DOI: 10.13382/j.jemi.B2104715

移动边缘计算中依赖型任务的调度模型研究*

王 瑶 卢先领 沈义峰
(江南大学物联网工程学院 无锡 214122)

摘 要:当前移动边缘计算(mobile edge computing, MEC)环境中关于任务调度的工作经常忽略任务间的依赖关系,导致其完成时延较长。针对此问题,首先,以降低系统完成时延为目标,在考虑到跨服务器协作的多用户、多边缘服务器场景下,利用广度优先搜索算法(breadth first search, BFS)构建一种依赖型任务的调度模型。然后,根据任务和边缘服务器之间的交互,将模型中各调度层的联合卸载和迁移问题建模为一个多领导者多跟随者的 Stackelberg 博弈。最后,为实现 Stackelberg 博弈均衡,提出基于 Q 值的卸载算法和分布式迭代迁移算法求解模型。仿真结果表明,与基线算法相比,所提算法在不同规模的用户和边缘服务器的场景下,将系统完成时延分别降低了 44.1%和 63.2%。进一步实验表明,与传统方案相比,所提模型在不同规模的用户和边缘服务器的场景下使系统完成时延分别降低了 20.1%和 6.7%,有效保证了服务质量。

关键词: 移动边缘计算;任务调度;依赖型任务;Stackelberg 博弈;Q 值
中图分类号: TP393 **文献标识码**: A **国家标准学科分类代码**: 520

Research on scheduling model of dependent tasks in mobile edge computing

Wang Yao Lu Xianling Shen Yifeng
(School of Internet of Things Engineering, Jiangnan University, Wuxi 214122, China)

Abstract: The task scheduling work in the current mobile edge computing (MEC) environment often ignores the dependency between tasks, resulting in a long delay in completion. In response to this problem, first of all, with the goal of reducing the system completion delay, in the multi-user and multi-edge server scenario that takes cross-server collaboration into account, the breadth first search algorithm (BFS) is used to build a dependent task scheduling model. Then, according to the interaction between tasks and edge servers, the joint offloading and migration problem of each scheduling layer in the model are modeled as a Stackelberg game with multiple leaders and multiple followers. Finally, in order to achieve Stackelberg equilibrium, an offloading algorithm based on the Q value and a distributed iterative migration algorithm are proposed to solve the model. The simulation results show that compared with the baseline algorithms, the proposed algorithm reduces the system completion delay by 44.1% and 63.2% respectively in the scenarios of users and edge servers with different scales. Further experiments show that compared with the traditional solutions, the proposed model reduces the system completion delay by 20.1% and 6.7% respectively in the scenarios of users and edge servers with different scales, and effectively guarantees the quality of service.

Keywords: mobile edge computing; task scheduling; dependent tasks; Stackelberg game; Q value

0 引 言

5G 时代的到来和互联网设备的广泛使用推动了人脸识别和自动驾驶等时延敏感型和资源密集型应用程序的发展[1]。目前虽然可以依靠云平台处理这类应用程序[2],但一方面呈指数性增长[3]的用户设备会受到资源的限制;另一方面设备与云服务器之间长距离的传输也可能会产生不可预测的通信延迟[4]。因此,移动边缘计算应运而生[5]。MEC 网络中,具有更多计算资源的边缘

收稿日期: 2021-09-08 Received Date: 2021-09-08
*基金项目:国家自然科学基金项目(61773181)资助

节点被部署到更靠近用户的位置上,为更有效地利用边缘节点,如何制定更加合理的调度方案受到了学术界的广泛关注[6]。

目前国内外学者对 MEC 中任务调度方案的研究主要针对于独立任务[7-8]或依赖型任务[9-19]的调度问题。文献[7-8]分别通过基于潜在博弈的卸载算法和分支定界算法获得了独立任务的调度策略。然而,实际应用程序中的任务大多都高度关联,例如,VR 类型的应用程序可分为 5 个相关任务:收集数据、渲染图像帧、压缩数据、传输和显示[9]。当处理这些任务时,必须考虑任务之间存在的依赖关系[10-11]。针对由依赖型任务组成的应用程序调度问题,文献[12]提出了一种启发式算法,以在应用完成截止日期的约束下,最小化应用程序的执行成本。文献[13]考虑到边缘节点缓存对任务调度的影响,设计了一种基于凸规划的算法。文献[14]研究了双用户之间任务的依赖性,采用吉布斯采样算法获得了最优决策。但上述文献只考虑了单用户或者单服务器场景。文献[15]解决了异构 MEC 环境下细粒度任务的调度问题,为多用户提出了一种轻量级的调度方案。文献[16]考虑到任务调度和服务缓存放置问题,提出了一种近似算法以最小化应用程序的完成时延。但上述文献都假设边缘服务器拥有无限资源,忽略了资源争用对任务调度性能的影响。文献[17]考虑到用户之间的资源竞争,设计了一种启发式迭代算法。文献[18]提出了一种面向多用户的串行任务动态调度策略,降低了任务完成时延和终端能耗。但上述文献只考虑到串行依赖任务,实际应用程序中还包含并行任务,因此所研究的任务关系不具有普适性和复杂性。文献[19]联合优化了服务器分配和资源管理,对来自多个用户的应用程序提出了一种有效的调度算法,减少了系统完成时延。但该文献未充分利用任务间的并行关系,要求边缘服务器为系统中的所有任务同时分配并预留资源,导致部分边缘服务器长时间处于空闲状态,造成资源浪费。上述所有文献均未利用到边缘服务器之间的跨服务器协作。一方面,过重的负载使得 MEC 上有限的资源成为瓶颈;另一方面,负载的不均衡导致部分资源被闲置。

针对以上研究存在的不足,首先在资源有限的多用户、多边缘服务器的 MEC 环境下,构建了一种依赖型任务调度模型。该模型为充分利用任务之间的并行关系,利用 BFS 算法设计了调度列表;此外,模型中还考虑到跨服务器协作,以实现负载均衡。然后将降低系统完成时延问题转化为面向模型中各调度层的联合卸载和迁移问题,并引入一个多领导者多跟随者的 Stackelberg 博弈对联合问题进行建模。最后提出了一种基于 Q 值的卸载算法和分布式迭代迁移算法。仿真结果表明在多用户和多边缘服务器环境下,本文模型和算法都能够有效降低系统完成时延。

1　任务调度模型

本文考虑了一个典型的 MEC 系统,如图 1 所示,定义边缘服务器(edge server, ES)集合为 $\mathcal{N}=\{1,2,\cdots,N\}$,用户集合为 $\mathcal{M}=\{1,2,\cdots,M\}$,每个 ES 都配有一个接入点 AP(access point),用户上的任务可以选择指定的 AP 接入,再传输给相应的 ES 处理,所有 AP 通过光纤链路互连。为便于说明,下文符号"ES"代指 AP 和 ES。

本文假设系统中每个用户一次请求单个应用程序。用户 $m\in\mathcal{M}$ 的应用程序可以用有向无环图 $G_m=\{V_m, ED_m\}$ 表示。其中 $V_m=\{v_{m,i}\mid 1\leqslant i\leqslant |V_m|\}$ 为用户 m 应用程序的任务集合,$|V_m|$ 为总的任务数量,$ED_m=\{(v_{m,i},v_{m,j})\mid v_{m,i},v_{m,j}\in V_m\wedge i\neq j\}$ 为有向边集合。若 $(v_{m,i},v_{m,i+1})\in ED_m$,则表明任务 $v_{m,i}$ 和 $v_{m,i+1}$ 之间存在依赖关系,$v_{m,i+1}$ 需要等待 $v_{m,i}$ 执行完成才能开始执行。

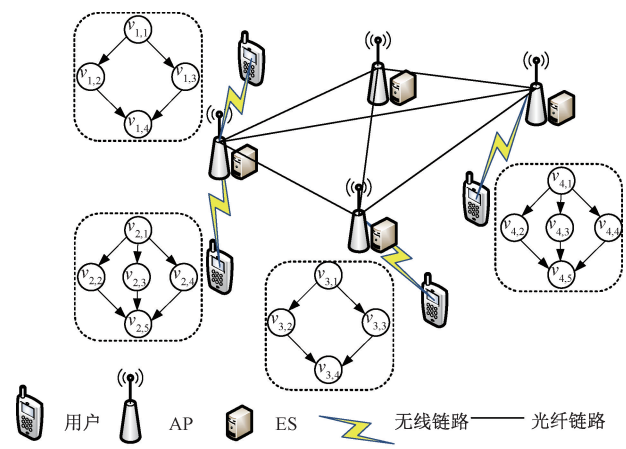
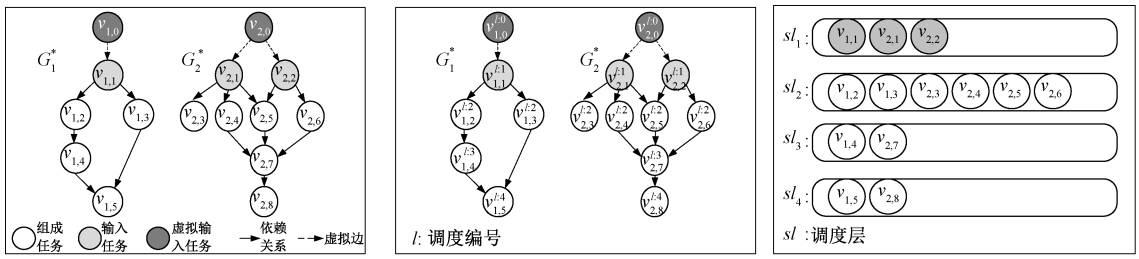


图 1　MEC 系统

Fig. 1　MEC system

1.1　调度列表构建

为保证依赖型任务正确的执行顺序,同时最大化系统中可并行执行的任务数量,本文利用 BFS 算法构建了调度列表。调度列表的构建过程如下:首先,创建虚拟输入任务节点 $v_{m,0}$(执行时延为 0),并将 $v_{m,0}$ 与 G_m 中的原输入任务节点连接,形成新的 DAG 图 G_m^*。如图 2(a)所示,为 G_1 和 G_2 创建 $v_{1,0}$ 和 $v_{2,0}$,并将 $v_{1,0}$ 与 $v_{1,1}$ 连接得到 G_1^*,将 $v_{2,0}$ 与 $v_{2,1}$ 和 $v_{2,2}$ 连接得到 G_2^*。然后,用 BFS 算法遍历每个 G_m^*,为各任务分配调度编号 l。如图 2(b)所示,分别遍历 G_1^* 和 G_2^*,分配 l,从而指定 G_1^* 和 G_2^* 中具有并行关系的任务。最后,将具有相同调度编号的任务归于同一调度层(虚拟输入任务除外),得到调度列表 $SL=\{sl_l\mid 1\leqslant l\leqslant |SL|\}$,其中 $|SL|$ 为总调度层数。如图 2(c)所示,根据 l 合并 G_1^* 和 G_2^*。



(a)创建新DAG (a) Create new DAG　(b)分配调度编号 (b)Assign dispatch number　(c)形成调度表 (c)Form a schedule

图 2　调度列表构建过程

Fig. 2　Schedule list construction process

由于同一调度层上任务之间的执行顺序互不影响，所以可以将同一层上的任务当作独立任务并行处理，下面对调度层上的任务具体分析。

1.2　任务完成时延建模

将调度层 sl_l 上的任务集合表示为 $U_l = \{v_{m,i} \mid v_{m,i} \in \cup_{m \in \mathcal{M}_l} V_m^l\}$，其中 $\mathcal{M}_l \subseteq \mathcal{M}$ 为 sl_l 上的用户集合，$V_m^l \subseteq V_m$ 为 sl_l 上用户 m 应用程序中的任务集合。任务 $v_{m,i}$ 可以用 $\{w_{m,i}, c_{m,i}\}$ 表示，其中 $w_{m,i}$ 为输入数据量，$c_{m,i}$ 为其计算所需的 CPU 周期数。本文对任务的调度包含卸载决策和迁移决策。

由于任务既可以在本地执行也可以在 ES 执行，所以令 $\boldsymbol{X}_l = \{x_{m,i}, \forall v_{m,i} \in U_l\}$ 为调度层 sl_l 上所有任务的卸载决策，其中 $x_{m,i} \in \mathcal{X}_{m,i}$ 且 $\mathcal{X}_{m,i} = \{0\} \cup \mathcal{N}_{m,i}$，$\mathcal{N}_{m,i} \subseteq \mathcal{N}$ 为任务 $v_{m,i}$ 可以选择的 ES 集合。当 $x_{m,i} = 0$ 时，表示任务 $v_{m,i}$ 在本地执行，当 $x_{m,i} \in \mathcal{N}_{m,i}$ 时，表示任务 $v_{m,i}$ 卸载执行（注意任务一次不能选择多个 ES）。给定卸载决策 $\boldsymbol{X}_l$，将卸载至 ESn 执行的任务集合表示为 $U_n^F = \{v_{m,i} \mid x_{m,i} = n\} \subseteq U_l$。

当 ES 接收到调度层 sl_l 上的任务 $v_{m,i}$ 后，考虑到跨服务器协作，它可以选择在自身执行，也可以通过有线连接将 $v_{m,i}$ 迁移至其他 ES 执行。用 $\boldsymbol{S}_l = \{\boldsymbol{S}_n, \forall n \in \mathcal{N}\}$ 表示处理调度层 sl_l 时，所有 ES 的迁移决策。其中 $\boldsymbol{S}_n = \{s_{m,i}^n, \forall v_{m,i} \in U_n^F\}$ 为 ESn 的迁移决策，$s_{m,i}^n \in \mathcal{N}$ 为 ESn 处理任务 $v_{m,i}$ 时的迁移决策，存在 $\boldsymbol{S}_n \in \times_{v_{m,i} \in U_n^F} \mathcal{N} \triangleq \mathbb{S}_n$，$\boldsymbol{S}_l \in \times_{n \in \mathcal{N}} \mathbb{S}_n \triangleq \mathbb{S}$。此外，将 $U_{d \to n} = \{v_{m,i} \mid x_{m,i} = d \wedge s_{m,i}^d = n\} \subseteq U_l$ 定义为从 ESd 迁移到 ESn 上的任务集合，因此，最终在 ESn 上执行的任务集合可以表示为 $U_n^E = \cup_{d \in \mathcal{N}} U_{d \to n}$。

1)通信模型

用户和 ES 之间存在无线通信，根据香农公式，用户 m 到 ESn 之间的数据传输速率可以表示为：

$$R_{m,n} = W_n \log_2\left(1 + \frac{p_m G_{m,n}}{\sigma^2}\right) \tag{1}$$

式中：W_n 为 ESn 的信道带宽，p_m 为用户 m 的传输功率，$G_{m,n}$ 为用户 m 到 ESn 之间的信道增益，σ^2 为高斯白噪声功率。当调度层 sl_l 上有多个任务同时卸载到 ESn 上时，所有决定卸载到 ESn 上的任务都必须共享 ESn 拥有的信道带宽。为达到最佳的任务处理效率，ESn 分配给任务 $v_{m,i}$ 的最佳无线资源百分比[20] 为 $\alpha_{m,i,n} = \sqrt{w_{m,i}/R_{m,n}} / \sum_{v_{m',i'} \in U_n^F} \sqrt{w_{m',i'}/R_{m',n}}$，此时，用户 m 上任务 $v_{m,i}$ 到 ESn 的数据传输速率表示为 $r_{m,i,n} = \alpha_{m,i,n} R_{m,n}$。

此外，用 $R_{n \to d}$ 表示 ESn 到 ESd 的最大数据传输速率，如果 ESn 决定将其上的任务 $v_{m,i}$ 进一步迁移至 ESd，由于资源竞争的影响，任务 $v_{m,i}$ 从 ESn 到 ESd 的数据传输速率同样可以表示为 $r_{m,i,n \to d} = \beta_{m,i,n \to d} R_{n \to d}$，其中任务 $v_{m,i}$ 所占 ESn 到 ESd 的最佳有线带宽百分比为 $\beta_{m,i,n \to d} = \sqrt{w_{m,i}/R_{n \to d}} / \sum_{v_{m',i'} \in U_{n \to d}} \sqrt{w_{m',i'}/R_{n \to d}}$。

2)计算模型

(1)本地计算

当任务 $v_{m,i}$ 决定在本地执行时，任务的完成时延由其计算所需的周期数和用户终端 m 的计算能力 F_m^L 决定，表示为：

$$T_{m,i}^L = \frac{c_{m,i}}{F_m^L} \tag{2}$$

(2)边缘计算

根据通信模型，当任务 $v_{m,i}$ 决定卸载至 ESn 执行时，传输时延可表示为 $T_{m,i,n}^{tra} = w_{m,i}/r_{m,i,n}$。另外，如果 ES$n$ 决定将任务 $v_{m,i}$ 进一步迁移至 ESd，则将 $v_{m,i}$ 的输入数据从 ESn 传输至 ESd 也会产生额外的迁移时延，表示为 $T_{m,i,n \to d}^{mig} = w_{m,i}/r_{m,i,n \to d}$。

所有由 ESn 执行的任务都需要共享 ESn 的计算资源，定义 ESn 分配给任务 $v_{m,i}$ 的计算资源为 $f_{m,i,n}^E = \xi_{m,i,n} F_n^E$，其中 $\xi_{m,i,n} = \sqrt{c_{m,i}/F_n^E} / \sum_{v_{m',i'} \in U_n^E} \sqrt{c_{m',i'}/F_n^E}$ 为分配资源所占的最佳百分比[20]，F_n^E 为 ESn 计算资源的上限。如果任务 $v_{m,i}$ 在 ESn 执行，计算时延为 $T_{m,i,n}^{exe} = c_{m,i}/f_{m,i,n}^E$。

因此,当任务 $v_{m,i}$ 卸载至 ESn 执行时, $v_{m,i}$ 的完成时延可以表示为:

$$T^{E}_{m,i,n} = T^{tra}_{m,i,n} + \sum_{d\in\mathcal{N}} I_{\{s^{n}_{m,i}=d\}}\left(T^{mig}_{m,i,n\to d} + T^{exe}_{m,i,d}\right) \tag{3}$$

式中: $I_{\{\times\}}$ 为指示函数,当满足条件×时,其值为 1,否则为 0。特别的,当 $d=n$ 时,表明 ESn 不迁移任务 $v_{m,i}$,此时 $T^{mig}_{m,i,n\to d}=0$。

1.3　系统完成时延建模

为确保任务的执行顺序满足依赖关系,同时使每一层上的任务拥有尽可能多的计算和通信资源以缓解资源竞争,模型设定了一种调度层处理方案:根据调度列表依次处理各调度层,只有处理完上一调度层中的所有任务时,下一调度层任务才能开始被处理。据此,调度层 sl_l 的完成时延可以表示为:

$$T_l = \max_{v_{m,i}\in U_l}\left(T_{m,i}\right) \tag{4}$$

式中: $T_{m,i} = I_{\{x_{m,i}=0\}}T^{L}_{m,i} + \sum_{n\in\mathcal{N}_{m,i}} I_{\{x_{m,i}=n\}}T^{E}_{m,i,n}$ 为任务 $v_{m,i}$ 的完成时延。系统完成时延为处理完所有调度层所需的时延,表示为:

$$T = \sum_{l=1}^{|SL|} T_l \tag{5}$$

2　问题构建

2.1　代价函数

本文旨在利用第 1 节中构建的任务调度模型,为系统中各任务找到合适的执行位置以降低系统完成时延。根据式(4)和(5),可以通过降低每个任务的完成时延来降低系统完成时延。据此,当处理调度层 sl_l 时,定义任务 $v_{m,i}$ 的代价函数为:

$$C_{m,i}(\boldsymbol{X}_l,\boldsymbol{S}_l) = I_{\{x_{m,i}=0\}}T^{L}_{m,i} + \sum_{n\in\mathcal{N}_{m,i}} I_{\{x_{m,i}=n\}}T^{E}_{m,i,n} \tag{6}$$

在本文设定的调度层处理方案中,当处理调度层 sl_l 时,如果负载不均衡,具有较重负载的 ES 将会延长下一调度层 sl_{l+1} 开始处理的时间,从而影响整个系统的完成时延。因此,在调度层上的任务卸载后,每个 ES 都应该通过自身执行或者将任务迁移到其他 ES 来尽快完成卸载至它的所有任务,从而实现负载均衡。据此,定义 ESn 的代价函数为:

$$C_n(\boldsymbol{X}_l,\boldsymbol{S}_l) = \sum_{v_{m,i}\in U^{F}_{n}} C_{m,i}(\boldsymbol{X}_l,\boldsymbol{S}_l) \tag{7}$$

2.2　博弈构建

根据式(6)和(7)的定义,降低系统完成时延的任务调度问题可以转化为面向模型中各调度层的联合卸载和迁移问题。在处理调度层 sl_l 时,每个任务和 ES 都需要分别通过卸载和迁移决策来降低各自的代价。此外,由于任务的卸载决策和 ES 的迁移决策之间彼此关联,且存在明显的行动次序区别,故为最小化双方代价,可以将调度层上的联合卸载和迁移问题建模为一个多领导者多跟随者 Stackelberg 博弈 $\mathcal{G}^{S} = <U_l \cup \mathcal{N}, \{(\chi_{m,i})_{v_{m,i}\in U_l}, (\mathcal{S}_n)_{n\in\mathcal{N}}\}, \{(C_{m,i})_{v_{m,i}\in U_l}, (C_n)_{n\in\mathcal{N}}\}>$。其中,任务为领导者,ES 为跟随者。

给定调度层 sl_l 上所有任务的卸载决策 $\boldsymbol{X}_l$,每个 ES 的目标是通过迁移决策最小化卸载至它的所有任务的完成时延。根据式(7),ESn ($\forall n\in\mathcal{N}$) 的迁移问题可以表示为:

$$\min_{S_n} C_n(\boldsymbol{X}_l,\boldsymbol{S}_n,\boldsymbol{S}_{-n}) \tag{8}$$

式中: $\boldsymbol{S}_{-n}$ 为除 ESn 外,其他所有 ES 的迁移决策。

在处理调度层 sl_l 时,基于所有 ES 的迁移决策 $\boldsymbol{S}_l$,每个任务的目标是通过卸载决策最小化自身的完成时延。根据式(6),任务 $v_{m,i}$ ($\forall v_{m,i}\in U_l$) 的卸载问题可以表示为:

$$\min_{x_{m,i}} C_{m,i}(x_{m,i},\boldsymbol{X}_{-(m,i)},\boldsymbol{S}_l) \tag{9}$$

式中: $\boldsymbol{X}_{-(m,i)}$ 为除任务 $v_{m,i}$ 外,调度层 sl_l 上其他所有任务的卸载决策。

3　模型求解

在 2.2 节构建的 $\mathcal{G}^{S}$ 中,一方面,ES 对应的代价不仅取决于其自身的迁移决策,还受到其他 ES 的影响;另一方面,每个任务之间的卸载决策相互影响。因此,可以将调度层上 ES 的迁移问题式(8)和任务的卸载问题式(9)分别建模为一个非合作博弈。换而言之, $\mathcal{G}^{S}$ 由跟随者和领导者的非合作博弈组成。$\mathcal{G}^{S}$ 的目的在于实现 Stackelberg 均衡[21],使跟随者和领导者都不能通过单方面的偏离来降低代价。

定义 1:用 $x^{*}_{m,i}$ 和 $\boldsymbol{S}^{*}_{n}$ 分别表示式(9)和(8)的解,若可行解集 $(\boldsymbol{X}_l,\boldsymbol{S}_l)$ 都满足以下公式,则 $(\boldsymbol{X}^{*}_{l}=\{x^{*}_{m,i}\}, \boldsymbol{S}^{*}_{l}=\{\boldsymbol{S}^{*}_{n}\})$ 为 $\mathcal{G}^{S}$ 的一个 Stackelberg 均衡。

$$C_n(\boldsymbol{X}^{*}_{l},\boldsymbol{S}^{*}_{n},\boldsymbol{S}^{*}_{-n}) \le C_n(\boldsymbol{X}^{*}_{l},\boldsymbol{S}_n,\boldsymbol{S}^{*}_{-n}), \forall\boldsymbol{S}_n\in\mathcal{S}_n, \forall n\in\mathcal{N} \tag{10}$$

$$C_{m,i}(x^{*}_{m,i},\boldsymbol{X}^{*}_{-(m,i)},\boldsymbol{S}^{*}_{l}) \le C_{m,i}(x_{m,i},\boldsymbol{X}^{*}_{-(m,i)},\boldsymbol{S}^{*}_{l}), \forall x_{m,i}\in\chi_{m,i}, \forall v_{m,i}\in U_l \tag{11}$$

将 ES 的跟随者非合作博弈定义为 $\mathcal{G}^{follower}=<\mathcal{N}, (\mathcal{S}_n)_{n\in\mathcal{N}}, (C_n)_{n\in\mathcal{N}}>$,任务的领导者非合作博弈定义为 $\mathcal{G}^{leader}=<U_l, (\chi_{m,i})_{v_{m,i}\in U_l}, (C_{m,i})_{v_{m,i}\in U_l}>$。由文献[21]可知 $\mathcal{G}^{follower}$ 为精确势博弈,且存在纯策略纳什均衡, $\mathcal{G}^{leader}$ 为有限策略博弈且存在混合策略纳什均衡。所以 $\mathcal{G}^{follower}$ 和 $\mathcal{G}^{leader}$ 都可以达到纳什均衡,即 $\mathcal{G}^{S}$ 中至少存在一个 Stackelberg 均衡。

为保证 $\mathcal{G}^{leader}$ 和 $\mathcal{G}^{follower}$ 都能达到一个纳什均衡,本节

设计了一种基于 Q 值的卸载算法来寻找领导者任务的最优卸载决策和一种分布式迭代迁移算法来寻找跟随者 ES 的最优迁移决策,如算法 1 和 2 所示。根据算法 1 和 2,本文总调度方案如算法 3 所示。

3.1 基于 Q 值的卸载算法

针对调度层 sl_l 上的所有任务,为解决卸载问题对应的 $\mathcal{G}^{\text{leader}}$,基于 Q-learning 思想,本文提出了一种基于 Q 值的卸载算法如算法 1 所示。将任务 $v_{m,i}$ 的混合策略表示为概率分布 $\boldsymbol{p}_{m,i}(t) = (p_{m,i,0}(t), p_{m,i,1}(t), \cdots, p_{m,i,N}(t))$,其中 $p_{m,i,x_{m,i}}(t)$ 为任务 $v_{m,i}$ 本地执行($x_{m,i}=0$)或选择决策 $x_{m,i}$($x_{m,i}\neq 0$)进行卸载执行的概率,存在 $0 \leqslant p_{m,i,x_{m,i}}(t) \leqslant 1$ 且 $\sum_{x_{m,i}\in |0|\cup\mathcal{N}} p_{m,i,x_{m,i}}(t) = 1$。对于 ES$n \notin \mathcal{N}_{m,i}$,$p_{m,i,n} = 0$。将 $x_{m,i}$ 与 Q 值相关联,给定迁移决策 $\boldsymbol{S}_l$ 下,任务卸载决策 $x_{m,i}$ 的 Q 值更新公式为:

$$Q_{t+1}(x_{m,i}) = (1-\kappa)Q_t(x_{m,i}) + \kappa(C^{-1}_{m,i}(x_{m,i}, \boldsymbol{X}_{-(m,i)}, \boldsymbol{S}_l) + \eta \max_{x'_{m,i}\in\mathcal{X}_{m,i}} Q_t(x'_{m,i})) \tag{12}$$

式中:$\kappa \in (0,1)$ 为学习率,$\eta \in (0,1]$ 为折扣因子。

为避免局部最优,加快收敛速度,本文引入玻尔兹曼概率分布法,对所有 $x_{m,i} \in \mathcal{X}_{m,i}$,$p_{m,i,x_{m,i}}(t+1)$ 采用如下计算方式:

$$p_{m,i,x_{m,i}}(t+1) = \frac{\exp[Q_{t+1}(x_{m,i})/\tau]}{\sum_{x'_{m,i}\in\mathcal{X}_{m,i}} \exp[Q_{t+1}(x'_{m,i})/\tau]} \tag{13}$$

式中:τ 为玻尔兹曼模型参数,控制探索与利用之间的平衡。设定 $\tau = \tau_0/|U_l|$,其中 τ_0 为常数值,$|U_l|$ 为调度层 sl_l 上存在的任务数量。当 $|U_l|$ 值较大时,τ 值趋近于 0,此时偏向利用以加快收敛;当 $|U_l|$ 值较小时,τ 值趋近于∞,此时偏向探索,有利于寻找最优决策。

算法 1 基于 Q 值的卸载算法

输入:调度层 sl_l 上的任务集合 U_l,任务 $v_{m,i}$ 可选择卸载的 ES 集合 $\mathcal{N}_{m,i}$,ES 集合 $\mathcal{N}$ 及相关参数
输出:任务的卸载决策 $\boldsymbol{X}_l$ 和 ES 的迁移决策 $\boldsymbol{S}_l$
1. 对于 $\forall v_{m,i} \in U_l, \forall x_{m,i} \in \mathcal{X}_{m,i}$,初始化 $Q_t(x_{m,i}) = 0$,$p_{m,i,x_{m,i}}(t) = 1/(1+|\mathcal{N}_{m,i}|)$,初始化迭代回合数 $t = 0$
2. repeat
3. 在 t 回迭代中,每个任务 $v_{m,i}$ 根据其 $\boldsymbol{p}_{m,i}(t)$ 选择卸载决策,并将信息广播给相应的 ES
4. 根据算法 2 得到 ES 的迁移决策 $\boldsymbol{S}_l$ 并将信息广播给所有任务
5. for each $v_{m,i}$ do
6. 根据式(12)更新卸载决策 $x_{m,i}$ 的 Q 值
根据式(13)更新 $\boldsymbol{p}_{m,i}(t)$
7. end for
8. $t = t+1$
9. until $\boldsymbol{p}_{m,i}(t) == \boldsymbol{p}_{m,i}(t-1), \forall v_{m,i} \in U_l$

3.2 分布式迭代迁移算法

当处理调度层 sl_l 时,针对所有 ES$n \in \mathcal{N}$,为解决迁移问题对应的 $\mathcal{G}^{follower}$,本文提出了一种分布式迭代迁移算法如算法 2 所示。设定所有 ES 有序进行抉择,初始时刻,所有 ES 的迁移决策为不迁移任何任务。已知 sl_l 上所有任务的卸载决策 $\boldsymbol{X}_l$,在第 k 回迭代中,当轮到 ESn 更新迁移决策时,ESn 将根据其他 ES 的迁移决策 $\boldsymbol{S}_{-n}$,选择使自己代价最小化的决策进行更新,即:

$$\boldsymbol{S}_n(k) = \underset{\boldsymbol{S}_n\in\boldsymbol{S}_n}{\operatorname{argmin}} C_n(\boldsymbol{X}_l, \boldsymbol{S}_n, \boldsymbol{S}_{-n}) \tag{14}$$

按照顺序,依次更新所有 ES 的迁移决策,重复上述过程,直到所有 ES 的决策选择与上一回保持一致。

算法 2 分布式迭代迁移算法

输入:卸载决策 $\boldsymbol{X}_l$,任务 $v_{m,i}$ 可选择卸载的 ES 集合 $\mathcal{N}_{m,i}$,ES 集合 $\mathcal{N}$ 及相关参数
输出:迁移决策 $\boldsymbol{S}_l$
1. 对于 $\forall v_{m,i} \in U_l, \forall n \in \mathcal{N}$,初始化 $s^n_{m,i}(k) = n$,初始化迭代回合数 $k = 0$
2. repeat
3. $k = k+1$
4. for each $n \in \mathcal{N}$ do
5. ES 采用暴力枚举法,根据式(14)得到最优迁移决策并广播给其他 ES
6. end for
7. until $\boldsymbol{S}_l(k) == \boldsymbol{S}_l(k-1)$

3.3 总调度方案

表 3 为总的调度方案,根据算法 3 可以得到系统完成时延。其中步骤 2~12 为模型中调度列表的构建过程。步骤 13~16 为利用算法 1 和 3 的算法依次处理每一调度层的过程。

算法 3 总调度方案

输入:所有用户的应用程序 G_m
输出:系统完成时延 T
1. 初始化系统完成时延 $T = 0$
2. for $m = 1$ to $\mathcal{M}$ do
3. 创建虚拟任务 $v_{m,0}$
4. $G^*_m = \text{connect}(v_{m,0}, G_m)$
5. 采用 BFS 遍历 G^*_m,并设置调度编号 l
6. end for
7. for $m = 1$ to $\mathcal{M}$ do
8. for $i = 1$ to $|v_{m,i}|$ do
9. 将任务存储在对应调度编号 l 的调度层 sl_l 中
10. end for
11. end for
12. 得到调度列表 SL
13. for $l = 1$ to $|SL|$ do
14. 根据算法 1 得到任务卸载决策 $\boldsymbol{X}_l$ 和迁移决策 $\boldsymbol{S}_l$
15. 根据式(4)计算调度层完成时延 T_l
16. end for
17. 根据式(5)计算系统完成时延 T

4　实验结果与分析

4.1　参数设置

本文使用 MATLAB 平台进行仿真实验，在仿真实验中，假设有多个用户和 ES 随机均匀分布在 1 km×1 km 的正方形小区中。本文使用两种类型的 DAG，一种根据当前主流应用生成，如人脸识别和虚拟现实[9]等；另一种随机生成，组成任务的输入数据大小和所需计算资源分别在[500，1 500] KB 和[200，500] Megacycles 随机分布[15]。考虑到用户终端和边缘服务器计算能力的异构性，F_m^L 和 F_n^E 分别在[0.5，1]和[20，40]随机取值[22]，单位为 GHz。设置信道增益为 $G_{m,n}=I_{m,n}^{-r}$，其中 $I_{m,n}$ 为用户 m 与 ESn 的距离，$r=4$ 为路径损耗因子[20]，高斯白噪声功率 σ^2 为 10^{-13} W。其余相关仿真参数设置如表 1 所示[22-23]。

表 1　仿真参数

Table 1　Simulation parameter

参数	值	参数	值
W_n /MHz	[5，20]	κ	0.2
p_m /W	[0.05，0.15]	η	0.9
$R_{n\to d}$ /Mbps	[100，1 000]	τ_0	30

4.2　结果与分析

1）算法

首先评估基于 Q 值的卸载算法的收敛性。考虑到用户与 ES 的通信范围，某特定任务除了本地只能卸载至 ES1 和 ES5，该任务卸载决策的概率分布如图 3 所示。观察图 3，经过 30 回合的迭代后，该任务混合卸载策略的概率分布逐渐收敛，即将该任务卸载至 ES1 的概率变为了 1，而任务在本地和卸载至 ES5 的概率都变为了 0。因此，仿真结果验证了基于 Q 值的卸载算法可以快速收敛于一个稳定的可行解。

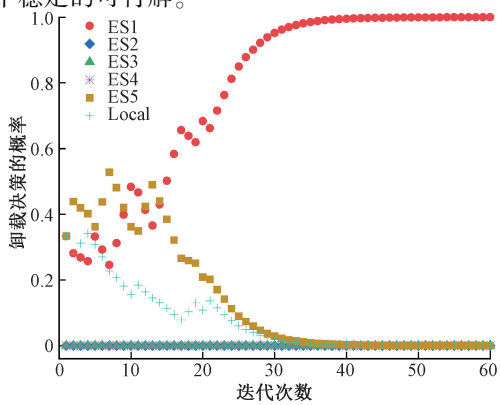


图 3　某任务卸载决策的概率分布

Fig. 3　Probability distribution of a task offloading decision

接着，为验证本文采用的算法在减少系统完成时延方面的性能，基于本文提出的依赖型任务的调度模型，引入两种基线方案进行对比：（1）随机：调度层上每个任务随机进行本地执行或者卸载给 ES，迁移算法参照本文。（2）贪婪：调度层上每个任务根据每次迭代中 Q 值最大的决策进行卸载，迁移算法参照本文。

图 4 显示 5 个 ES 下，系统完成时延与用户数量的关系图。由图可知，系统完成时延随着用户的增加而增加。对于其他两种方案，本文方案的系统完成时延最低，特别当用户数量为 40 时，系统完成时延比其他方案分别低 62.9%和 44.1%。这是因为在随机方案中，用户数量的增加使得解空间的范围过大，造成其不能快速收敛并找到最优卸载决策。在贪婪方案中不能有效探索其他卸载决策，容易陷入局部最优。而本文方案根据调度层的任务数量，合理调整了参数 τ 的大小，有效平衡了探索与利用，有利于寻找最优决策。

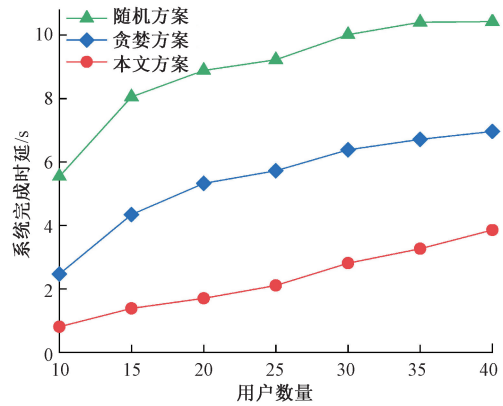


图 4　不同用户数量下的系统完成时延

Fig. 4　System completion delay under different number of users

图 5 显示 20 个用户下，系统完成时延与 ES 数量的关系图。由图 5 可知，系统完成时延随着 ES 数量的增加而减少，这是因为资源的充足缓解了任务对有限资源的竞争。特别的，当 ES 数量为 8 时，对比于其他方案，本文方案的系统完成时延分别低 81.9%和 63.2%，因此，本文方案在 ES 数量增加时依然可以保持系统完成时延平稳占优。

2）调度模型

为验证本文提出的依赖型任务的调度模型在减少系统完成时延方面的优越性，本文进行了 50 次重复实验，并取实验结果的平均值作为最终结果与以下方案进行了对比：（1）Mutas+SA：基于凸优化和模拟退火的服务器分配和资源管理方案[19]。（2）顺序调度：每个用户的应用程序按顺序逐个串行调度，如文献[18]，并采用本文的卸载和迁移算法。（3）非协作：在本文设计的调度模型下，任务仅采用基于 Q 值的卸载算法，任务不迁移。

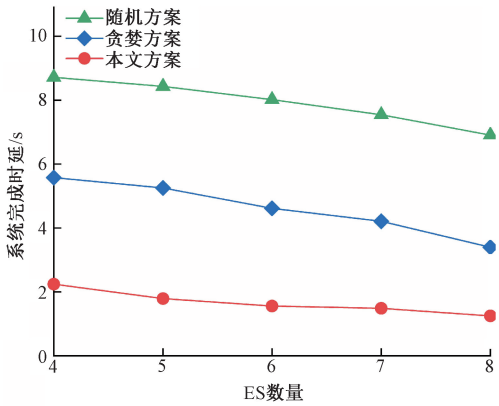


图 5 不同 ES 数量下的系统完成时延

Fig. 5 System completion delay under different number of ESs

图 6 显示了 5 个 ES 下,系统完成时延与用户数量的关系图。由图 6 可知,对比其他 3 种方案,本文方案的系统完成时延始终保持最低,特别当用户数量为 40 时,分别低 86.2%,70.5%和 20.1%。主要原因在于 Mutas+SA 方案要求 ES 为系统中所有任务同时分配并预留资源,这不仅加剧了任务对有限资源的竞争,也造成了部分资源长时间处于闲置状态。顺序调度方案未考虑到利用不同应用程序之间任务的并行关系。非协作方案未利用到跨服务器协作,从而不能有效均衡负载。而本文方案通过构建的调度模型,不仅考虑了跨服务器协作来提高资源利用率,还建立了调度列表来保证依赖型任务的执行顺序,最大化了可并行执行的任务数量。由图可知,随着用户数量增加,本文方案的性能优势更加明显,这验证了本文模型的优越性。

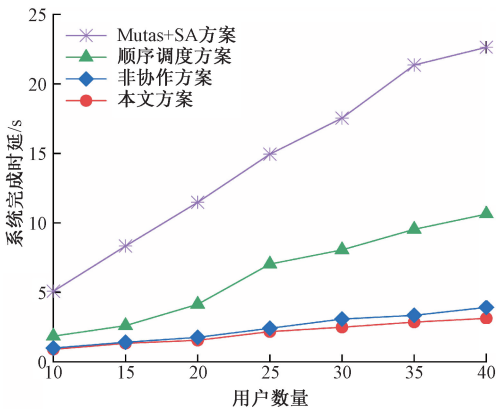


图 6 不同用户数量下各方案的系统完成时延

Fig. 6 System completion delay of each scheme under different number of users

图 7 显示了 20 个用户下,系统完成时延与 ES 数量的关系图。由图可知,对于 Mutas+SA 方案,ES 数量的增加缓解了任务对有限资源的竞争,系统完成时延逐渐减少。对于顺序调度方案,因为没有能有效利用任务间的并行关系,导致后期增加的部分资源处于闲置状态,所以当 ES 超过 5 时,系统完成时延降低不再明显。对于非协作方案,ES 数量的增加使得系统中的资源增多,此时不能完全发挥跨服务器协作的优势,所以此方案与本文方案的系统完成时延会呈现逐渐接近的趋势。整体而言,本文方案通过构建调度模型可以有效发掘资源的可用性,使所有计算资源都尽可能地保持了繁忙状态。因此对比其他方案,本文方案在 ES 数量增加时依然可以保持系统完成时延最低。

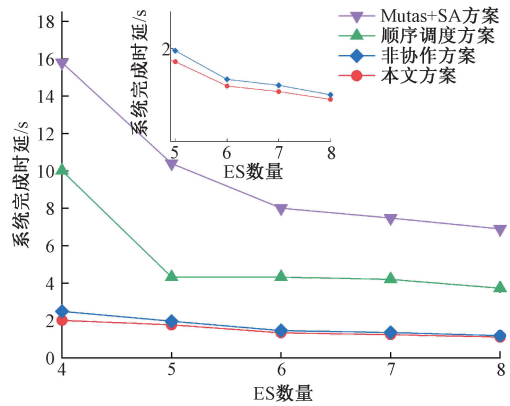


图 7 不同 ES 数量下各方案的系统完成时延

Fig. 7 System completion delay of each scheme under different number of ESs

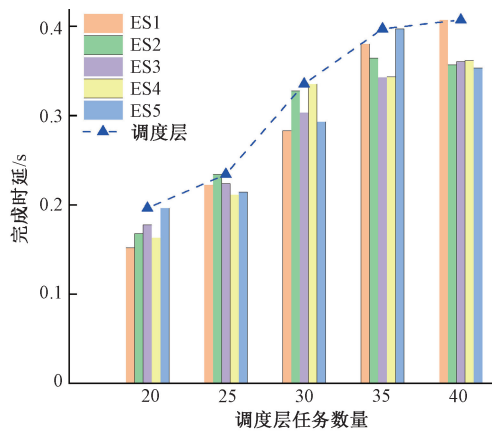


图 8 不同调度层任务数量下各 ES 的完成时延

Fig. 8 Completion delay of each ES under different scheduling layer tasks

ES 的异构性和负载的不均衡性导致它们完成其上所有任务所需的时延具有差异性。因此,为避免本文在调度模型中设定的调度层处理方案影响到系统完成时延,本文引入了跨服务器协作来均衡负载,图 8 验证了跨服务器协作的有效性。由图 8 可知,调度层完成时延随着任务数量的增加而增加,这是因为调度层任务数量的

增加减少了每个任务可使用的资源。图 8 比较了每个 ES 完成分配给它的所有任务所需的时延,即使是当调度层任务数量为 40 时,不同 ES 之间最大的完成时延落差也仅为 0.066 s,这说明在不同的调度层任务数量下,不同 ES 的完成时延(资源完全释放时延)都基本相近,所以在本文调度模型中考虑跨服务器协作可以有效地均衡负载,减少下一调度层的等待时延,降低了调度层处理方案对系统完成时延的影响。

5　结　论

本文在资源有限的多用户、多边缘服务器的 MEC 环境下为降低系统完成时延,首先利用 BFS 算法提出了一种基于跨服务器协作的依赖型任务调度模型,将降低系统完成时延问题转化为面向各调度层的卸载和迁移问题。然后引入多领导者多跟随者的 Stackelberg 博弈对卸载和迁移问题进行建模。最后通过基于 Q 值的卸载算法和分布式迭代迁移算法实现了 Stackelberg 均衡,获得了任务和 ES 的最优卸载和迁移决策。实验结果验证了所提模型能够有效减少系统完成时延。然而,本文忽略了任务之间可能存在的数据通信时延,后续工作中将关注此因素对系统完成时延造成的影响。

参考文献

[1] NING Z, DONG P, WANG X, et al. Mobile edge computing enabled 5G health monitoring for internet of medical things: A decentralized game theoretic approach[J]. IEEE Journal on Selected Areas in Communications, 2021, 39(2): 463-478.

[2] 马学森, 谈杰, 陈树友, 等. 云计算多目标任务调度的优化粒子群算法研究[J]. 电子测量与仪器学报, 2020, 34(8): 133-143.
MA X S, TAN J, CHEN SH Y, et al. Optimization particle swarm optimization algorithm for multi-objective task scheduling in cloud computing[J]. Journal of Electronic Measurement and Instrumentation, 2020, 34(8): 133-143.

[3] 谢人超, 廉晓飞, 贾庆民, 等. 移动边缘计算卸载技术综述[J]. 通信学报, 2018, 39(11): 138-155.
XIE R CH, LIAN X F, JIA Q M, et al. Overview of mobile edge computing offloading technology[J]. Journal on Communications, 2018, 39(11): 138-155.

[4] CHEN W, WANG D, LI K. Multi-user multi-task computation offloading in green mobile edge cloud computing[J]. IEEE Transactions on Services Computing, 2018, 12(5): 726-738.

[5] JI W, LIANG B, WANG Y, et al. Crowd V-IoE: Visual internet of everything architecture in AI-driven fog computing[J]. IEEE Wireless Communications, 2020, 27(2): 51-57.

[6] KUANG Z, LI L, GAO J, et al. Partial offloading scheduling and power allocation for mobile edge computing systems[J]. IEEE Internet of Things Journal, 2019, 6(4): 6774-6785.

[7] YANG L, ZHANG H, LI X, et al. A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing[J]. IEEE Transactions on Networking, 2018, 26(6): 2762-2773.

[8] HABER E E, NGUYEN M T, ASSI C. Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds[J]. IEEE Transactions on Communications, 2019, 67(5): 3407-3421.

[9] LIU L, ZHONG R, ZHANG W, et al. Cutting the cord: Designing a high-quality untethered VR system with low latency remote rendering[C]. The 16th Annual International Conference on Mobile Systems, Applications and Services, New York: ACM, 2018: 68-80.

[10] LIU B, XU X, QI L, et al. Task scheduling with precedence and placement constraints for resource utilization improvement in multi-user MEC environment[J]. Journal of Systems Architecture, 2021(114): 101970.

[11] XU X L. A computation offloading method over big data for IoT-enabled cloud-edge computing[J]. Future Generation Computer Systems, 2019(95): 522-533.

[12] SUNDAR S, LIANG B. Offloading dependent tasks with communication delay and deadline constraint[C]. IEEE Conference on Computer Communications (INFOCOM), Piscataway: IEEE, 2018: 37-45.

[13] ZHAO G, XU H, ZHAO Y, et al. Offloading dependent tasks in mobile edge computing with service caching[C]. IEEE Conference on Computer Communications (INFOCOM), Piscataway: IEEE, 2020: 1997-2006.

[14] YAN J, BI S, ZHANG Y A. Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency[J]. IEEE Transactions on Wireless Communications, 2020, 19(1): 235-250.

[15] SHU C, ZHAO Z, HAN Y, et al. Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach[J]. IEEE Internet of Things Journal, 2020, 7(3): 1678-1689.

[16] LIU L, TAN H, HUANG H, et al. Dependent task placement and scheduling with function configuration in edge computing[C]. IEEE/ACM International Symposium on Quality of Service (IWQoS), New York: ACM, 2019: 20.

[17] NING Z L, DONG P, KONG X J, et al. A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things[J]. IEEE Internet of Things Journal, 2019, 6(3): 4804-4814.

[18] 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[J]. 软件学报, 2020, 31(6): 1889-1908.
LIU W, HUANG Y CH, DU W, et al. Resource-constrained serial task offloading strategy in mobile edge computing[J]. Journal of Software, 2020, 31(6): 1889-1908.

[19] WANG C, ZHANG S, QIAN Z Z, et al. Joint server assignment and resource management for edge-based MAR system [J]. IEEE/ACM Transactions on Networking, 2020, 28(5): 2378-2391.

[20] JOŠILO S, DÁN G. Joint management of wireless and computing resources for computation offloading in mobile edge clouds [J]. IEEE Transactions on Cloud Computing, 2019, 9(4): 1507-1520.

[21] FRIHAUF M, KRSTIC M, BASAR T, et al. Nash equilibrium seeking in noncooperative games[J]. IEEE Transactions on Automatic Control, 2012, 57(5): 1192-1207.

[22] ZHENG J, CAI Y, WU Y, et al. Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach [J]. IEEE Transactions on Mobile Computing, 2018, 18(4): 771-786.

[23] HAN C, HUO L, TONG X, et al. Spatial anti-jamming scheme for internet of satellites based on the deep reinforcement learning and Stackelberg game[J]. IEEE Transactions on Vehicular Technology, 2020, 69(5): 5331-5342.

作者简介

王瑶,2019 年于南通大学获得学士学位,现为江南大学硕士研究生,主要研究方向为移动边缘计算。
E-mail:1254019282@qq.com

Wang Yao received B. Sc. degree from Nantong University in 2019. Now she is a M. Sc. candidate at Jiangnan University. Her main research interest includes mobile edge computing.

卢先领,2009 年于南京理工大学获得博士学位,现为江南大学物联网工程学院教授,主要研究方向为无线传感器网络、移动边缘计算。
E-mail:jnluxl@jiangnan.edu.cn

Lu Xianling received Ph. D. degree from Nanjing University of Science and Technology in 2009. Now he is a professor at Jiangnan University. His main research interests include wireless sensor networks and mobile edge computing.

沈义峰,2016 年于常熟理工学院获得学士学位,现为江南大学硕士研究生,主要研究方向为数据挖掘、深度学习。
E-mail:745781529@qq.com

Shen Yifeng received B. Sc. degree from Changshu Institute of Technology in 2016. Now he is a M. Sc. candidate at Jiangnan University. His main research interests include data mining and deep learning.