Digital Communications and Networks 10 (2024) 693–705

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

Joint computation offloading and parallel scheduling to maximize delay-guarantee in cooperative MEC systems

Mian Guo [a], Mithun Mukherjee [b], Jaime Lloret [c,*], Lei Li [d], Quansheng Guan [d], Fei Ji [d]

[a] School of Electronics and Information, Guangdong Polytechnic Normal University, Guangzhou, China
[b] School of Artificial Intelligence, Nanjing University of Information Science and Technology, Nanjing, China
[c] Universitat Politecnica de Valencia, 46022, Valencia, Spain
[d] South China University of Technology, China

ARTICLE INFO

Keywords:
Edge computing
Computation offloading
Parallel scheduling
Mobile-edge cooperation
Delay guarantee

ABSTRACT

The growing development of the Internet of Things (IoT) is accelerating the emergence and growth of new IoT services and applications, which will result in massive amounts of data being generated, transmitted and processed in wireless communication networks. Mobile Edge Computing (MEC) is a desired paradigm to timely process the data from IoT for value maximization. In MEC, a number of computing-capable devices are deployed at the network edge near data sources to support edge computing, such that the long network transmission delay in cloud computing paradigm could be avoided. Since an edge device might not always have sufficient resources to process the massive amount of data, computation offloading is significantly important considering the cooperation among edge devices. However, the dynamic traffic characteristics and heterogeneous computing capabilities of edge devices challenge the offloading. In addition, different scheduling schemes might provide different computation delays to the offloaded tasks. Thus, offloading in mobile nodes and scheduling in the MEC server are coupled to determine service delay. This paper seeks to guarantee low delay for computation intensive applications by jointly optimizing the offloading and scheduling in such an MEC system. We propose a Delay-Greedy Computation Offloading (DGCO) algorithm to make offloading decisions for new tasks in distributed computing-enabled mobile devices. A Reinforcement Learning-based Parallel Scheduling (RLPS) algorithm is further designed to schedule offloaded tasks in the multi-core MEC server. With an offloading delay broadcast mechanism, the DGCO and RLPS cooperate to achieve the goal of delay-guarantee-ratio maximization. Finally, the simulation results show that our proposal can bound the end-to-end delay of various tasks. Even under slightly heavy task load, the delay-guarantee-ratio given by DGCO-RLPS can still approximate 95%, while that given by benchmarked algorithms is reduced to intolerable value. The simulation results are demonstrated the effectiveness of DGCO-RLPS for delay guarantee in MEC.

1. Introduction

With the growing development of Internet of Things (IoT), a lot of new services and applications are conceived for the fifth generation (5G) and beyond wireless communication networks [1,2]. In 5G and beyond, the data velocity and volume are growing exponentially from industry, agriculture, etc. It is estimated that there will be 80 billion connected devices, and the global data will reach 163 zettabytes by 2025 [3]. Most of data is from delay-sensitive and computation intensive applications, such as autonomous vehicles, real-time manufacturing, which require high performance computing with ultra-low latency [4–6]. Although cloud computing supports high performance computing, it is unable to satisfy the ultra-low latency requirement, since the network transmission delay from data sources to the remote cloud is inevitable and becomes the bottleneck for ultra-low latency [7,8].

Mobile Edge Computing (MEC), which enables high performance computing at the proximity area of data sources, is regarded as a promising paradigm for Quality of Service (QoS) provisioning, particularly delay guarantee for mobile applications requiring ultra-low latency [9,10]. In MEC, the computing-enabled mobile nodes (e.g., smartphones, IoT devices with Raspberry Pi) [3,11] and the edge server (such as edge routers, base stations, wireless Access Point (AP), etc.) [12,13] can

* Corresponding author.
E-mail addresses: mianguo@gpnu.edu.cn (M. Guo), m.mukherjee@ieee.org (M. Mukherjee), jlloret@dcom.upv.es (J. Lloret), eelilei@scut.edu.cn (L. Li), eeqshguan@scut.edu.cn (Q. Guan), eefeiji@scut.edu.cn (F. Ji).

https://doi.org/10.1016/j.dcan.2022.09.020
Received 5 January 2021; Received in revised form 29 June 2022; Accepted 25 September 2022
Available online 1 October 2022

provide computing services for the close end users. An edge server is a highly-virtualized computing system that supports running multiple computation intensive tasks in parallel by equipping with on-board large-volume data storage and computing facility [12]. Differently, a mobile node generally runs one computation intensive task at a time, since its computing capability is far smaller than that of an edge server due to its battery and volume constraints.

To reduce transmission delay, it is desired to process all data within an MEC network [14,15]. However, computing-enable edge devices, including edge servers and mobile nodes, might not always have sufficient computation resources to handle the massive amount of data [16,17]. Therefore, the cooperation among nodes and servers at the vicinity of end users is indispensable for the utilization maximization of the edge computing paradigm. In a cooperative MEC system, complicated computing algorithms, such as deep learning [18,19] and federated learning [14,20], could be operated within a radio access network. Accordingly, ultra-low latency might be provided for computation intensive applications.

1.1. Motivation

The end-to-end delay of a computation intensive task mainly consists of computation delay and offloading transmission delay. Computation delay is determined by the computation resource and scheduling priority allocated to the task in a computing device, while offloading transmission delay would be affected by the network position of the computing device. Accordingly, the quality of service provision for delay-sensitive applications in an MEC system has to address the following issues. Firstly, computation resources and tasks are both heterogeneous. The MEC server has multi-core CPU to carry out parallel scheduling, while each mobile node often has only a single-core CPU to execute a task at a time. The tasks also have distinct latency requirements for the MEC system. Thus, the stochastic task arrival, distinct delay requirements, heterogeneous computation capabilities in mobile nodes and the server, offloading delay, etc., would make the offloading and scheduling problem complicated. Secondly, the offloading and scheduling are coupled in determining the delay of a task. Distinct resource allocation and scheduling algorithms provide quite different computation delays for distinct types of tasks [21]. Thus, the parallel scheduling algorithm in the edge server could affect the offloading decisions at mobile nodes by providing distinct computation delays for tasks from distinct nodes. In addtition, the offloading decisions at mobile nodes could also affect the resource allocation and scheduling of an edge server by adjusting the task offloading rate.

Although a number of computation offloading algorithms are proposed to reduce service delay in recent years, most of them addressed partial problems aforementioned by optimizing either the offloading decision [22,23], wireless uplink transmission resource [24,25], or computation resources [21]. Some proposals focuse on the joint task offloading, radio and computation resource allocation [26,27], offline or partially offline algorithms are designed so as to iteratively search for the optimal solutions.

1.2. Objective

Unlike the proposals mentioned above, the objective of this paper is to design an online solution for joint computation offloading and intra-server parallel scheduling in an MEC system so as to maximize the tasks' delay-guarantee-ratio, that is, the ratio of the number of tasks that meet their delay requirements to the total number of tasks requested. Specifically, we consider a cooperative MEC system in which a number of computing-enable mobile nodes and a multi-core MEC server are connected via a wireless network. The computation tasks, generated from mobile nodes, can be executed locally or be offloaded to the MEC server. To find out optimal offloading and intra-server scheduling policies, we first formulate a delay-based mobile-edge cooperative offloading and scheduling problem with the goal of maximizing the delay-guarantee-ratio. Since offloading and scheduling decisions are made in different stages for a task, we decompose the problem into two concatenated subproblems, including the Computation Offloading (CO) and Parallel Scheduling (PS) subproblems. Then we design a scheme with two stages: Delay-Greedy Computation Offloading (DGCO) for distributed mobile nodes in the offloading stage and Reinforcement-Learning based Parallel Scheduling (RLPS) for the MEC server in the scheduling stage for solving the CO and PS subproblems respectively. Under an offloading delay broadcast mechanism, the DGCO in mobile nodes and RLPS in the MEC server cooperatively achieve a one-shot optimization of delay-guarantee-ratio.

1.3. Contributions

The main contributions are summarized as follows.

- Formulation for a delay-based computation offloading and intra-server parallel scheduling problem. We use the delay-guarantee-ratio as the optimization objective to construct the joint offloading and scheduling problem in an MEC system consisting of heterogeneous computing devices, including a multi-core MEC server and multiple mobile nodes.
- Joint offloading and parallel scheduling with mobile-edge cooperation. The online DGCO-RLPS scheme is designed to find out optimal offloading decisions and parallel scheduling policies in the MEC system for delay-guarantee-ratio maximization. In the proposal, the problem is decomposed into two concatenated subproblems and solved via DGCO and RLPS respectively. The one-shot optimization of delay-guarantee-ratio is achieved by joining DGCO and RLPS via the mobile-edge cooperative delay broadcast mechanism.
- Extensive simulations are conducted to illustrate that DGCO-RLPS can bound the end-to-end delay of tasks, even under slightly heavy task load, the delay-guarantee-ratio can still approximate 95%.

The remaining part of this paper is organized as follows. Section 2 introduces the related work. Section 3 describes the network and delay models. Section 4 formulates the problem and then decomposes it into two subproblems. Section 5 describes the details of DGCO-RLPS. Simulation studies are conducted to demonstrate the efficiency of DGCO-RLPS in Section 6. Section 7 concludes this paper.

2. Related work

Edge computing [28,29] has emerged as an attractive computing model to provide QoSs for the ever-increasing delay-sensitive and computation intensive applications. Recently, a number of computation offloading and resource allocation algorithms have been proposed to deploy cloud-like computation resources at the network edge, achieving different objectives. Basically, these algorithms can be classified into two catalogues depending on whether they join computation offloading and intra-server scheduling.

2.1. Offloading without intra-server scheduling

The computation offloading among edge clouds and the remote cloud are studied in Ref. [22], where an interior point method is used to solve the offloading problem for maximizing the probability that tasks can meet the delay requirements. The joint scheduling and cloud offloading problem is formulated linearly and then addressed with Linear Programming (LP) in Ref. [23] for reducing the execution time of mobile applications. Mukherjee et al. in Ref. [30] have studied the

latency-driven task data offloading in fog computing network for industrial applications. The parallel task offloading is implemented by splitting tasks into tne local fog and neighbor fog nodes as well as the remote cloud. Le et al. have designed a deep reinforcement learning based offloading scheme in ad-hoc mobile cloud for maximizing the user's utility [31]. The knowledge-driven service offloading decision for vehicular edge computing is studied based on deep reinforcement learning in Ref. [32]. However, the existing reinforcement learning based approaches are designed either for an MEC environment consisting of one end device [31], or for an MEC with multiple end devices using the same trained model [32,33], they can not be applied to an MEC system with multiple types of end devices, making independent offloading decisions and performing one-shot optimization.

Although the computation resource in a cloud is abundant, it is limited in an edge server. Thus, some offloading algorithms assum that each edge server is capable of serving one task at a time. Liu et al. have studied the tradeoff between latency and reliability in task offloading for MEC in Ref. [34]. They partitioned a task into sub-tasks and then offloaded them to multiple nearby edge nodes in sequence. A multi-dimensional searching and adjusting method is also proposed to join computing partitioning and resource allocation in mobile edge clouds to reduce the average delay of latency sensitive applications [35]. A delay-based workload allocation for IoT-edge-cloud computing systems is studied in Ref. [36] to improve energy efficiency. In Ref. [25], a joint subcarrier and CPU time allocation for MEC is studied. In the proposal, the offloaded tasks are scheduled one-by-one in the ascending order of users' queue lengths.

In contrast to the above proposals, which assumes that the remote task execution time is known or ignored in computation offloading decision, several recent works are focused on offloading algorithms based on the predictive remote task execution time. For example, Khairy et al. proposed an adaptive and context-aware offloading algorithm that uses supervised learning to predict the remote execution time and energy consumption, with the aim of optimizing both the execution time and energy consumption for smartphones [37]. Later, Hu et al. proposed the Maximum Efficiency First Ordered (MEFO) task offloading algorithm based on learning driven task execution time prediction in an asymmetrically informed edge computing environment, with the aim of achieving high success rate of task offloading and small processing delay [38].

2.2. Joint offloading and intra-server scheduling

A power-constrained delay minimization computation offloading problem is studied for MEC systems in Ref. [39]. However, the proposal only considered the task scheduling from one mobile device. In addition, tasks are queued in the local mobile device. The decision to computing offloading is made only after the previous task is completed. In Ref. [26], a joint task offloading and parallel resource allocation for multi-server MEC networks is formulated as a Mixed Integer Non-Linear Program (MINLP). They address the resource allocation subproblem by using convex and quasi-convex optimization techniques with fixed task offloading decision, and using a heuristic algorithm to solve the task offloading subproblem. The task assignment in a multi-cloudlet network connected via a wireless SDN network is studied in Ref. [40] to reduce the task latency. In the proposal, the arrival of offloaded tasks is assumed to be a Poisson process, and the cloudlet is modeled as an M/G/1 queue. They allocated the computation resource of a cloudlet to various flows according to traffic arrival rates. In Ref. [27], the computation offloading, resource allocation, and flying trajectory scheduling problem (named JSORT) in the UAV-assisted mobile edge computing environment are studied. In order to minimize the average weighted energy consumption, a Lyapunov-based approach is applied to analyze the task queue. Then, the problem to minimize the energy consumption is decomposed into three manageable subproblems. In order to find the optimal CPU-cycle frequency of the UAV, the classical interior method is used.

The joint computation offloading and intra-server scheduling in MEC are studied. In contrast to the previous proposals, an MEC with various types of computation tasks is considered. Different types of tasks have distinct resource requirements and tolerate different delay deadlines. Moreover, task arrival process could be non-independent and identical (non-i.i.d) distribution. In addition, in contrast to the aforementioned proposals that concerned on average performance, our study focuses on satisfying individuals' delay requirements. We design online algorithms to solve the joint computation offloading and intra-server scheduling problem in such MEC.

3. Network and delay model

As illustrated in Fig. 1, an MEC system with computing-enabled mobile nodes and a multi-core MEC server is considered. Each mobile node generates computation tasks stochastically, where the data is from the nearest low-tier IoT devices. Generally, the computing capability of a mobile node is lower than that of an MEC server. Accordingly, it is assumed that a mobile node executes a single task at a time with its full computing capability, while the MEC server supports the execution of multiple tasks in parallel via virtualization technologies [41]. In particular, the computation resource of the MEC server is allocated to multiple Virtual Machine (VM) instances such that distinct tasks can be executed on distinct VM instances in parallel.

In the MEC system, the following two-stage decision for a task is considered. The first is the **computation offloading decision** in each of computing-enabled mobile nodes. When a new task is generated, the mobile node makes offloading decision (i.e., either local computing or offloading to the MEC server) for the task. If the task is offloaded, it will be transmitted from the mobile node to the MEC server via the wireless communication network. The second is the **parallel scheduling**

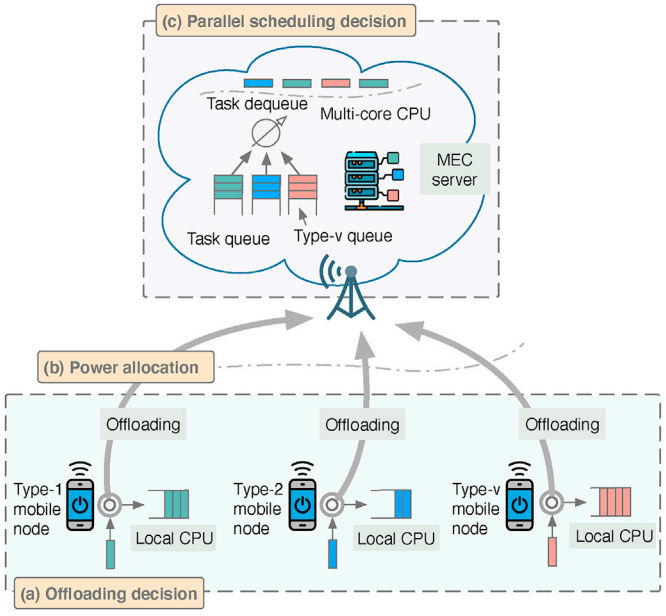

Fig. 1. Model of an MEC system. (a) Distributed mobile nodes make offloading decisions depending on either local computing or offloading to the MEC server for new computation tasks; (b) If a task is determined to offloaded, it will be transmitted from the mobile node to the MEC server via the base station; (c) The MEC server determines how many tasks can be served in parallel and which tasks to be executed.

Table 1
Summary of notations.

Symbols	Definition
V	The number of task types
M_v	The number of type-v mobile nodes
$f^{\mathrm{M}}_{v,m}$	The computation capability (CPU cycles per second) of the mth type-v mobile node
f^{F}	The computation capability (CPU cycles per second) of the MEC server
$A_{v,m}(t)$	The task arrival indicator for the mth type-v mobile node in slot t
$I_{v,m}(t)$	The binary offloading decision indicator for the mth type-v task in slot t
$X_v(t)$	The total number of type-v tasks in slot t
$Y^{\mathrm{M}}_v(t)$	The number of type-v tasks that are locally computed
$Y^{\mathrm{F}}_v(t)$	The number of type-v tasks that are offloaded
S_v	The data size (bits) of a type-v task
L_v	The total CPU cycles required to execute a type-v task
$Q_{v,m}(t)$	The queue length of the mth type-v node at the beginning of slot t
$Q^{\mathrm{F}}_v(t)$	The queue length of type-v tasks in the MEC server at the beginning of slot t
$T_{v,m}(t)$	The end-to-end delay of the mth type-v task generated in slot t
T^{Th}_v	The delay deadline of type-v task
$T^{\mathrm{M}}_{v,m}(t)$	The local computation delay of the task from the mth type-v mobile node
$T^{\mathrm{F}}_{v,m}(t)$	The offloading delay of the task from the mth type-v node
$T^{\mathrm{F}}_{v,m,\mathrm{Tx}}(t)$	The uploading delay of the task from the mth type-v node
$T^{\mathrm{F}}_{v,m,\mathrm{CPU}}(t)$	The MEC server computation delay of the task from the mth type-v mobile node
$R_{v,m}(t)$	The uploading data rate of the mth type-v node in slot t
$\mathcal{R}(t)$	The parallel scheduling decision vector (number of tasks) with respect to task types scheduled in the MEC server in slot t
I	The offloading decision vector with respect to task types and task number in all time slots
$\mathcal{R}$	The parallel scheduling decision (number of tasks) with respect to task types scheduled in the MEC server in all time slots
$P(t)$	The delay-guarantee-ratio of the MEC system in slot t
P	The long-term delay-guarantee-ratio of the MEC system
P^{F}	The delay-guarantee-ratio in the MEC server
$\mathbb{M}$	The mobile node vector with respect to mobile node types
$\mathbb{V}$	The set of task types
$\mathbb{X}_v(t)$	The set of type-v tasks generated in slot t
$\mathbb{Y}^{\mathrm{M}}_v(t)$	The set of type-v tasks for local computing
$\mathbb{Y}^{\mathrm{F}}_v(t)$	The set of type-v tasks for offloading

decision in the MEC server. At each decision epoch, the MEC server determines how many tasks can be served in parallel and which tasks to be executed. Since these two types of decisions are made on distinct stages for a task, they can be performed in parallel for different tasks, as shown in Fig. 1.

In practice, the computation-intensive IoT applications can be classified into a limited number of types according to their expected latency [42]. It is assumed that the computation tasks are classified into V number of types based on the size of task data and tolerable latency. A mobile node is assumed to generate the same type of tasks during all time slots, thus a node that generates type-v tasks is called type-v mobile node herein. Let $\mathbb{M} = (M_1, M_2, \ldots, M_v, \ldots, M_V)$ be the mobile node vector in the MEC system, where M_v is the number of type-v nodes. Let $f^{\mathrm{M}}_{v,m}$ be the computation capability [CPU cycles per second] of the mth type-v node. Similarly, the computation capability of the MEC server is denoted by f^{F} [CPU cycles per second]. The details of some notations are listed in Table 1.

3.1. Task model

A dynamic task arrival model is considered, e.g., each mobile node generates computation tasks stochastically and independently. Let $A_{v,m}(t)$ be the binary indicator for the mth type-v mobile node in slot t,[1] where $A_{v,m}(t) = 1$ indicates that a task is generated in slot t, otherwise $A_{v,m}(t) = 0$. Each type-v task has a delay deadline denoted as T^{Th}_v. Note that if a task fails to meet its delay deadline, the task is said to be delay-deadline unguaranteed. The number of type-v tasks generated from M_v number of type-v nodes in slot t is expressed by

$$X_v(t) = \sum_{m=1}^{M_v} A_{v,m}(t) \tag{1}$$

Let $I_{v,m}(t)$ be a binary offloading decision indicator for the mth type-v task in slot t. It is assumed that when $I_{v,m}(t) = 1$, the mth type-v task is offloaded, and when $I_{v,m}(t) = 0$, the task is locally computed. Then, we have

$$X_v(t) = Y^{\mathrm{M}}_v(t) + Y^{\mathrm{F}}_v(t), \quad v \in \mathbb{V} \tag{2}$$

where $Y^{\mathrm{M}}_v(t) = \sum_{m=1}^{M_v} A_{v,m}(t)\,(1 - I_{v,m}(t))$ and $Y^{\mathrm{F}}_v(t) = \sum_{m=1}^{M_v} A_{v,m}(t)\, I_{v,m}(t)$.

3.2. Local computation delay

When a type-v node determines to locally execute the task generated in time slot t, the task will only experience local computation delay, which is calculated by

$$T^{\mathrm{M}}_{v,m}(t) = T^{\mathrm{M}}_{v,m,\mathrm{que}}(t) + T^{\mathrm{M}}_{v,m,\mathrm{exe}}(t), v \in \mathbb{V}, m \in \mathbb{Y}^{M}_v(t) \tag{3}$$

where $T^{\mathrm{M}}_{v,m,\mathrm{que}}(t) = L_v\, Q_{v,m}(t)/f^{\mathrm{M}}_v$ is the queuing delay; $T^{\mathrm{M}}_{v,m,\mathrm{exe}}(t) = A_{v,m}(t)\,(1 - I_{v,m}(t))\, L_v/f_v$ is the execution delay of the task. The queue length is evolved by

$$Q_{v,m}(t+1) = \max\left[Q_{v,m}(t) + A_{v,m}(t)\left(1 - I_{v,m}(t)\right) - \left\lfloor \frac{f^{M}_v}{L_v} \right\rfloor, 0\right] \tag{4}$$

where $\lfloor X \rfloor$ is the maximum integer number that does not exceed X.

3.3. Offloading delay

When a type-v node determines to offload a task to the MEC server, the task will experience both uploading and computation delays. Let $T^{\mathrm{F}}_{v,m}(t)$ be the delay of the type-v task that is generated in node m in slot t and is offloaded to the MEC server. Then, we have

$$T^{\mathrm{F}}_{v,m}(t) = T^{\mathrm{F}}_{v,m,\mathrm{Tx}}(t) + T^{\mathrm{F}}_{v,m,\mathrm{CPU}}(t) \tag{5}$$

Note that due to small size of the downloading result, similar to literature [22], the uploading transmission delay is mainly considered. Nevertheless, the downloading time is added when the size of the result data becomes larger in specific applications.

3.3.1. Uploading delay

For the uplink transmission from a mobile node to the MEC server, the Orthogonal Frequency Division Multiple Access (OFDMA) system is considered. Let $N^t_{v,m}$ be the number of subbands available to the mth type-v mobile node in slot t. Denote $\mathcal{G}^n_{v,m}(t)$ as the uplink channel-gain-to-noise ratio. Then, the data rate for the mobile node is derived by

$$R_{v,m}(t) = B \sum_{n=1}^{N^t_{v,m}} w^n_{v,m}(t) \log_2\left(1 + \mathcal{P}^n_{v,m}(t)\mathcal{G}^n_{v,m}(t)\right) \tag{6}$$

where B is the bandwidth of a subband, $w^n_{v,m}(t)$ is a binary channel selection variable. $w^n_{v,m}(t) = 1$ indicates that the mobile node decides to offload the task over subband n, otherwise $w^n_{v,m}(t) = 0$. $\mathcal{P}^n_{v,m}(t)$ is the transmission power over subband n. The transmission power is constrained by $\sum_{n=1}^{N^t_{v,m}} w^n_{v,m}(t)\mathcal{P}^n_{v,m}(t) \le \mathcal{P}_{v,m}$, where $\mathcal{P}_{v,m}$ is the maximum transmission power of the mth type-v mobile node.

Therefore, the uploading transmission delay for the type-v task from the mth node to the MEC server is expressed as

[1] The slot t refers to the time interval $[t, t+1)$.

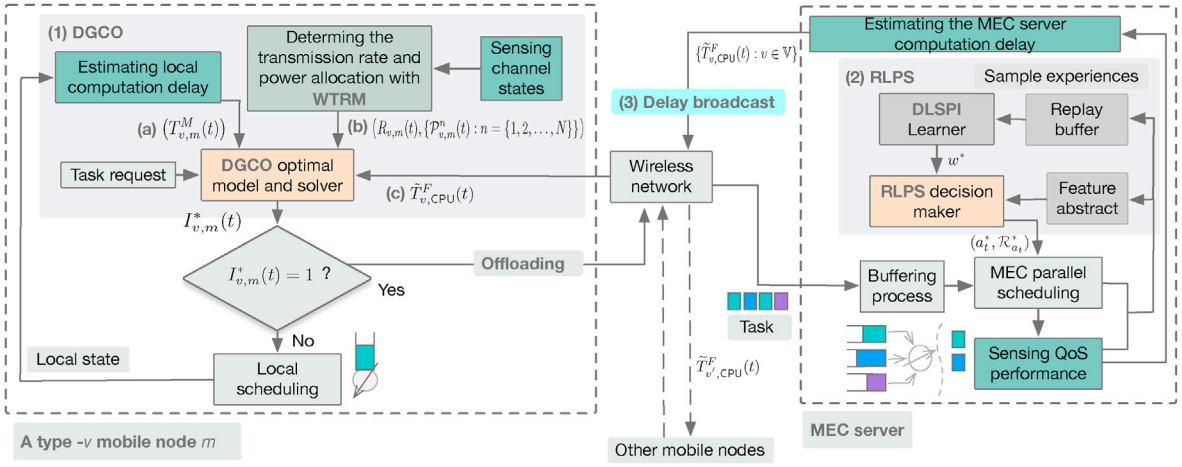

Fig. 2. Online mobile-edge cooperative DGCO-RLPS scheme for the JCOPS problem. (1) The DGCO algorithm is designed for distributed mobile nodes to solve the computation offloading subproblem; (2) The RLPS algorithm is used in the MEC server to solve the parallel scheduling subproblem; (3) Concatenating the two subproblems to obtain optimal solutions for the JCOPS problem via mobile-edge cooperative delay broadcast.

$$T^{\mathrm{F}}_{v,m,\mathrm{Tx}}(t) = \frac{S_v}{R_{v,m}(t)} \tag{7}$$

3.3.2. MEC server computation delay

When a type-v task from mobile node m arrives at the MEC server in slot t, its computation delay can be derived by

$$T^{\mathrm{F}}_{v,m,\mathrm{CPU}}(t) = T^{\mathrm{F}}_{v,m,\mathrm{CPU,que}}(t) + T^{\mathrm{F}}_{v,m,\mathrm{CPU,exe}}(t) \tag{8}$$

where $T^{\mathrm{F}}_{v,m,\mathrm{CPU,que}}(t) = L_v(Q^{\mathrm{F}}_v(t) + Y^{\mathrm{F}'}_v(t))/f^{\mathrm{F}}_v(t)$ is the queuing delay of type-v task in the MEC server; $T^{\mathrm{F}}_{v,m,\mathrm{CPU,exe}}(t) = A_{v,m}(t)I_{v,m}(t)L_v/f^{\mathrm{F}}_v(t)$ is the execution delay; $f^{\mathrm{F}}_v(t)$ is the CPU rate of the MEC server allocated to the vth type tasks; $0 \le Y^{\mathrm{F}'}_v(t)$ is the number of the vth type tasks offloaded from distributed mobile nodes before this task during slot t. The queue length $Q^{\mathrm{F}}_v(t)$ is evolved as the following

$$Q^{\mathrm{F}}_v(t+1) = \max\left[Q^{\mathrm{F}}_v(t) + Y^{\mathrm{F}}_v(t) - r_v(t), 0\right] \tag{9}$$

where $r_v(t) = \lfloor f^{F}_v(t)/L_v \rfloor$ is the number of the vth type tasks executed in slot t in the MEC server. Then the number of tasks from type-1 to type-V scheduled in slot t can be expressed as $\mathcal{R}(t) = (r_1(t), \ldots, r_v(t), \ldots, r_V(t))$.

4. Problem formulation

This paper mainly focuses on the delay performance in term of delay-guarantee-ratio, and it is expressed in the slot t as

$$P(t) = \frac{1}{V}\sum_{v=1}^{V}\frac{\sum_{m=1}^{X_v(t)}\mathbb{1}\left(T_{v,m}(t) \le T^{\mathrm{Th}}_v\right)}{X_v(t)} \tag{10}$$

where $\mathbb{1}(K) = \{0,1\}$, if K is true then, $\mathbb{1}(K) = 1$; and $\mathbb{1}(K) = 0$, otherwise. Moreover, the end-to-end delay for the mth type-v task in slot t is expressed as

$$T_{v,m}(t) = (1 - I_{v,m}(t))\,T^{\mathrm{M}}_{v,m}(t) + I_{v,m}(t)\,T^{\mathrm{F}}_{v,m}(t) \tag{11}$$

As a result, the long-term delay-guarantee-ratio is defined as

$$P = \mathbb{E}[P(t)] = \lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}P(t) \tag{12}$$

4.1. Joint computation offloading and parallel scheduling problem

The objective of this paper is to find a sequential optimal offloading decisions $I^{*} = (I^{*}_{v,m}(t) : \forall v \in \mathbb{V}, \forall m \in \mathbb{X}_v(t), t = 1,2,\ldots,\infty)$ for mobile nodes, and parallel scheduling policies $\mathcal{R}^{*} = (\mathcal{R}^{*}(t) : t = 1,2,\ldots,\infty)$ for the MEC server, to maximize the long term delay-guarantee-ratio. We call the above problem as the Joint Computation Offloading and Parallel Scheduling (JCOPS) problem, and formulate it as

$$\max_{I^{*},\,\mathcal{R}^{*}} \quad P \tag{13a}$$

$$\text{s.t.} \quad (2),(3),(5),(7),(8),(11),(12) \tag{13b}$$

$$\mathbb{E}[A_{v,m}(t)(1 - I_{v,m}(t))] \le f^{\mathrm{M}}_v, \forall v \in \mathbb{V}, \quad m \in \mathbb{X}_v(t) \tag{13c}$$

$$\sum_{v\in\mathbb{V}}\mathbb{E}[Y^{\mathrm{F}}_v(t)] \le f^{\mathrm{F}} \tag{13d}$$

where (13c) and (13d) are queue stability constraints at the mobile nodes and the MEC server, respectively.

4.2. Problem decomposition

Since the JCOPS problem described in (13a)-(13d) involves the offloading decisions in multiple mobile nodes and task scheduling in the MEC server, our approach in this paper is to decompose it into two concatenated subproblems, referred to as the CO subproblem in the involved mobile nodes and the PS subproblem in the MEC server. The offloading decision I^{*} of the CO subproblem is based on the up-to-date result of the PS subproblem. The solution of the PS subproblem will be affected by the offloading decision of the CO subproblem. The detail of the decomposition process is described as the following.

Firstly, given the most updated $\mathcal{R}^{*}(t)$ of the PS subproblem, the current value of MEC server computation delay is also determined. The mobile nodes can also obtain the most updated MEC server computation delay with the offloading delay. Thus, as to mobile nodes, the JCOPS problem is reduced to the CO subproblem as follows:

$$(\mathrm{CO}): \max_{I^{*}} \quad P \tag{14a}$$

$$\text{s.t.} \quad (13b),(13c),(13d) \tag{14b}$$

$$\mathcal{R}(t) = \mathcal{R}^*(t) \tag{14c}$$

where (14a) follows (13a)-(13d) given $\mathcal{R}^*(t)$; (14c) is the most updated parallel scheduling policy of the PS subproblem.

Secondly, the objective of the PS subproblem is further to optimize the delay-guarantee-ratio within the MEC server with respect to the solution of the CO subproblem. That is, the PS subproblem is formulated as

$$(\mathrm{PS}): \max_{R^*} \quad P^{\mathrm{F}} = \mathbb{E}[P^{\mathrm{F}}(t)] \tag{15a}$$

$$\text{s.t.} \quad Y_v^{\mathrm{F}*}(t) \le X_v(t)\,, v \in \mathbb{V} \tag{15b}$$

$$Y_v^{\mathrm{F}*}(t) = \sum\nolimits_{m=1}^{M_v} A_{v,m}(t) I_{v,m}^*(t)\,, v \in \mathbb{V} \tag{15c}$$

$$(13d) \tag{15d}$$

$$\sum\nolimits_{v=1}^{V} r_v(t)\, L_v \le f^{\mathrm{F}} \tag{15e}$$

$$r_v(t) \le Q_v^{\mathrm{F}}(t)\,, v \in \mathbb{V} \tag{15f}$$

where (15b) and (15c) are the task constraints that follow (2) and the solution of the CO subproblem, respectively; (15d) is the queue stability constraint of the MEC server; (15e) is the CPU resource constraint in the MEC server; (15f) follows (9); $P^{\mathrm{F}}(t)$ in (15a) is the delay-guarantee-ratio of the MEC server in slot t, which is defined as

$$\begin{aligned} P^{\mathrm{F}}(t) &= \frac{1}{V}\sum\nolimits_{v=1}^{V} P_v^{\mathrm{F}}(t) \\ &= \frac{1}{V}\sum\nolimits_{v=1}^{V} \frac{\sum_{m=1}^{Y_v^{\mathrm{F}*}(t)} \mathbb{1}\left(T_{v,m,CPU}^{\mathrm{F}}(t) \le T_{v,m}^{\mathrm{F\_Th}}(t)\right)}{Y_v^{\mathrm{F}}(t)} \end{aligned} \tag{16}$$

where $T_{v,m}^{F\_\mathrm{Th}}(t)$ is the maximum tolerable delay of the task in the MEC server, which is calculated by $T_{v,m}^{\mathrm{F\_Th}}(t) = T_v^{\mathrm{Th}} - T_{v,m,\mathsf{Tx}}^{\mathrm{F}}(t)$.

In the next section, online algorithms to solve both the CO and PS subproblems are designed. The one-shot solution to the original JCOPS problem will be obtained by combining computation offloading and parallel scheduling together via mobile-edge cooperative offloading-delay broadcast mechanism.

5. Proposed online DGCO-RLPS scheme for JCOPS

An online DGCO-RLPS scheme is designed to solve the JCOPS problem. As shown in Fig. 2, the DGCO-RLPS scheme consists of 1) the DGCO algorithm in mobile nodes for solving the CO subproblem, 2) the RLPS algorithm in the MEC for solving the PS subproblem, and 3) the delay broadcast mechanism to combine DGCO and RLPS for the solution of the JCOPS problem.

Specifically, as to the CO subproblem, when there is a task generated in a mobile node, the node initiates the DGCO algorithm to determine where to execute the task, by comparing the estimated delays of local computing and MEC server computing respectively, where the estimated MEC server computation delay is extracted from the broadcasting packets by the MEC server. The uploading delay is determined by the Water-filling based Transmission Rate Maximization (WTRM) algorithm. As to the PS subproblem, an intelligent algorithm, i.e., RLPS, is used in the MEC server to determine how many and which tasks to be executed in parallel, with the aim of optimizing the delay performance of the offloaded tasks.

The mobile nodes and the MEC server cooperate to provide delay guarantee for computation intensive applications by the joint DGCO-RLPS algorithm. Through the computation delay broadcasted by the MEC server, DGCO and RLPS affect each other iteratively to obtain optimal solutions for the JCOPS problem. The details of the scheme are as the following.

5.1. DGCO for computation offloading

Given the per-slot task arrival events $X_v(t)$ for $v \in \mathbb{V}$, and (10), we obtain

$$\max P(t) \sim \max \sum\nolimits_{v=1}^{V} \sum\nolimits_{m=1}^{X_v(t)} \mathbb{1}\left(T_{v,m}(t) \le T_v^{\mathrm{Th}}\right) \tag{17}$$

Accordingly, the CO subproblem as in (14a) can be equivalently written as

$$(\mathrm{CO}'): \max_{I^*} \lim_{t\to\infty} \frac{1}{T} \sum\nolimits_{t=1}^{T} \sum\nolimits_{v=1}^{V} \sum\nolimits_{m=1}^{X_v(t)} \mathbb{1}\left(T_{v,m}(t) \le T_v^{\mathrm{Th}}\right) \tag{18a}$$

$$\text{s.t.} \quad (14b), (14c) \tag{18b}$$

Let $T_{v,m}^{\min}(t) = \min\left(T_{v,m}^{\mathrm{M}}(t), T_{v,m}^{\mathrm{F}}(t)\right)$, then $\mathbb{1}\left(T_{v,m}(t) \le T_v^{\mathrm{Th}}\right) \le \mathbb{1}\left(T_{v,m}^{\min}(t) \le T_v^{\mathrm{Th}}\right)$ holds. Thus, given the evaluated local computation delay and offloading delay, we obtain $\max \mathbb{1}\left(T_{v,m}(t) \le T_v^{\mathrm{Th}}\right) = \mathbb{1}\left(T_{v,m}^{\min}(t) \le T_v^{\mathrm{Th}}\right)$, $\forall v = \mathbb{V}$ and $\forall m = \mathbb{X}_v(t)$.

Based on the above discussion, a distributed online DGCO algorithm for the computation offloading subproblem is designed. Specifically, DGCO is executed at each mobile node. When a task is generated, the node evaluates the delays for local computing and offloading based on the task size, the delay deadline and the present states, including its queue length, channel states, etc. Mobile nodes then make their offloading decisions independently according to the evaluating values. The details of DGCO is summarized in Algorithm 1.

Algorithm 1: DGCO algorithm for computation offloading

Input: S_v, $N_{v,m}^t$, B, $\{\mathcal{G}_{v,m}^n(t) : n = \{1, 2, \ldots, N_{v,m}^t\}\}$, $\mathcal{P}_{v,m}$, $Q_{v,m}(t)$, f_v^{M}, $\widetilde{T}_{v,m,\mathsf{CPU}}^{\mathrm{F}}(t)$.

Output: $I_{v,m}^*(t)$.

1 Calculate the local computation delay $T_{v,m}^{\mathrm{M}}(t)$ with (3).

2 Determine the uploading rate $R_{v,m}(t)$ and power allocation policy $\{\mathcal{P}_{v,m}(t) : n = \{1, 2, \ldots, N_{v,m}^t\}\}$ with *Algorithm 2*.

3 Calculate the uploading delay $T_{v,m,\mathsf{Tx}}^{\mathrm{F}}(t)$ with (7).

4 Evaluate the offloading delay $\widetilde{T}_{v,m}^{\mathrm{F}}(t)$ according to (5), that is,

$$\widetilde{T}_{v,m}^{\mathrm{F}}(t) = T_{v,m,\mathsf{Tx}}^{\mathrm{F}}(t) + \widetilde{T}_{v,\mathsf{CPU}}^{\mathrm{F}}(t) \tag{19}$$

where $\widetilde{T}_{v,\mathsf{CPU}}^{\mathrm{F}}(t)$ is the broadcasted MEC server computation delay.

5 **if** $\widetilde{T}_{v,m}^{F}(t) \le T_{v,m}^{M}(t)$, **then**

6 set $I_{v,m}^*(t) = 1$;

7 **else**

8 set $I_{v,m}^*(t) = 0$.

9 **end**

The uploading rate is an important factor that affects the computation offloading decision, as shown in Algorithm 1. Therefore, the WTRM algorithm (Algorithm 2) is used to find the optimal uploading rate as well as the power allocated to the multiple channels constrained to the maximum transmission power of the mobile node as in Refs. [43,44].

Algorithm 2: WTRM algorithm for power allocation

Input: $N_{v,m}^t$, B, $\{\mathcal{G}_{v,m}^n(t) : n = \{1, 2, \ldots, N_{v,m}^t\}\}$, $\mathcal{P}_{v,m}$.

Output: $R_{v,m}(t)$, $\{\mathcal{P}_{v,m}^n(t) : n = \{1, 2, \ldots, N_{v,m}^t\}\}$.

1 Allocate the initial power for channel $n = \{1, 2, \ldots, N_{v,m}^t\}$ with (20) as the following.

$$\mathcal{P}_{v,m}^n(t) = \frac{1}{N_{v,m}^t}\left(\mathcal{P}_{v,m} + \sum_{n=1}^{N_{v,m}^t} \frac{1}{\mathcal{G}_{v,m}^n(t)}\right) - \frac{1}{\mathcal{G}_{v,m}^n(t)} \tag{20}$$

2 **while** $\mathtt{length}(\mathtt{find}(\{\mathcal{P}_{v,m}^n(t)\} < 0)) > 0$ **do**

3 Find $S_n = \{\mathcal{P}_{v,m}^n(t)\} < 0$ and set $\mathcal{P}_{v,m}^n(t) = 0$ for $n \in S_n$;

4 Find $S_p = \{\mathcal{P}_{v,m}^n(t)\} > 0$, and reallocate the power for channel $n \in S_p$ with (21).

$$\mathcal{P}_{v,m}^n(t) = \frac{1}{N'}\left(\mathcal{P}_{v,m} + \sum_{n \in S_p} \frac{1}{\mathcal{G}_{v,m}^n(t)}\right) - \frac{1}{\mathcal{G}_{v,m}^n(t)} \tag{21}$$

where $N' = \mathtt{length}(S_p)$.

5 **end**

6 Find $S_u = \{\mathcal{P}_{v,m}^n(t)\} = 0$, set $w_{v,m}^n(t) = 0$ for $n \in S_u$ and set $w_{v,m}^n(t) = 1$ for $n \in \{1, 2, \ldots, N_{v,m}^t\} - S_u$;

7 Calculate the optimal data rate $R_{v,m}(t)$ for the type-v node m with (6).

5.2. RLPS for parallel scheduling

Given the high complexity of the parallel scheduling subproblem due to the NP-hard nature of the parallel scheduling decision [21], our approach in this paper is to, firstly, reduce the algorithm complexity by formulating the multi-dimensional PS subproblem into a single-dimensional Markov decision process; and then, construct a single-dimensional Markov decision model based on reinforcement learning; finally, discover optimal solution based on discrete-time least-squares policy iteration (DLSPI). The detail is presented as the following.

5.2.1. Problem transformation

With multi-core CPU resources and virtual technologies, the MEC server supports the execution of multiple tasks in parallel. We define a feasible parallel scheduling policy for the MEC server as the following.

Definition 1. (A feasible parallel scheduling policy). A V-elemental vector $\mathcal{R} = (r_1, r_2, \ldots, r_V)$ is defined as a feasible parallel scheduling policy for the MEC server if the server can simultaneously execute r_1 number of type-1 tasks, $\cdots$, and r_V number of type-V tasks, which satisfies the following CPU-cycle resource constraint.

$$\sum_{v=1}^{V} r_v L_v \le f^F \tag{22}$$

A parallel scheduling array $\mathbb{R}_{A_t \times V} = (\mathcal{R}_{a_t,v})_{A_t \times V}$, which represents a set of supportable parallel scheduling strategies with respect to the queuing lengths of various types of tasks and the capacity of the MEC server is further defined as.

Definition 2. (Parallel scheduling array). $\mathbb{R}_{A_t \times V}$ is said to be a parallel scheduling array at slot t for $t = 1, 2, \ldots, \infty$ if and only if the a_tth row vector $\mathcal{R}_{a_t} = (r_{a_t,v})_{1 \times V}$ for $a_t = 1, 2, \ldots, A_t$ is a feasible parallel scheduling policy at slot t. The parallel scheduling policy $\mathcal{R}_{a_t}$ satisfies the following constraints.

$$\begin{cases} \sum_{v=1}^{V} r_{a_t,v} L_v \le f^F \\ r_{a_t,v} \le Q_v^F(t), \qquad \forall v \in \mathbb{V} \end{cases} \tag{23}$$

where $r_{a_t,v}$ represents the number of type-v tasks scheduled in slot t under policy a_t; A_t is a numerical variable representing the number of feasible parallel scheduling policies at slot t.

With the definition of the parallel scheduling array, the parallel scheduling subproblem in (15a) becomes a single-dimensional decision-making for $a_t \in \{1, 2, \ldots, A_t\}$, $t = 1, 2, \ldots, \infty$, such that the long-term P^F is maximized.

Let $P^F(\{\mathcal{R}_{a_t}\})$ denote a function of a sequential $\mathcal{R}_{a_t}$ for $t = 1, 2, \ldots, \infty$, representing the average delay-guarantee-ratio of all tasks processed in the MEC server in the long run. Then, the PS subproblem in (15a) is transformed into

$$(\mathrm{PS}') : \max_{\mathcal{R}_{a_t}} \quad P^F(\{\mathcal{R}_{a_t}\}) \tag{24a}$$

$$\text{s.t.} \quad (15b), (15c) \tag{24b}$$

$$\mathcal{R}_{a_t} \subseteq \mathbb{R}_{A_t \times V} \tag{24c}$$

Therefore, the optimal problem of parallel scheduling in the MEC server is equivalent to determining a sequential a_t^* to maximize the long-term $P^F(\blacksquare)$. If a state is defined as an MEC environmental vector containing all useful information from its history affecting the present parallel scheduling decision a_t^*, then the sequential decisions a_t^* and corresponding $\mathcal{R}_{a_t^*}$ are indeed a Markovian Decision Process (MDP). Accordingly, reinforcement learning can be used to solve the problem described in (24a).

5.2.2. RL-based construction of task parallel scheduling

According to the reinforcement learning theory, if a reward signal is given to the MEC server, then it can discover which parallel scheduling decision yields the most rewards by training. Thus, it is possible to find out the optimal decisions that are adaptive to the dynamic of MEC environments by carefully designing a reward signal. Moreover, since the decisions based on reinforcement learning affect not only the immediate reward but also all subsequent rewards, it is possible to achieve a good long-term delay performance $P^F(\blacksquare)$ via optimal sequential decisions. In addition, decisions based on reinforcement learning can influence the subsequential environments. It is possible for an optimal parallel scheduling policy to avoid the event in which some type of task violates its delay bound due to the continual arrival of another type of tasks.

Therefore, we construct the optimization problem in (24a) based on reinforcement learning [45,46] as follow.

Features of a state. It is observed that, the values of $Q_v^F(t)$ and $P_v^F(t-1)$ for $v \in \mathbb{V}$ affect the selection behavior of action a_t. Therefore, these parameters are used as features to represent a state, i.e., $s_t = \left(\mathbb{Q}^F(t), \mathbb{P}^F(t-1)\right)$, where $\mathbb{Q}^F(t) = \{Q_v(t) : v \in \mathbb{V}\}$ and $\mathbb{P}^F(t-1) = \{P_v^F(t-1) : v \in \mathbb{V}\}$.

Decision epochs and action space. A decision epoch is defined as the beginning of a time slot. In each decision epoch, a parallel scheduling decision is made to decide which types of tasks and how many of them should be scheduled. Since the feasible scheduling policies are abstracted by the parallel scheduling array $\mathbb{R}_{A_t \times V}$, the decision is about which row vector to select from the parallel scheduling array. Equivalently, the action is to schedule the tasks in the row vector $\mathcal{R}_{a_t} = (r_{a_t,v})_{1\times V}$ for computation at slot t.

Following references [46,47], we consider a binary plane to represent the state and action features with respect to the task types for simplifying the calculation of state-action values. For example, the state of queue length and the delay-guarantee-ratio are all represented by $N \times V$ number of planes, where the integer $N > 0$, while the feasible actions are represented by A number of planes, where A is the maximum number of actions.

Notice that, by classifying the value of queue length and the delay-guarantee-ratio into N grades and then mapping them into one of N planes with respect to their task types, the state space is limited.

Reward function. To achieve the goal of maximizing the long term delay-guarantee-ratio with fast algorithm convergence, we classify the observed delay-guarantee-ratio under an action into six discrete levels {[0, 0.5), [0.5,0.6), [0.6,0.7), [0.7,0.8),[0.8,0.9), [0.9,1]}. The corresponding rewards are set to {-3, −2, −1, 1, 2, 3}, respectively.

Based on the Bellman expectation equation [45], the state-action value is obtained by $V(s,a) = \mathbb{E}[\Re_{t+1} + \gamma V(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$ and $\Re_{t+1}$ is the immediate reward of action a in state s in slot t, $V(S_{t+1}, A_{t+1})$ is the state-action value in next slot, and γ is a step-size parameter.

***ε*-greedy action policy**. The action is chosen based on the ε-greedy policy. That is, action a in state s is chosen with probability$(1 - \varepsilon/2)$, if a satisfies $a = \arg\max V(s, a)$ for $a = \{1, 2, \ldots, A_t\}$.

5.2.3. Optimal solutions

According to the ε-greedy action policy, the state-action value $V(s, a)$ is required for action selection. However, storing and updating the state-action values during a scheduling process are impractical for large state and action spaces in a dynamic scheduling environment. A parameter function approximator $\hat{V}(s,a,w)$ is introduced, to represent the state-action value function, where w is an adjustable parameter vector. Therefore, only the parameter vector w is required to store. Compared to storing $V(s, a)$, the storage requirements are explicitly reduced. Specifically, the state-action value is approximated by

$$\hat{V}(s,a,w) = x(s,a)^T w = \sum_{j=1}^{n} x_j(s,a) w_j \tag{25}$$

where $x(s, a)$ is a binary state-action feature vector, w is an adjustable parameter vector.

The desired parameter w is expected to minimize the mean-squared error between the actual state-action value and the approximated state-action value. This paper uses the DLSPI, which is a discrete-time variant of LSPI, to learn w based on the following equation.

$$w = \frac{\sum_{t=1}^{T} x(s_t,a_t)\Re_{t+1}}{\sum_{t=1}^{T} x(s_t,a_t)[x(s_t,a_t) - \gamma x(s_{t+1},a_{t+1})]^T} \tag{26}$$

The detail of the DLSPI is shown in Algorithm 3.

Once w^* is learned, the state-action value can be approximated by (25), and the optimal action can be further determined with the ε-greedy action policy. The detail of the RLPS algorithm is shown in Algorithm 4.

Algorithm 3: Learning the optimal w^* via DLSPI

Output: w^*.

initial $\widetilde{A} = 0$, $\widetilde{b} = 0$, $w = \{0\}$.
repeat
  for $t = 1$ *to* T **do**
    Sense state $s_t \leftarrow (\mathbb{Q}^F(t), \mathbb{P}^F(t-1))$;
    Calculate the available number of actions A_t and parallel scheduling array $\mathbb{R}_{A_t \times V}$ using (23);
    Select an action with ϵ-greedy, that is, a_t is chosen with probability $1 - \frac{\epsilon}{2}$ if it satisfies

$$a_t = \arg\max_{a_t \in \mathbb{A}_t} \hat{V}(s_t, a_t, w) \tag{27}$$

    Process tasks based on the selected action a_t;
    Observe reward $\Re_{t+1}$ using the rule given in Section 5.2.2;
    Observe next state s_{t+1} and select next action a_{t+1} with ϵ-greedy;
    Learn next state-action features $x(s_{t+1}, a_{t+1})$ using the method given in Section 5.2.2;
    Update $\widetilde{A}$ and $\widetilde{b}$ by

$$\begin{cases} \widetilde{A}(t) = \widetilde{A}(t-1) + x(s_t,a_t)[x(s_t,a_t) \\ \qquad\qquad - \gamma x(s_{t+1},a_{t+1})]^T \\ \widetilde{b}(t) = \widetilde{b}(t-1) + x(s_t,a_t)\Re_{t+1} \end{cases} \tag{28}$$

  end
  Calculate the parameter vector $w = \widetilde{A}^{-1}\widetilde{b}$ based on (26);
  Calculate the average *delay-guarantee-ratio* $\widetilde{P}^F$ based on (16).
until $\widetilde{P}^F \sim P^{F*}$, ***then*** *let* $w^* = w$;

5.3. Mobile-edge cooperative delay broadcast for one-shot optimization

As show in Section 4.2, the JCOPS problem is decomposed into the CO and PS subproblems. The solution of the CO subproblem is based on the solution of the PS subproblem, and vice versa. Our proposed algorithms, e.g., DGCO for the CO subproblem and RLPS for the PS subproblem, also present the mutual matching of offloading decisions in mobile nodes and parallel scheduling decision in the MEC server for one-shot optimization. Specifically, as shown in lines 10–11 of Algorithm 4, at every time slot, the MEC server calculates the time-averaged MEC server computation delay of every type of tasks based on the RLPS-based parallel scheduling decision, and then broadcasts them to the nodes located in MEC. Every mobile node makes offloading decision based on the computation delay it receives from MEC server, as shown in lines 4–9 of Algorithm 1.

Algorithm 4: RLPS algorithm for parallel scheduling

initial $w = \{0\}$, $Q_v(0) = 0$ for $v \in \mathbb{V}$.
for $t = 1$ *to* ∞ **do**
  Buffering process: All the type-v tasks arrived in slot t are buffered in the vth queue with the First-In-First-Service (FIFS) discipline. Update the queue length by

$$Q_v(t+1) = Q_v(t) + Y_v^F(t), \quad \forall v \in \mathbb{V} \quad (29)$$

  Parallel scheduling decision: In decision epoch t, **do**
    Sense $s_t \leftarrow (\mathbb{Q}^F(t), \mathbb{P}^F(t-1))$;
    Calculate A_t and $\mathbb{R}_{A_t \times V}$ using (23);
    Select $a_t^* \subseteq \mathcal{A}_t$ with (27), such that $\mathcal{R}_{a_t^*} \subseteq \mathbb{R}_{A_t \times V}$ is determined.
  Scheduling process:
    Scheduling: for $v \in \mathbb{V}$, de-queue $r_{a_t^*,v}$ tasks from queue v in a head-of-line manner and execute them. Update the queue length by

$$Q_v(t+1) = Q_v(t) - r_{a_t^*,v} \quad (30)$$

    Calculate the time-averaged MEC server computation delay of type-v tasks, $\forall v \in \mathbb{V}$, by

$$\widetilde{T}^F_{v,\mathsf{CPU}}(t) = a\widetilde{T}^F_{v,\mathsf{CPU}}(t-1) + (1-a)T^F_{v,\mathsf{CPU}}(t) \quad (31)$$

    where $a \in (0,1)$ is a weighted parameter. $T^F_{v,\mathsf{CPU}}(t) = \frac{\sum_{m=1}^{r_{a_t^*,v}} T_{v,m,\mathsf{CPU}}(t)}{r_{a_t^*,v}}$ is the average MEC server computation delay in slot t. $T_{v,m,\mathsf{CPU}}(t)$ is the actual delay.
    Broadcast $\{\widetilde{T}^F_{v,\mathsf{CPU}}(t)\}$ to the mobile nodes located in MEC.
    Calculate the time-averaged delay-guarantee-ratio of the MEC server by

$$\widetilde{P}^F(t) = a\widetilde{P}^F(t-1) + (1-a)P^F(t) \quad (32)$$

    where $P^F(t)$ is the average delay-guarantee-ratio of all types of tasks in slot t.
    If $\widetilde{P}^F(t) < \alpha P^{F*}$, then update the parameter vector w^* with *Algorithm* 3.
    Store the latest T number of task arrival information $\{Y_v^F(t-T+1), \ldots, Y_v^F(t)\}$ to replay buffer.
end

5.4. Algorithm complexity

As shown in Fig. 2, since the DGCO and RLPS algorithms operate in different edge devices (e.g., DGCO in mobile nodes and RLPS in the MEC server) for different subproblems (e.g., DGCO for the CO subproblem, and RLPS for the PS subproblem), they can work in parallel for the same task. Accordingly, similar to Ref. [21], the time complexity of the proposed DGCO-RLPS scheme is $\max[\mathcal{O}(\text{DGCO}), \mathcal{O}(\text{RLPS})]$, where $\mathcal{O}(\text{DGCO})$ and $\mathcal{O}(\text{RLPS})$ are the time complexity of the DGCO and RLSP algorithms, respectively.

It is easy to see from Algorithm 1 that, the time complexity of the DGCO algorithm is $\max[\mathcal{O}(1), \mathcal{O}(\text{WTRM}), \mathcal{O}(1)]$, where the former $\mathcal{O}(1)$ is the time complexity of obtaining the local computation delay, and the latter $\mathcal{O}(1)$ is the time complexity of evaluating the MEC server processing delay. $\mathcal{O}(\text{WTRM})$ is the time complexity of Algorithm 2, which is equivalent to $\mathcal{O}(N^{\max})$ [44], where $N^{\max} = \max[\max[N^t_{v,m} : t = 1, 2, \ldots, \infty] : v \in \mathbb{V}, m \in \mathbb{X}_v(t)]$ is the maximum number of subbands available to the mobile node that has the maximum available subbands in MEC. Accordingly, the time complexity of the DGCO algorithm is $\mathcal{O}(N^{\max})$.

As shown in Algorithm 4, the time complexity of the RLPS algorithm is mainly determined by the parallel scheduling decision making process. The time complexity of the parallel scheduling decision making process is further determined by (23) and (27). Since (23) could operate offline by storing the mapping between various available computing resource and action space, the time complexity of the RLPS algorithm is $\mathcal{O}(A)$ according to (27), where A is the number of feasible parallel scheduling policies in the MEC.

Finally, the time complexity of the proposed DGCO-RLPS scheme is $\max[\mathcal{O}(N^{\max}), \mathcal{O}(A)]$.

It is noted that, in most of existing proposals (e.g., Refs. [26,27,40]), the queue number in the MEC server equals to the number of mobile nodes, such that their parallel scheduling complexity would be at least $\mathcal{O}(U \times r_u^{\max})$, where U is the number of mobile nodes in MEC, $r_u^{\max} = \min[\lfloor f^{\mathrm{F}}/L_u \rfloor, Q_u(t)]$ is the maximum number of tasks from the uth $(0 < u \le U)$ queue that can be processed in parallel. The difference is that the parallel scheduling complexity $\mathcal{O}(A)$ is a function of V, where V is the number of task types, which is generally far smaller than the value of U, since we divide the computation tasks as well as mobile nodes into a number of types based on the task data size and tolerable latency. Accordingly, our algorithm complexity is quite lower than existing approaches.

Table 2
The basic parameter settings.

	Parameters	Type-1	Type-2	Type-3
Mobile node	Number of mobile nodes	50	30	10
	CPU rate $f^{\mathrm{M}}_{v,m}$ (MHz)	600		
Task	Mean data size S (KB/task)	200	250	420
	Mean CPU cycle requirement L (MHz/task)	320	475	882
	Delay deadline T^{Th} (ms)	3	5	10
Wireless network	Number of subbands	[1,16]		
	Subband bandwidth (MHz)	0.3125		
	Transmission power $\mathcal{P}_{v,m}$ (W)	0.1		
	Fading channel	Rayleigh		
	Channel noise density	1e-11		
Server	CPU rate f^{F} (GHz)	10		

6. Performance evaluation

This section evaluates the efficiency in delay guarantee of the DGCO-RLPS scheme. Since the delay-sensitive applications can be classified into a limited number of types according to the expected latency [42,48], we consider the scenario with three types of mobile nodes according to the types of tasks they generate. As listed in Table 2, type-1 task with delay deadline of 3 ms represents autonomous vehicles/industrial Internet, 5 ms delay deadline for the type-2 task with respect to gaming/factory

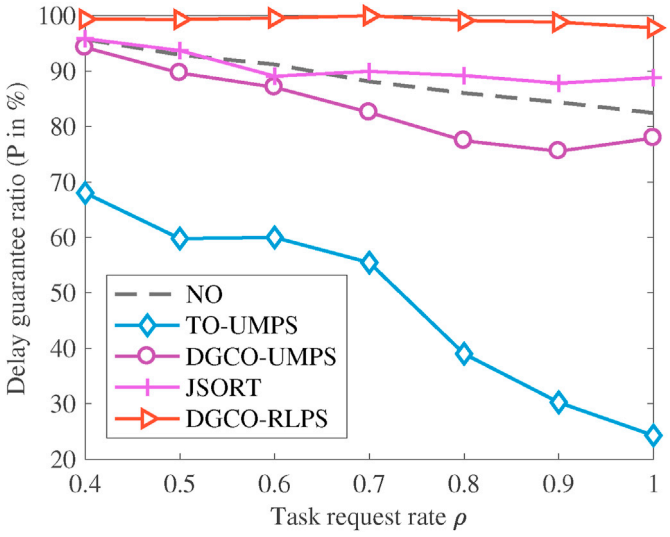

Fig. 3. Delay-guarantee-ratio with respect to task request rate.

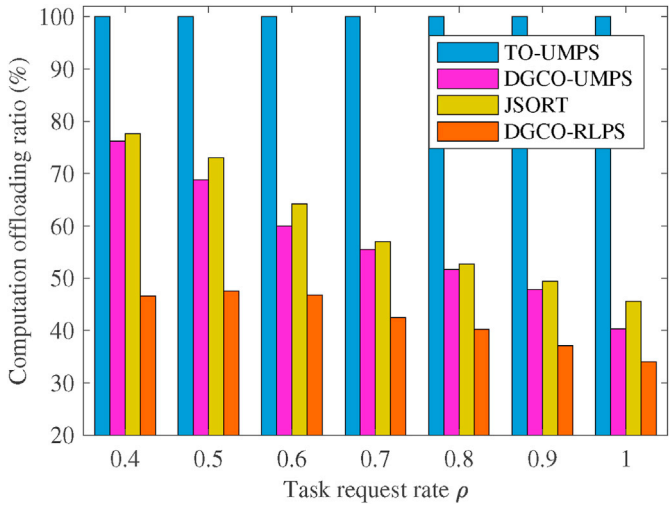

Fig. 4. Computation offloading ratio (defined as the ratio of offloaded tasks to all tasks) with respect to task request rate.

automation, and type-3 task mainly represents smart grid with a delay deadline of 10 ms.

The mobile nodes are randomly distributed over the network. For simplification, it is assumed that the per-slot task request probabilities of a mobile node follow the same i.i.d. Let $\rho = (0.5, 0.5, 0.5)$ tasks/ms/node be the basic per-slot task request rate vector for the three types of tasks. Then, the request rate of type-v tasks can be derived by $\rho_v = \frac{1}{M_v}\sum_{m=1}^{M_v} \lim_{t\to\infty} \frac{1}{t}\sum_t A_{v,m}(t)$ for $v \in \mathbb{V}$ and $M_v \in \mathbb{M}$. In order to satisfy the IoT applications' ultra-low latency requirement, the transmission delay should be less than 1 ms [49]. With the novel multiple-access techniques in next generation wireless system [50], it is reasonable for us to set the number of subbands available to any mobile node to a range of [1,16]. Hence the uploading delay of a large task (e.g., a type-3 task) could be less than 1 ms, such that offloading decision makes sense. The detail of parameter settings is listed in Table 2.

The performance of DGCO-RLPS is compared with four benchmarked schemes, of those most relevant to our work: (1) No Offloading (NO) [51]; (2) joint Total-Offloading (TO) [51] and Utility-Maximum Parallel Scheduling (UMPS), called as TO-UMPS; (3) DGCO-UMPS; (4) Joint optimization algorithm of Stochastic computation Offloading, Resource allocation, and Trajectory scheduling (JSORT) [27], under various task intensities and various MEC server computing capabilities with a discrete event-based simulator that combines Matlab and C++.

- *NO*: All tasks are executed in local mobile nodes in the FIFS discipline [51].
- *TO-UMPS*: Under TO Ref. [51], all tasks are offloaded to the MEC server for computing. That is, there is no cooperation among mobile nodes and the MEC server. The MEC server schedules the offloaded tasks with UMPS, which is a variant of min-min best fit in Ref. [21]. Specifically, under the UMPS algorithm, the parallel scheduling policy that satisfies $a_t = \arg\max_{a_t\in\mathbb{A}_t} \mathcal{R}_{a_t} L$ is chosen.
- *DGCO-UMPS*: The delay-greedy algorithm is adopted to make computation offloading decisions in mobile nodes considering the cooperation among mobile nodes and the MEC server, and the UMPS policy is used as the parallel scheduling algorithm in the MEC server.
- *JSORT*: Based on Lyapunov drift [52], the variant of JSORT [27] is adopted to make computation offloading decisions in mobile nodes and parallel scheduling decisions in the MEC server. Specifically, under JSORT, the offloading decision vector $\{I^*_{v,m}(t)\}$ for $v \in \mathbb{V}$ and $m \in \mathbb{X}_v(t)$ in slot t is chosen if it minimizes the drift of the task queue length; similarly, a feasible parallel scheduling policy $\mathcal{R}^*(t)$ is chosen if it minimizes the drift plus penalty.

6.1. Impact of task request rate

We first evaluate the performance of DGCO-RPLS under various task intensities. We set $\rho = \rho_1 = \rho_2 = \rho_3$, that is, the per-slot task request rates of all mobile nodes are identical. It is noticed that, the request rates for different types of tasks are different due to different number of mobile nodes. Then, we vary ρ from 0.4 to 1, evaluating how it affects the delay performance of the investigated schemes.

As shown in Fig. 3, the delay-guarantee-ratio given by our proposed DGCO-RLPS outperforms the other schemes by providing the highest delay-guarantee-ratio under various task request rates. Particularly, when the workload is heavy (e.g., $\rho = 1$), the delay-guarantee-ratio given by DGCO-RLPS can still approximate 98%, while the worst one given by the TO-UMPS is less than 30%.

The delay-guarantee-ratio given by TO-UMPS decreases explicitly with the increasing task request rate, as shown in Fig. 3. Because TO-UMPS offloads all tasks to the MEC server, leading to a large transmission delay and heavy computation loads in the MEC server. Accordingly, the MEC queuing delay would increase quickly with the increasing task request rate due to limited MEC computing resource.

Similarly, due to the limitation of computation resource, the queuing delay under the NO scheme increases explicitly with the increasing task request rate. As a result, the delay-guarantee-ratio given by NO decreases with the increasing task request rate, as shown in Fig. 3. DGCO-UMPS provides a worse delay-guarantee-ratio than both DGCO-RLPS and NO under various task request rates. Because UMPS always chooses a parallel scheduling policy that maximize the resource utility. This may starve some types of delay-sensitive tasks, leading to more local computation in mobile nodes, as shown in Fig. 4. Therefore, these tasks might experience long local queuing delay.

The JSORT algorithm outperforms the NO, TO-UMPS and DGCO-UMPS algorithms by providing higher delay-guarantee-ratio under various task request rates. Because JSORT adapts to the varying of task request rate by controlling the drift of queue length, thus avoiding extremely long queuing delay. However, since it is difficult to accurately tradeoff between queue length and delay-guarantee-ratio, the delay-guarantee-ratio given by the JSORT algorithm may be suboptimal, thus it is unsurprising to see that the delay-guarantee-ratio given by JSORT is worse than that of our proposed DGCO-RLPS.

6.2. Impact of the number of mobile nodes

We next investigate the delay-guarantee-ratio under various number of mobile nodes. We set $\rho = (1.0, 1.0, 1.0)$ tasks/ms/node. In the first

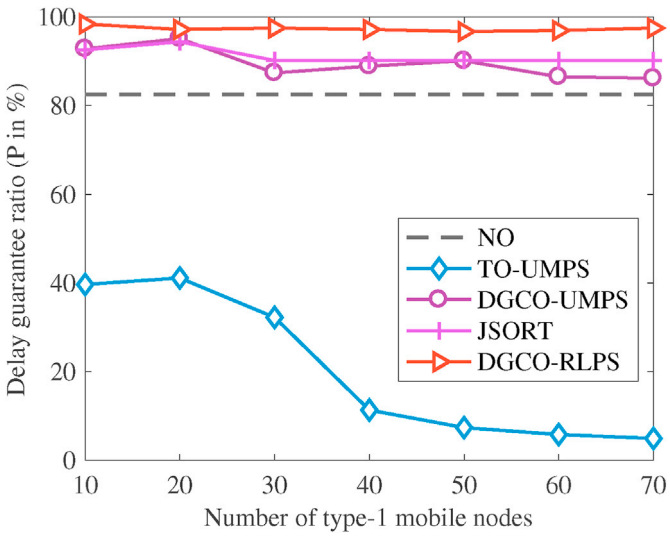

Fig. 5. *Delay-guarantee-ratio* with respect to type-1 mobile node number.

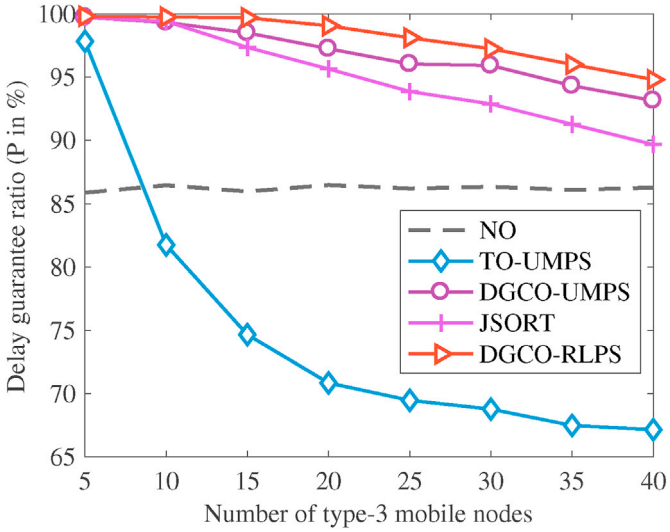

Fig. 6. Delay-guarantee-ratio with respect to type-3 mobile node number.

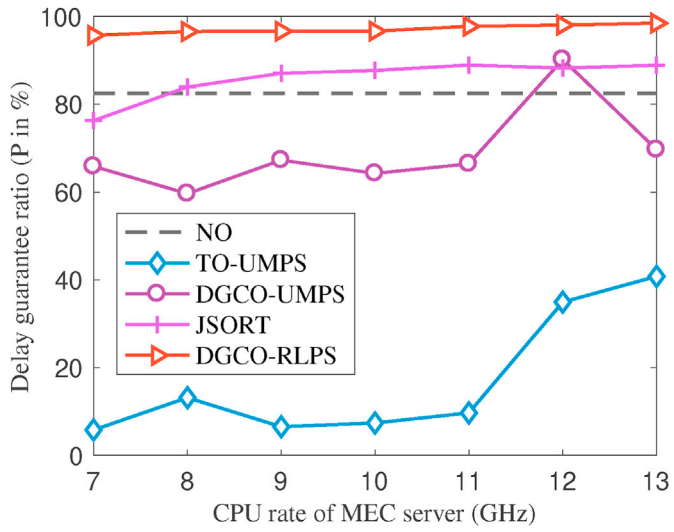

Fig. 7. Delay-guarantee-ratio with respect to computing capability of the MEC server.

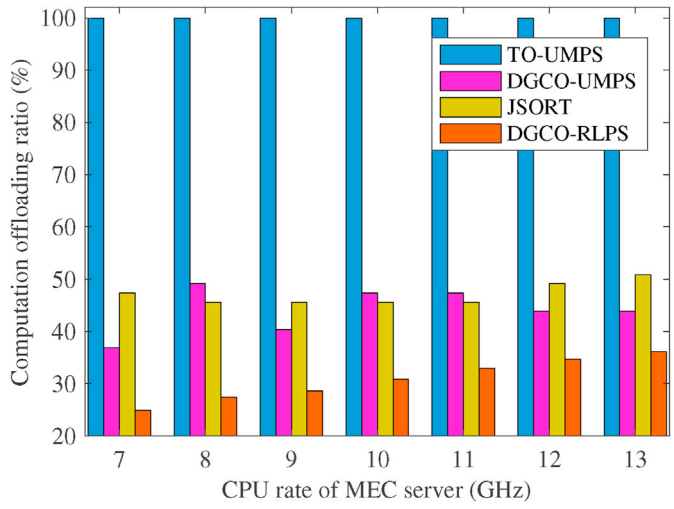

Fig. 8. Computation offloading ratio with respect to computing capability of the MEC server.

scenario, we set the number of type-2 and type-3 nodes to 30 and 10, respectively, then we change the number of type-1 nodes to evaluate how it affects the delay guarantee performance of the investigated schemes. In the second scenario, we set the number of type-1 and type-2 mobile nodes both to 10, then change the number of type-3 nodes to evaluate how it affects the delay guarantee performance of the investigated schemes. The other parameter settings are listed in Table 2.

As shown in Fig. 5, the delay-guarantee-ratio given by DGCO-RLPS, JSORT, DGCO-UMPS and NO decrease slowly with the increasing number of type-1 nodes. Because the tasks of the CPU cycle required from type-1 nodes are quite small compared to the CPU rate of local type-1 node. Thus, under the DGCO-RLPS, JSORT, DGCO-UMPS and NO schemes, the resource competition in the MEC server increases slowly with the increasing number of type-1 nodes as more type-1 tasks need to perform local computing. However, since TO-UMPS is a scheme that all tasks will be offloaded to the MEC server for computing, the resource competition in the MEC server increases quickly with the increasing number of type-1 nodes. Thus, TO-UMPS provides the worst performance by giving the lowest delay-guarantee-ratio under various number of type-1 nodes. For example, the delay-guarantee-ratio given by TO-UMPS decreases from 39.6% to 4.9% when the number of type-1 mobile nodes increases from 10 to 70. However, under our proposed DGCO-RLPS, the corresponding delay-guarantee-ratio only decreases from 98.4% to 97.5%.

Since the CPU cycle requirement of a type-3 task is considerable large compared to the CPU rate of a type-3 mobile node, under any investigated scheme, the computation resource competition increases quickly with the increasing number of type-3 nodes. Thus, the delay-guarantee-ratios given by all the investigated schemes decrease explicitly, as shown in Fig. 6. Particularly, under TO-UMPS, since all tasks are offloaded to the MEC server, the delay-guarantee-ratio decreases the fastest, e.g., the delay-guarantee-ratio given by TO-UMPS decreases from 97.8% to 67.1% when the number of type-3 nodes increases from 5 to 40. However, under our proposed DGCO-RLPS, the delay-guarantee-ratio could still approximate 95% when the number of type-3 nodes reaches 40, as illustrated in Fig. 6.

The simulation results in Figs. 5–6 demonstrate that, under a slightly heavy number of mobile nodes generated a slightly massive of tasks, our proposed DGCO-RLPS scheme could still provide a high delay-guarantee-ratio (e.g., $\geq$95%) to computation tasks, which demonstrates the effectiveness of DGCO-RLPS for delay guarantee adapting to mobile node numbers.

Table 3
Overhead analysis.

Task request rate ρ ($f^F = 7$ GHz)	Delay-guarantee-ratio (%)				Overhead(bytes/slot)			
	Dedicated			Piggyback	Dedicated			Piggyback
	period = 1 slot	period = 2 slots	period = 4 slots		period = 1 slot	period = 2 slots	period = 4 slots	
0.4	98.82	98.81	95.21	97.34	13.3	7	3.5	49.89
0.5	99.29	96.94	91.18	83.5	13.3	7	3.5	60.2
0.6	98.75	94.7	89.52	77.35	13.3	7	3.5	65.79
0.7	97.21	93.15	86.4	74.95	13.3	7	3.5	70.88
0.8	96.76	91.97	83.62	74.07	13.3	7	3.5	82.32
0.9	96.29	91.18	82.74	71.81	13.3	7	3.5	96.87
1.0	95.52	89.49	79.89	73.8	13.3	7	3.5	102.08

6.3. Impact of computing capability

In this scenario, the delay-guarantee-ratio under various computing capabilities of the MEC server is investigated. We set $\rho = (1.0, 1.0, 1.0)$ tasks/ms/node as the task request rate vector. Then we change the computing capability of the MEC server to evaluate how it affects the delay performance of the investigated schemes. Under NO, no task is offloaded to the MEC server for computing, so the delay-guarantee-ratio given by the NO scheme does not vary with the CPU rates of the MEC server, as shown in Fig. 7.

The larger the computing capability, the shorter the delay a task would experience in the MEC server. Hence the number of tasks offloaded to the MEC server would increase with the increasing MEC server computing capability under the delay-aware mobile-edge cooperative schemes, e.g., DGCO-UMPS, JSORT and DGCO-RLPS, as shown in Fig. 8, resulting in the decrease of the queuing delays in both local and offloading computing. Therefore, the delay-guarantee-ratios given by DGCO-UMPS, JSORT and DGCO-RLPS increase with the increasing MEC server computing capability. It is noted that, under UMPS, some types of delay-sensitive tasks might be starved, so the average delay-guarantee-ratio given by both TO-UMPS and DGCO-UMPS may not increase with the increasing MEC server computing capability when the CPU rate is smaller than 11 GHz.

It is observed that our proposed scheme outperforms state-of-the-art in terms of delay-guarantee-ratio under various MEC server computing capabilities. The main reason is that the proposed DGCO-RLPS scheme can dynamically switch between local and MEC server offloading based on dependency between offloading decisions and scheduling priorities in the MEC server. More interestingly, the proposed method can fully guarantee the delay-guarantee-ratio when the CPU rate of the MEC server goes beyond 13 GHz based on the current simulation parameter settings.

6.4. Message overhead

As shown in Fig. 2, the behavior of broadcasting the MEC server computation delay from the MEC server to mobile nodes via the wireless network consumes communication overhead. Generally, the accurateness of MEC computation delay estimation increases with the increasing broadcast frequency, and the preciseness of computation offloading decision in mobile node increases with the increasing accuracy of MEC computation delay estimation. However, the communication overhead also increases with the broadcast frequency. Therefore, we compare the overhead and delay-guarantee-ratio of the following two MEC server computation delay broadcast approaches.

- *Dedicated broadcast.* There is a dedicated packet (called as delay update packet) carrying the MEC server computation delay.
- *Piggyback broadcast.* The MEC server computation delay is fed back to a mobile node along with the computation results of the task offloaded from that node.

In the dedicated broadcast approach, it is assumed that the delay update packet is broadcasted over the wireless network via User Datagram Protocol (UDP) [53], then it consumes 8 bytes of UDP header. It is assumed that the overhead of each delay value is 2 bytes, then the delay update packet consumes extra overhead of $2V$, where V is the number of task types. Accordingly, the overhead of a delay update packet is $8 + 2V$ per broadcast period. In the piggyback broadcast approach, the extra overhead is 2 bytes per task per mobile node. As to dedicated broadcast, we set the broadcasting interval to 1 slot, 2 slots and 4 slots, and observe the performance under these intervals. In piggyback broadcast, once a task is finished, the MEC server computation delay is fed back.

From the results summarized in Table 3, it is observed that the dedicated broadcast approach with the period of 1 slot yields the highest delay-guarantee-ratio under various task request rates. Moreover, the delay-guarantee-ratio decreases with the increasing broadcast interval under the same traffic condition. The piggyback broadcast approach gives the lowest delay-guarantee-ratio and consumes the highest overhead. Because under the piggyback broadcast mechanism, a mobile node might not catch up with the up-to-date MEC server computation delay if its task request rate is low. In addition, the overhead would increase quickly with the increasing task request rate under piggyback broadcast. Therefore, it is better to adopt dedicated broadcast in DGCO-RLPS. However, the trade-off between QoS provisioning efficiency and overhead consumption can be regulated by adjusting the broadcasting interval of the delay update packet.

7. Conclusion

This paper studies a delay-based joint computation offloading and parallel scheduling to maximize delay-guarantee-ratio for computation intensive applications in a cooperative MEC system. We have decomposed the above problem into two concatenated subproblems, e.g., computation offloading and parallel scheduling. Afterward, we solve the problem of joint computation offloading and parallel scheduling through the exploitation of (a) the dependency between offloading decisions for the computation offloading subproblem in mobile nodes and (b) the scheduling priorities of offloaded tasks for the parallel scheduling subproblem in an MEC server. The proposed cooperative offloading and parallel scheduling scheme, which is DGCO-RLPS, outperforms state-of-the-arts by providing high delay-guarantee-ratio to computation tasks even in slightly heavy task load situation while consuming acceptable communication overhead. Our future work is to derive the lower bound of delay-guarantee-ratio with a given task model and to extend the proposal to an MEC system with multiple heterogeneous MEC servers with different computational capabilities.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61901128 and 62273109, the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (21KJB510032).

References

[1] D. Gupta, S. Rani, S.H. Ahmed, S. Verma, M.F. Ijaz, J. Shafi, Edge caching based on collaborative filtering for heterogeneous ICN-IoT applications, Sensors 21 (16) (2021) 5491.
[2] J. Guo, X. Ding, W. Wu, A blockchain-enabled ecosystem for distributed electricity trading in smart city, IEEE Internet Things J. 8 (3) (2021) 2040–2050.
[3] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, K. Huang, Toward an intelligent edge: wireless communication meets machine learning, IEEE Commun. Mag. 58 (1) (2020) 19–25.
[4] G. Premsankar, M. Di Francesco, T. Taleb, Edge computing for the internet of things: a case study, IEEE Internet Things J. 5 (2) (2018) 1275–1284.
[5] S. Rani, D. Koundal, Kavita, M.F. Ijaz, M. Elhoseny, M.I. Alghamdi, An optimized framework for WSN routing in the context of industry 4.0, Sensors 21 (19) (2021) 6474.
[6] J. Guo, X. Ding, W. Wu, Reliable traffic monitoring mechanisms based on blockchain in vehicular networks, IEEE Trans. Reliab. 71 (3) (2021) 1219–1229.
[7] A.V. Dastjerdi, R. Buyya, Fog computing: helping the internet of things realize its potential, Computer 49 (8) (2016) 112–116.
[8] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J.P. Jue, All one needs to know about fog computing and related edge computing paradigms: a complete survey, J. Syst. Architect. 98 (2019) 289–330.
[9] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.
[10] Y.C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, Mobile Edge Computing a Key Technology towards 5G, ETSI, White Paper, 2015, pp. 1–-16.
[11] S. Wang, T. Tuor, T. Salonidis, K.K. Leung, C. Makaya, T. He, K. Chan, Adaptive federated learning in resource constrained edge computing systems, IEEE J. Sel. Area. Commun. 37 (6) (2019) 1205–1221.
[12] R. Deng, R. Lu, C. Lai, T.H. Luan, H. Liang, Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption, IEEE Internet Things J. 3 (6) (2016) 1171–1181.
[13] Y. Miao, G. Wu, M. Li, A. Ghoneim, M. Al-Rakhami, M.S. Hossain, Intelligent task prediction and computation offloading based on mobile-edge cloud computing, Future Generat. Comput. Syst. 102 (2020) 925–931.
[14] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: concept and applications, ACM Trans. Intell. Syst. Technol. 10 (2) (2019) 1–19, 12.
[15] S.M.A. Oteafy, H.S. Hassanein, Leveraging tactile internet cognizance and operation via IoT and edge technologies, Proc. IEEE 107 (2) (2019) 364–375.
[16] A. Alnoman, S.K. Sharma, W. Ejaz, A. Anpalagan, Emerging edge computing technologies for distributed IoT systems, IEEE Network 33 (6) (2019) 140–147.
[17] X. Ding, J. Guo, D. Li, W. Wu, An incentive mechanism for building a secure blockchain-based internet of things, IEEE Trans. Netw. Sci. Eng. 8 (1) (2021) 477–487.
[18] S. Shalev-Shwartz, S. Ben-David, Understanding Machine Learning: from Theory to Algorithms, Cambridge university press, 2014.
[19] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016. http://www.deeplearningbook.org.
[20] H. B. McMahan, E. Moore, D. Ramage, B. A. y Arcas, Federated Learning of Deep Networks Using Model Averaging, CoRR abs/1602.05629. arXiv:1602.05629.
[21] M. Guo, Q. Guan, W. Chen, F. Ji, Z. Peng, Delay-optimal scheduling of VMs in a queueing cloud computing system with heterogeneous workloads, IEEE Tran. Serv. Comput. 15 (1) (2022) 110–123.
[22] T. Zhao, S. Zhou, X. Guo, Z. Niu, Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing, in: IEEE International Conference on Communications (ICC), IEEE, 2017, pp. 1–7.
[23] S.E. Mahmoodi, R.N. Uma, K.P. Subbalakshmi, Optimal joint scheduling and cloud offloading for mobile applications, IEEE Tran. Cloud Comput. 7 (2) (2019) 301–313.
[24] Z. Ning, X. Wang, J.J.P.C. Rodrigues, F. Xia, Joint computation offloading, power allocation, and channel assignment for 5G-enabled traffic management systems, IEEE Trans. Ind. Inf. 15 (5) (2019) 3058–3067.
[25] Y. Yu, J. Zhang, K.B. Letaief, Joint subcarrier and CPU time allocation for mobile edge computing, in: IEEE Global Communications Conference (GLOBECOM), IEEE, 2016, pp. 1–6.
[26] T.X. Tran, D. Pompili, Joint task offloading and resource allocation for multi-server mobile-edge computing networks, IEEE Trans. Veh. Technol. 68 (1) (2019) 856–868.
[27] J. Zhang, L. Zhou, Q. Tang, E.C.-H. Ngai, X. Hu, H. Zhao, J. Wei, Stochastic computation offloading and trajectory scheduling for UAV-assisted mobile edge computing, IEEE Internet Things J. 6 (2) (2019) 3688–3699.
[28] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (5) (2016) 78–81.
[29] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39.
[30] M. Mukherjee, S. Kumar, C.X. Mavromoustakis, G. Mastorakis, R. Matam, V. Kumar, Q. Zhang, Latency-driven parallel task data offloading in fog computing networks for industrial applications, IEEE Trans. Ind. Inf. 16 (9) (2020) 6050–6058.
[31] D. Van Le, C. Tham, A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds, in: Proceedings of the 2018 IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS), 2018, pp. 760–765.
[32] Q. Qi, J. Wang, Z. Ma, H. Sun, Y. Cao, L. Zhang, J. Liao, Knowledge-driven service offloading decision for vehicular edge computing: a deep reinforcement learning approach, IEEE Trans. Veh. Technol. 68 (5) (2019) 4192–4203.
[33] Z. Tong, X. Deng, F. Ye, S. Basodi, X. Xiao, Y. Pan, Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment, Inf. Sci. 537 (2020) 116–131.
[34] J. Liu, Q. Zhang, Offloading schemes in mobile edge computing for ultra-reliable low latency communications, IEEE Access 6 (2018) 12825–12837.
[35] L. Yang, B. Liu, J. Cao, Y. Sahni, Z. Wang, Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds, in: Proceedings of the 10th IEEE International Conference on Cloud Computing (CLOUD), IEEE, 2017, pp. 246–253.
[36] M. Guo, L. Li, Q. Guan, Energy-efficient and delay-guaranteed workload allocation in IoT-edge-cloud computing systems, IEEE Access 7 (2019) 78685–78697.
[37] A. Khairy, H.H. Ammar, R. Bahgat, Smartphone energizer: extending smartphone's battery life with smart offloading, in: Proceedings of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC), IEEE, 2013, pp. 329–336.
[38] M. Hu, L. Zhuang, D. Wu, Y. Zhou, X. Chen, L. Xiao, Learning driven computation offloading for asymmetrically informed edge computing, IEEE Trans. Parall. Distr. 30 (8) (2019) 1802–1815.
[39] J. Liu, Y. Mao, J. Zhang, K.B. Letaief, Delay-optimal computation task scheduling for mobile-edge computing systems, in: Proceedings of 2016 IEEE International Symposium on Information Theory (ISIT), IEEE, 2016, pp. 1451–1455.
[40] S. C. G, V. Chamola, C.-K. Tham, G. S, N. Ansari, An optimal delay aware task assignment scheme for wireless SDN networked edge cloudlets, Future Generat. Comput. Syst. 102 (2020) 862–875.
[41] R. Uhlig, G. Neiger, D. Rodgers, A.L. Santoni, F.C.M. Martins, A.V. Anderson, S.M. Bennett, A. Kagi, F.H. Leung, L. Smith, Intel virtualization technology, Computer 38 (5) (2005) 48–56.
[42] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, T. Taleb, Survey on multi-access edge computing for internet of things realization, IEEE Commun. Surv. Tut. 20 (4) (2018) 2961–2991.
[43] W. Yu, W. Rhee, S. Boyd, J.M. Cioffi, Iterative water-filling for Gaussian vector multiple-access channels, IEEE Trans. Inf. Theor. 50 (1) (2004) 145–152.
[44] C. Xing, Y. Jing, S. Wang, S. Ma, H.V. Poor, New viewpoint and algorithms for water-filling solutions in wireless communications, IEEE Trans. Signal Process. 68 (2020) 1618–1634.
[45] R.S. Sutton, A.G. Barto, Reinforcement Learning: an Introduction, MIT Press, Cambridge, MA, USA, 1998.
[46] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, Science 362 (6419) (2018) 1140–1144.
[47] D. Silver, A. Huang, C.J. Maddison, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (2016) 484–489.
[48] H.A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, C. Assi, Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing, IEEE J. Sel. Area. Commun. 37 (3) (2019) 668–682.
[49] M. Mukherjee, M. Guo, J. Lloret, Q. Zhang, Leveraging intelligent computation offloading with fog/edge computing for tactile internet: advantages and limitations, IEEE Network 34 (5) (2020) 322–329.
[50] H. Tataria, M. Shafi, A.F. Molisch, M. Dohler, H. Sjöland, F. Tufvesson, 6G wireless systems: vision, requirements, challenges, insights, and opportunities, Proc. IEEE 109 (7) (2021) 1166–1199.
[51] Z. Xiao, X. Dai, H. Jiang, D. Wang, H. Chen, L. Yang, F. Zeng, Vehicular task offloading via heat-aware MEC cooperation using game-theoretic method, IEEE Internet Things J. 7 (3) (2020) 2038–2052.
[52] M.J. Neely, Stochastic Network Optimization with Application to Communication and Queueing Systems, Morgan and Claypool Publishers, 2010.
[53] J. Kurose, K. Ross, Computer Networking: A Top-Down Approach 7th, Pearson Education, Boston, MA, 2016.