器,又因为

$W_{\sigma(j,i)}(t)\geq\mathcal{W}_j(t)+w_j$，因此有

$$\Delta_j^1(t^*)\leq\int_{t-r_j}^{t^*}\left[\left(1-\left(1-\frac{w_j}{\mathcal{W}_j(t)+w_j}\right)^{\kappa+1}\right)\cdot(\mathcal{W}_j(t)-w_j)+w_j\right]\mathrm{d}t,$$

又因为对于任意 $\kappa\geq0$，$\kappa\in N$ 和 $0\leq w\leq1$ 有 $(1-w)^\kappa\geq1-\kappa w$，因此：

$$\Delta_j^1(t^*)\leq\int_{t-r_j}^{t^*}\left[\frac{w_j\cdot(\kappa+1)}{\mathcal{W}_j(t)+w_j}\cdot(\mathcal{W}_j(t)+w_j)+w_j\right]\mathrm{d}t=(\kappa+2)\cdot w_j\cdot(t^*-r_j).$$

由 $\Delta_j^2(t^*)$ 的定义可知：

$$\Delta_j^2(t^*)=\frac{1}{\eta}\int_{t-t^*}^{C_j}\left[v_j(t)\cdot(\mathcal{W}_j(t)+w_j)+w_j\cdot\left(\sum_{r_{j'}\in\mathcal{J}_j(t)}v_{j'}(t)\right)\right]\mathrm{d}t=\frac{1}{\eta}\int_{t-t^*}^{C_j}\left[v_j(t)\cdot(\mathcal{W}_j(t)+w_j)+\eta\cdot w_j\cdot\frac{\mathcal{W}_j(t)^{\kappa+1}}{W_{\delta(j,i)}(t)^{\kappa+1}}\right]\mathrm{d}t$$

$$=\frac{1}{\eta}\int_{t-t^*}^{C_j}v_j(t)\cdot\left[\mathcal{W}_j(t)+w_j\cdot\left(\frac{[\mathcal{W}_j(t)+w_j]^{\kappa+1}}{[\mathcal{W}_j(t)+w_j]^{\kappa+1}-\mathcal{W}_j(t)^{\kappa+1}}L\right)\right]\mathrm{d}t,$$

因为对于任意 $\kappa\in N$ 和 $w\geq0$，$w'\geq0$ 有 $(w+w')^\kappa\geq w^\kappa+\kappa w^{\kappa-1}w'$，所以有

$$\Delta_j^2(t^*)\leq\frac{1}{\eta}\int_{t-t^*}^{C_j}v_j(t)\cdot\left[\mathcal{W}_j(t)+w_j+\frac{\mathcal{W}_j(t)+w_j}{\kappa+1}\right]\mathrm{d}t=\frac{1}{\eta}\cdot\frac{\kappa+2}{\kappa+1}\cdot\int_{t-t^*}^{C_j}l_{\delta(j,i)j}\cdot v_j(t)\cdot\frac{1}{\phi(j,t)}\mathrm{d}t,$$

由引理 2 可知：

$$\Delta_j^2(t^*)\leq\frac{1}{\eta}\cdot\frac{\kappa+2}{\kappa+1}\cdot\frac{1}{\phi(j,t^*)}\cdot\int_{t-t^*}^{C_j}l_{\delta(j,i)j}\cdot v_j(t)\mathrm{d}t=\frac{1}{\eta}\cdot\frac{\kappa+2}{\kappa+1}\cdot\frac{p_j(t^*)}{\phi(j,t^*)}\leq\frac{1}{\eta}\cdot\frac{\kappa+2}{\kappa+1}\cdot\frac{\mathcal{W}_j(t^*)+w_j}{l_{i^*j}}.$$

证毕.

PENG Qing-Lan, Ph.D. candidate. His research interests include cloud computing, edge computing, and service computing.

XIA Yun-Ni, Ph.D., professor. His research interests include service computing, cloud computing, edge computing, and stochastic Petri net.

ZHENG Wan-Bo, Ph.D., associate professor. His research interests include cloud computing, disaster and emergency management, trustworthy software.

WU Chun-Rong, Ph.D. candidate. Her research interests include data mining and service computing.

PANG Shan-Chen, Ph.D., professor. His research interests include trustworthy computing, service computing, cloud computing, and Petri net.

LONG Mei, M.S., senior engineer. Her research interests include crowdsourcing system and service computing.

JIANG Ning, M.S. His research interests include service computing and cloud computing.

Background

Mobile task offloading aims at executing the computation-intensive and latency-sensitive mobile tasks with the help of external resources (e.g., remote cloud, mobile cloud, and edge cloud). As a promising computing paradigm, Multi-access Edge Computing (MEC) has the ability to provide low-latency computing resources and high-responsiveness services by deploying edge servers at the Internet access level (e.g., base station, smart gateway, and wireless access point). Nevertheless, different from the cloud-computing paradigm that has infinite resources, MEC is restricted by limited computing capabilities and restricted wireless communication distance, which calls for appropriate task scheduling algorithms and smart resource provision strategies.

Most of the current joint task offloading and resource allocation works on MEC environments are based on a centralized architecture, and they usually handle the arrived offloading requests in a batch-processing way. However, the centralized architecture might be susceptible to single-point-of-failure, and the batch-processing mode is unfriendly to delay-sensitive mobile applications (e.g., real-time augmented reality, virtual reality, and automatic drive).

In this paper, we propose a decentralized approach towards the online edge task scheduling and resource allocation problem. The proposed approach consists of an online task scheduling strategy, an edge resource allocation strategy, and an online task migration strategy. The main feature of the proposed approach is that it can be deployed on every edge servers and yield real-time scheduling, migration, and resource provision decisions collaboratively with each other. Besides, to capture the diversity of different mobile applications and user groups, we also consider the priority of mobile tasks in our system model and take the average weighted offloading response time as the scheduling target. Experimental results have demonstrated the effectiveness and efficiency of the proposed decentralized online approach.

This work is in part supported by the National Natural Science Foundation of China under Grant No. 62172062 and No. 62162036, and the Chongqing Key Research and Development Project under Grant No. cstc2019jscx-xyd0385.