Task scheduling for mobile edge computing enabled crowd sensing applications

Jingya Zhou*

School of Computer Science and Technology,
Soochow University,
Suzhou, Jiangsu, 215006, China

and

State Key Laboratory of Mathematical Engineering
and Advanced Computing,
Wuxi, Jiangsu, 214125, China
Email: jy_zhou@suda.edu.cn
*Corresponding author

Jianxi Fan and Jin Wang

School of Computer Science and Technology,
Soochow University,
Suzhou, Jiangsu, 215006, China
Email: jxfan@suda.edu.cn
Email: wjin1985@suda.edu.cn

Abstract: Crowd sensing effectively solves the dilemma of massive data collection faced by most data-driven applications. Recently, mobile edge computing (MEC) is proposed to extend the frontier of cloud to the network edge so that it is quite suitable to integrate MEC with current crowd sensing systems. In this paper, we focus on the basic problem of task scheduling in such systems. The problem discussed here has some unique challenges, e.g., edge devices are not dedicated to perform sensing tasks, task scheduling on edge devices and edge servers are highly coupled, and it is hard to achieve long-term objectives. To this end, we first present a workflow framework that captures the unique execution logic of sensing tasks. Then we propose a staged scheme to decouple the original scheduling problem. Moreover, we leverage Lyapunov optimisation technique to achieve long-term objective. The experiment results verify the effectiveness and efficiency of our proposed algorithm.

Keywords: task scheduling; crowd sensing; MEC; mobile edge computing; task offloading; task shifting; Lyapunov optimisation.

Reference to this paper should be made as follows: Zhou, J., Fan, J. and Wang, J. (2021) 'Task scheduling for mobile edge computing enabled crowd sensing applications', *Int. J. Sensor Networks*, Vol. 35, No. 2, pp.88–98.

Biographical notes: Jingya Zhou received his BS in Computer Science from Anhui Normal University, Wuhu, in 2005, and his PhD in Computer Science from Southeast University, Nanjing, in 2013. He is currently an Associate Professor with the School of Computer Science and Technology, Soochow University, Suzhou. His research interests include cloud and edge computing, parallel and distributed systems, online social networks, network representation learning and data center networking.

Jianxi Fan received his BS in Computer Science from Shandong Normal University, Jinan, in 1988, his MS in Computer Science from Shandong University, Jinan, in 1991, and his PhD degree in Computer Science from City University of Hong Kong, Hong Kong, in 2006. He is currently a Professor with the School of Computer Science and Technology, Soochow University, Suzhou. His research interests include interconnection architectures, the design and analysis of algorithms, and graph theory.

Jin Wang received his BS from Ocean University of China, in 2006, and the PhD in Computer Science jointly awarded by City University of Hong Kong and University of Science and Technology of China, in 2011. He is currently a Professor with the School of Computer Science and Technology, Soochow University, Suzhou. His research interests include network coding and edge computing.

This paper is a revised and expanded version of a paper entitled 'Task offloading for social sensing applications in mobile edge computing' presented at *CBD 2019*, Suzhou, China, 21–22 September, 2019.

1 Introduction

Nowadays, the advance of next generation cellular communication techniques (e.g., 5G) has significantly promoted the development of Internet of Things (IoT) (Gubbi et al., 2013; Chen et al., 2016b). Dense network provides pervasive network coverage, and enables more and more devices connected to the Internet, such as smartphones, fitness trackers, drones, and even cars. The prevalence of IoT also promotes the rise of crowd sensing (Liu et al., 2018b; Capponi et al., 2019). Following the idea of collecting real-time information from the physical world by humans or mobile devices on behalf of them (Dai et al., 2016), crowd sensing extracts valuable information in a crowd computing fashion and applies them to many application scenarios. Currently crowd sensing has extremely flourished crowd computing and IoT applications in various domains. For example, in traffic management domain, the traffic control mainly relies on traffic data collected from roadside surveillance equipments (most of them reside on traffic intersections), but the data collected via this traditional method are sparse and cannot seamlessly cover all sections. Crowd sensing allows us to get real-time traffic congestions by analysing the trajectories of individual drivers and those drivers are able to record trajectories via their smartphones or dash cams (Ilarri et al., 2014). In environmental protection domain, traditional methods detect air quality from fixed detection points, the detection range is limited and cannot realise full coverage with fine granularity. Crowd sensing uses mobile devices to collect more and richer sensing data (Devarakonda et al., 2013). In the same way, crowd sensing can also be applied to disaster relief domain by detecting the real-time disaster situation based on the data collected from unmanned aerial vehicles such as drones (Wang et al., 2020).

There are two main components inside a crowd sensing system: a set of crowd sensing devices and a group of crowd sensing servers. Each component is designed with a specific function, e.g., crowd sensing devices are usually owned by people and can be used to collect data from physical world, while crowd sensing servers are available for further data analysis. Considering that the huge computing power of a cloud datacenter can help crowd sensing providers to deal with challenges derived from big sensing data and high computational complexity, those analysis tasks are often finished within a cloud datacenter. For example, Xiao et al. (2014) presented a cloud-based framework that enables a group of exciting sensing-oriented applications. However, the cloud-based system architecture may not be ideal for crowd sensing applications, since it occupies a huge amount of backbone bandwidth resources for data transmission between mobile devices and cloud datacenter. Moreover, it cannot provide low-latency guarantee for many real-time applications. As a promising computing paradigm, mobile edge computing (MEC) (Satyanarayanan et al., 2015; Shi et al., 2016; Liu et al., 2020) extends the cloud to the edge of networks so that task computing can be performed at the edge of clouds that are close to the data source. Following this idea by deploying a crowd sensing application in MEC environments, many computation tasks can be performed locally on the edge including edge devices (be interchangeable with mobile devices and user devices) and edge servers, so that the valuable backbone bandwidth can be significantly saved and the response latency will be shortened accordingly.

In a hybrid crowd sensing system, there are multiple types of resources available for performing crowd sensing tasks, e.g., cloud resources like cloud servers and edge resources like edge devices and edge servers. The most important problem is to find a suitable way to schedule tasks among those resources. Though many work have explored task offloading toward edge computing (Lin et al., 2019), few of them can be applied directly to solve the task scheduling problem in the crowd sensing scenario. Specifically, current work mainly aims to explore scheduling mechanisms that can efficiently assign tasks from end users to edge servers with a focus on performing tasks on edge servers, while tasks of crowd sensing applications are often issued by the application provider. These tasks are firstly assigned to edge devices for data collection and pre-processing, and then be sent to edge servers for further processing. There are three major challenges as we are dealing with task scheduling in a crowd sensing system:

1 Different from edge servers that are set for a specific application purpose, edge devices are usually owned by individuals, and these private devices are non-dedicated for executing sensing tasks.

2 Before returning results back to the cloud, sensing tasks need to be performed on both edge devices as well as edge servers. This execution logic is unique in the edge-enabled crowd sensing system, where task scheduling on both types of resources is highly coupled.

3 Moreover, most crowd sensing applications have to perform tasks cyclically. The long-term objective of task scheduling makes the problem more challenging.

To address these challenges, we investigate the task scheduling for MEC-enabled crowd sensing applications with an emphasis on the cost-efficient edge resources utilisation that is less explored but more compelling from the perspective of a crowd sensing service provider. We first present a MEC-enabled crowd sensing service system framework, and divide the task execution into multiple stages: task offloading stage and task uploading stage. Then we conduct in-depth analysis on the response latency and operation cost at different stages, and design a staged scheduling algorithm to jointly optimise the total latency and the cost. In particular, this paper makes the following contributions:

- We build a new MEC system framework specialised for crowd sensing applications that captures the unique workflow of crowd sensing tasks in MEC environments. Specifically, the framework allows us to investigate task scheduling from application datacenter to edge devices and then to edge servers and finally back to datacenter. From the perspective of application provider, we define a practical task scheduling problem with its objective to minimise the operation cost under a latency constraint.
- It is impossible to generate scheduling profiles for both task offloading and task uploading at the same time since they are highly coupled. In this paper, we propose a flexible solution to stage scheduling into task offloading from application provider to edge devices and task shifting across edge servers.
- In order to solve the problem under the assumption of long period of time, we propose a multi-period optimisation algorithm by leveraging the Lyapunov technique. We verify the effectiveness and efficiency of our proposed algorithm via extensive experiments.

The rest of this paper is organised as follows. In Section 2, we review the recent work that are closely related to our work. In Section 3, we elaborate the system framework and present a formal definition of the problem. In Section 4, we analyse the impacts of scheduling profiles upon latency and cost, and present the details of the algorithm design. In Section 5, we evaluate our algorithm with multiple experiments. Finally, we conclude the paper in Section 6.

2 Related work

Crowd sensing provides an efficient way to implement applications that requires collecting massive information regarding physical world in a distributed manner. Owing to the super computing power of cloud, most crowd sensing systems put large amounts of sensing data and computing tasks into the remote cloud datacenter (Merlino et al., 2016). But the combination of crowd sensing with cloud computing suffers from high latency and may not apply to today's applications that are mostly sensitive to response latency. Edge computing schedules tasks to edge devices and servers, which can be used as an excellent technical supplement for those crowd sensing applications.

Recently, a considerable amount of efforts have been devoted to the task scheduling in MEC environments (Wang et al., 2018a). Dinh et al. (2017) explored the task assignment optimisation with a focus on minimising the latency and energy consumption. It mainly considers the computational resources and optimises the task assignment by employing dynamic CPU frequency adjustment algorithms. Sardellitti et al. (2015) took both computational and network resources into account and formalised the problem as a latency constraint energy consumption optimisation. Chen et al. (2016*a*) exploited the distributed computation offloading among users, and converted it as a multi-user offloading game, where users can choose whether to offload tasks and how much calculation to be offloaded. A decision-making algorithm was designed to help users reach Nash equilibrium. Mao et al. (2016) proposed a novel task offloading method for a green MEC system equipped with energy harvesting (EH) devices. Tan et al. (2017) presented a general model for job dispatching and scheduling among edge servers and proposed a provable approximate algorithm.

Most of those studies assume that tasks are issued by individual users and need to be offloaded to edge servers for execution, which are quite different from crowd sensing scenario where tasks are issued by an application service provider. Wang et al. (2016) studied the mobile task scheduling in a multi-cloudlet environment and attempted to optimise the computation efficiency. Zhang et al. (2018) studied the task offloading problem from a bottom-up perspective and developed a non-cooperative task allocation framework based on game theory. However, users have to make their own decisions without being aware of others, which makes it hard to guarantee the existence of optimal solution. Liu et al. (2018a) designed a cooperative Stackelberg game to guide sensing task allocation as well as pricing mechanism. Different from those work, we investigate the problem from sensing service provider's perspective, and focus on the long-term objective achievement. Wang et al. (2018b) proposed a crowdsourcing model that is used to recommend user-preferred and trustful tasks for end users. The recommendation decisions are made based on the computation of both user similarity and task similarity. In contrast, our work primarily focuses on the task scheduling for latency-sensitive applications, which is more complicated since it involves task offloading from servers to devices, task uploading from devices to servers and task shifting across servers.

3 System framework

In this paper, we assume that the crowd sensing services are provided by a hybrid system built on multiple types of resources. The system framework is illustrated in Figure 1, at the top level, the sensing application servers are deployed within a remote cloud datacenter and they provide all individual users access to the crowd sensing application. At the middle level, a group of edge servers are available for additional computation power and temporary intermediate data storage. These edge servers are deployed at the edge of network with a base station (BS) beside providing the radio access for individual users. In the 4G network, the average communication distance to a BS is up to 0.12 square kilometers (Ge et al., 2016; Zhai et al., 2017). Current cellular networks

Table 1 Symbols and their descriptions

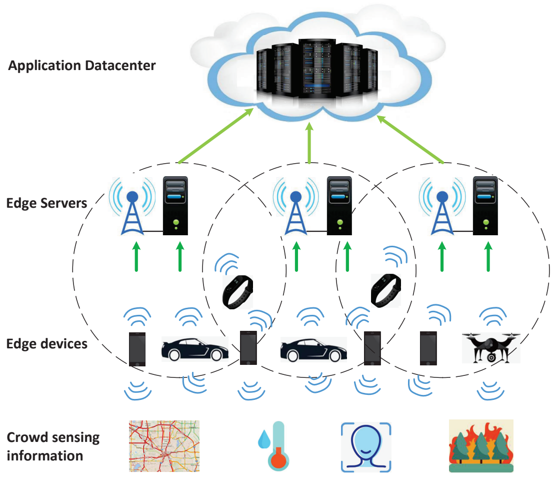
Symbol	*Description*
R^t	Set of tasks r_i at period t, $\vert R^t\vert = m, t \in [0, T-1]$
S^t	Set of edge servers at period t, $\vert S^t\vert = k$
D^t	Set of edge devices at period t, $\vert D^t\vert = n$
$\boldsymbol{\pi}^t$	Task offloading profile at period t
$\boldsymbol{\pi}_i^t$	Task r_i's offloading vector
L_{dev-i}^t	The execution time of r_i spent on the device
L_{up-i}^t	The uploading latency of r_i from device to server
L_{sh-i}^t	The shifting latency of r_i across servers
L_{ser-i}^t	The latency to complete r_i on the server
$Int_{i,x}$	Intermediate data size of r_i at device x
C_{dev-i}^t	Cost paid for r_i's execution on the device
C_{up-i}^t	Cost paid for r_i's intermediate data uploading
C_{sh-i}^t	Cost paid for r_i's intermediate data shifting
C_{ser-i}^t	Cost paid for r_i's execution on the server
$\boldsymbol{\eta}^t$	Task uploading profile
$\boldsymbol{\eta}_i^t$	Task r_i's uploading vector
$\boldsymbol{\sigma}^t$	Task shifting profile
$\boldsymbol{\sigma}_i^t$	Task r_i's shifting vector
R_a^t	Set of tasks uploaded from devices to server a
${R'}_a^t$	Set of tasks arrived on server a
λ_a^t	Arrival rate from devices to server a
${\lambda'}_a^t$	Arrival rate on server a
$Q(t)$	Queue backlog function
$\Psi(Q(t))$	Lyapunov function
$o(Q(t))$	Lyapunov drift function

are dense enough to ensure that individuals who are interested in the crowd sensing application can be covered by multiple BSs. Those users are encouraged to use their own devices to collect sensing data from physical world at the bottom layer and perform some computation tasks for data preprocessing. These edge devices finish tasks and upload intermediate data to any edge server via the an available BS. In return users will be rewarded by the crowd sensing service provider for their contributions. The key symbols used in this paper are summarised in Table 1.

3.1 Crowd sensing task model

As required by many real-time crowd sensing applications, sensing tasks should be periodically initialised and performed to provide uninterrupted services. Those tasks are usually grouped into batches over time and each batch has the same deadline. Here the system is assumed to run for a long period of time say T periods ($T \to \infty$), and the length of each period depends on the specific sensing application which may vary from seconds to hours. At the beginning of each period t, a batch R^t of tasks would be initialised by the application system, and these tasks has to be completed within this period. Otherwise the results returned by them will be invalid. Meanwhile, a set D^t of edge devices owned by individuals compete for those tasks by sending requests to the application center. The request contains each device's configure profiles and current workloads.

Figure 1 The overview of a MEC-enabled system for crowd sensing applications (see online version for colours)



The sensing task's primary function is to rely on edge devices to collect raw data from the physical world and to conduct lightweight computation so that the raw data can be converted into qualified intermediate data. Another function of a task is to rely on edge servers to complete middleweight computation against the intermediate data uploaded from edge devices. We define a quadruple $(r_{i-time}, r_{i-dev}, r_{i-ser}, r_{i-prio})$ to describe a sensing task r_i, where r_{i-time} represents the required sensing time, r_{i-dev} and r_{i-ser} represent the computation demands per data unit on device and server respectively, and r_{i-prio} represents task r_i's priority which reflects the importance of a task and is set to 1 by default for regular tasks. Besides regular tasks, we also consider the existence of a tiny fraction of kernel tasks and grant higher priority to them, i.e., $r_{i-prio} > 1$. Let an n-dimensional vector $\boldsymbol{\pi}_i^t$ be the offloading decision on task r_i at period t, where variable n is the number of edge devices and entry $\pi_{i,x}^t = 1$ indicates task r_i being offloaded to device x, otherwise $\pi_{i,x}^t = 0$.

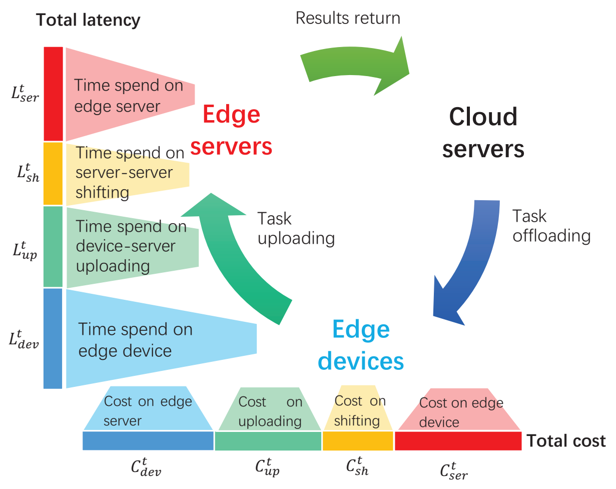
In our framework, the task execution workflow consists of three stages as shown in Figure 2, i.e., task offloading, task uploading and results return. The first stage is in charge of offloading tasks from datacenter to edge devices. At the second stage, edge devices continue to upload tasks accompanied by intermediate data to a set S^t of edge servers for further processing. If an edge device is covered by communication regions of multiple edge servers, it may choose one of them randomly for task uploading. Let k-dimensional vector $\boldsymbol{\eta}_i^t$ be the uploading decision at period t, where variable k is the number of edge servers and entry $\eta_{i,a}^t = 1$ indicates that task r_i and its corresponding intermediate data are uploaded to edge server a, otherwise $\eta_{i,a}^t = 0$. It is worth noting that tasks are allowed to be shifted to other servers for execution due to the following considerations: first, edge servers are often interconnected via the high-speed network, which facilitates task shifting; second, the workloads of edge servers are probably imbalance due to the uneven location distribution of edge devices, which requires rescheduling optimisation; last but not the least, the task uploading profile is hard to be optimised due to the limited choices of servers for each device, while task shifting optimisation can instead be explored by the service provider. Therefore, we add a substage of task shifting to optimise the task rescheduling across edge servers. Similarly let $\boldsymbol{\sigma}_i^t$ be the scheduling vector of task r_i on servers and entry $\sigma_{i,a}^t$ denote whether r_i is finally scheduled on server a. At the last stage, the output of tasks performed on edge servers are returned back to application servers located in the cloud datacenter. Through the first two stages' processing, the final output to be transferred are usually very small compared to raw data, and then the amount of data transferred over internet backbone are reduced significantly and no longer need to be optimised. Therefore, we focus on the optimisation of latency L_i^t and cost C_i^t involved in the first two stages, i.e.,

$$\begin{aligned} L_i^t &= L_{dev-i}^t + L_{up-i}^t + L_{sh-i}^t + L_{ser-i}^t, \\ C_i^t &= \left(C_{dev-i}^t + C_{up-i}^t + C_{sh-i}^t + C_{ser-i}^t\right) \cdot P_i^t, \end{aligned} \tag{1}$$

where L_{dev-i}^t, L_{up-i}^t, L_{sh-i}^t and L_{ser-i}^t are latencies spent on device, uploading, shifting and server respectively, and C_{dev-i}^t, C_{up-i}^t, C_{sh-i}^t and C_{ser-i}^t are costs paid for device, uploading, shifting and server respectively. P_i^t is used to denote whether task r_i is completed successfully. Clearly, we have $P_i^t = 0$ if $L_i^t > L^*$; otherwise, $P_i^t = 1$.

In order to illustrate the execution workflow of sensing task model, we take road traffic management as an example. In this application scenario, drivers in moving cars are able to use dash cam to monitor road conditions. Vehicle built-in smart device can preprocess the recorded videos by extracting visual and motion features from the raw data. The extracted features will be sent to nearby edge servers to do object detection. Finally, the output detections are returned back to cloud servers to perform traffic statistics and forecasting.

Figure 2 An illustration of task execution workflow in the MEC-enabled crowd sensing service system (see online version for colours)



3.2 *Problem formulation*

Based on the above models, our work focuses on the sensing task scheduling problem from the perspective of a crowd sensing service provider. The objective is to minimise the operation cost as well as guarantee response latency over a long period of time. The problem takes sensing tasks and two types of edge resources (i.e., edge devices and edge servers) as input, where tasks will be issued periodically. Hence, time is divided into multiple periods in terms of task updating cycle. To achieve the objective, task scheduling has to jointly optimise both task offloading and task shifting in each period. We formally define the task scheduling problem as follows:

Task scheduling: In a MEC-enabled crowd sensing system, there are a group S^t of edge servers and a group D^t of edge devices available at each period, a set R^t of tasks will be issued and need to be firstly offloaded to edge devices, after uploading to edge servers, they also should be shifted across servers. Let $\boldsymbol{\pi}^t = (\boldsymbol{\pi}_1^t, ..., \boldsymbol{\pi}_n^t)$ and $\boldsymbol{\sigma}^t = (\boldsymbol{\sigma}_1^t, ..., \boldsymbol{\sigma}_k^t)$ denote the task offloading profile and task shifting profile respectively. The objective of the crowd sensing service provider is to find the optimal profiles $\boldsymbol{\pi}^t$ and $\boldsymbol{\sigma}^t$ that minimise the total cost C^t of all tasks as well as guarantee the latency L_i^t of each task r_i less than L^*, i.e.,

$$\begin{aligned} &\arg\min_{\boldsymbol{\pi}^t, \boldsymbol{\sigma}^t} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} C^t \\ &where\, C^t = \sum_{r_i \in R^t} C_i^t, \\ &s.t. \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} L_i^t \le L^*, r_i \in R^t. \end{aligned} \tag{2}$$

4 Staged scheduling

According to the task execution workflow, the scheduling profiles will be used at different stages, e.g., task offloading profile works at the task offloading stage, while task shifting profile determines task rescheduling across edge servers at the task shifting substage. The impacts of two scheduling profiles on latency and cost are highly coupled. The optimisation of shifting profile largely relies on the results of task offloading. To this end, we propose a staged scheduling scheme to optimise both scheduling profiles in a divide-and-conquer fashion.

4.1 Task offloading

When task r_i is offloaded to device x, it takes r_{i-time} to collect raw data, and the data size is denoted by $Raw_{i,x} = f_x(r_{i-time})$, which is a function of r_{i-time} and may vary by device. Let x_{comp} denote device x's computation capacity and x_{load} denote its workload at the beginning of current period ($0 \leq x_{load} < 1$), then the execution time of r_i spent on device x is given by

$$L^t_{i,x} = r_{i-time} + \frac{r_{i-dev} Raw_{i,x}}{x_{comp}(1 - x_{load})}. \tag{3}$$

The execution time of r_i spent on all devices form an n-dimensional vector $(L^t_{i,1}, ..., L^t_{i,n})$. For a given $\boldsymbol{\pi}^t_i$, the expected execution time of r_i spent on the device should be represented by

$$L^t_{dev-i} = (L^t_{i,1}, ..., L^t_{i,n}) \cdot \boldsymbol{\pi}^{t\top}_i. \tag{4}$$

Every individual with an edge device involved in the crowd sensing task execution will be rewarded for their contributions. Note that the system may suffers from data security, such as data tampering or returning faked data maliciously. There are many efforts (Fang et al., 2018; Cao et al., 2019) have been devoted to enhance data security in IoT and edge computing environments. However, since that is beyond the scope of this paper, here we assume that a security mechanism has been integrated for data verification. The cost $C_{i,x}$ paid for r_i's execution on device x, is defined as a function of task r_i and device x, i.e.,

$$C^t_{i,x} = \alpha_{dev} r_{i-time} + \log\left(r^{\beta_{dev}}_{i-dev} \cdot x^{\gamma_{dev}(1-x_{load})}_{comp}\right), \tag{5}$$

where α_{dev}, β_{dev} and γ_{dev} are device cost parameters. $C_{i,x}$ has positive relationships with r_{i-time}, r_{i-dev} and x_{comp} and a negative relationship with x_{load}. As a result, the expected cost paid for r_i's execution on the device is represented by

$$C^t_{dev-i} = (C^t_{i,1}, ..., C^t_{i,n}) \cdot \boldsymbol{\pi}^{t\top}_i. \tag{6}$$

Considering that the uploading profile $\boldsymbol{\eta}^t$ and shifting profile $\boldsymbol{\sigma}^t$ have not been determined yet at this stage, we may not be able to minimise the total cost and latency defined in equation (1) via optimising the offloading profile. Instead we turn the objective into the minimisation of cost and latency spent on devices, i.e.,

$$\begin{aligned} &\arg\min_{\boldsymbol{\pi}^t} \sum_{r_i \in R^t} C^t_{dev-i} \\ &s.t. \frac{1}{m} \sum_{r_i \in R^t} L^t_{dev-i} \leq L^*_{dev}, \end{aligned} \tag{7}$$

where L^*_{dev} is a flexible latency constraint and always such that $L^*_{dev} < L^*$.

The offloading profile $\boldsymbol{\pi}^t$ is a matrix that collects all task offloading vectors $\boldsymbol{\pi}^t_i$. For a specific task r_i, considering that its offloading decision $\pi^t_{i,x}$ regarding each device x is a 0-1 binary variable, the problem is a nonlinear 0-1 programming which can be solved by leveraging a branch-and-bound algorithm (Hansen, 1979).

4.2 Task shifting

Assume that task r_i is offloaded to device x, i.e., $\pi^t_{i,x} = 1$. When x completes the data preprocessing, we obtain the intermediate data of task r_i and its data size is represented by $Int_{i,x} = f_i(Raw_{i,x})$, which is an increasing function of the raw data size. Each device's communication scope may be covered by a multiple edge servers, then the device randomly select one of them for task uploading. As a result, the uploading profile $\boldsymbol{\eta}^t$ is generated in this way. The time spent on uploading task r_i's intermediate data from x to a is calculated by

$$L^t_{i,x,a} = \frac{Int_{i,x}}{w_{x,a}}, \tag{8}$$

where $w_{x,a}$ is the upload bandwidth between x and a. If server a is outside of the communication scope of device r_i, then $w_{x,a} = 0$ and the uploading delay is set to be a maximal value $L_{max} \gg L^*$. The expected uploading latency of r_i is given by

$$L^t_{up-i} = (L^t_{i,x,1}, ..., L^t_{i,x,k}) \cdot \boldsymbol{\eta}^{t\top}_i. \tag{9}$$

The cost paid for data uploading positively depends on the size of intermediate data and upload bandwidth, and its definition is given by

$$C^t_{i,x,a} = \alpha_{up} w_{x,a} \log Int_{i,x}, \tag{10}$$

where α_{up} is the uploading cost parameter. If device x is not covered by server a, tasks on x cannot be uploaded to a, then we have $C^t_{i,x,a} = 0$. Similarly the uploading cost of task r_i is represented by

$$C^t_{up-i} = (C^t_{i,x,1}, ..., C^t_{i,x,k}) \cdot \boldsymbol{\eta}^{t\top}_i. \tag{11}$$

After tasks and their intermediate data are uploaded to the edge servers, they can be rescheduled across servers for further optimisation via task shifting. Assume that task r_i was originally offloaded to server a, when it is shifted to server b, the transfer delay will be

$$L^t_{sh-i,b} = \begin{cases} \frac{Int_{i,x}}{w_{a,b}}, b \neq a, \\ 0, b = a. \end{cases} \tag{12}$$

For a given $\boldsymbol{\sigma}^t_i$, the expected shifting latency for task r_i is calculated by

$$L^t_{sh-i} = (L^t_{sh-i,1}, ..., L^t_{sh-i,k}) \cdot \boldsymbol{\sigma}^{t\top}_i. \tag{13}$$

The corresponding cost for task shifting from a to b is defined as an increasing function of intermediate data size and bandwidth, i.e.,

$$C^t_{i,x,a,b} = \alpha_{sh} w_{ab} \log Int_{i,x}, \tag{14}$$

where α_{sh} is the shifting cost parameter. If $a = b$, then there is no shifting cost and we set $w_{ab} = 0$ so that $C^t_{i,x,a,b} = 0$. The expected shifting cost for task r_i is given by

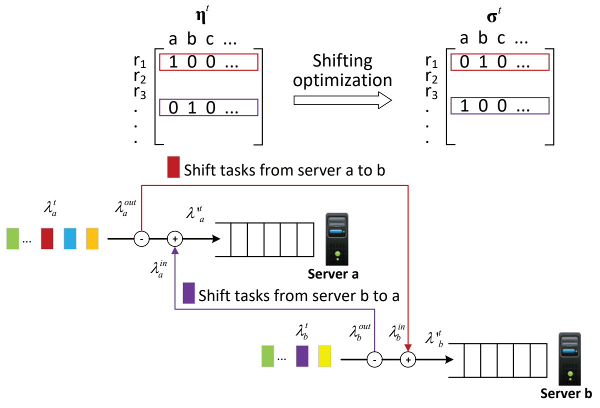
$$C^t_{sh-i} = (C^t_{i,x,a,1}, ..., C^t_{i,x,a,k}) \cdot \boldsymbol{\sigma}^{t\top}_i. \tag{15}$$

For an arbitrary edge server a, it may receives tasks from both edge devices and other edge servers in the same period. In order to analyse the impact of scheduling profile upon latency and cost, we model each edge server as a $M/G/1$ queue by leveraging queuing theory, where M refers to the Poisson distribution of task arrival, and G represents that the computation time follows an arbitrary distribution. Given the uploading profile $\boldsymbol{\eta}^t$, we can obtain the set of tasks on each server, $R^t_a, a \in [1, k]$, and deduce its corresponding arrival rate, λ^t_a. In addition, as shown in Figure 3, by comparing $\boldsymbol{\eta}^t$ with shifting profile $\boldsymbol{\sigma}^t$, we can also deduce the hybrid set of tasks and the corresponding arrival rate, i.e.,

$$R'^t_a = R^t_a / R^{out}_a \cup R^{in}_a, \lambda'^t_a = \lambda^t_a - \lambda^{out}_a + \lambda^{in}_a, \tag{16}$$

where R^{out}_a and R^{in}_a are the sets of tasks shifting out and shifting in, respectively, and λ^{out}_a and λ^{in}_a are the corresponding leaving rate and arrival rate, respectively.

Figure 3 An example of task shifting between two edge servers (see online version for colours)



Let a_{comp} denote the computation capacity of edge server a. The computation time of task r_i on server a should be $L^t_{comp-i,a} = \frac{r_{i-ser} Int_{i,x}}{a_{comp}}$. We use θ to denote the variable of computation time for any task and use $\mathrm{E}[\theta]$ and $\mathrm{E}\left[\theta^2\right]$ to represent its first and second moments, respectively. In terms of Pollaczek-Khinchin mean formula (Chan et al., 1997), the expected waiting time of task r_i on server a is calculated by

$$L^t_{wait-a} = \frac{\lambda'^t_a E[\theta^2]}{2(1 - \lambda'^t_a E[\theta])}. \tag{17}$$

Combining the computation time, we obtain the sojourn time for server a to complete task r_i, i.e., $L^t_{i,x,a} = L^t_{wait-a} + L^t_{comp-i,a}$. Then the expected latency to complete task r_i on the server is represented by

$$L^t_{ser-i} = (L^t_{i,x,1}, ..., L^t_{i,x,k}) \cdot \boldsymbol{\sigma}^{t\top}_i. \tag{18}$$

The cost paid for task r_i's execution on server a is defined as an increasing function of intermediate data size, task computation demand and a's computation capacity, i.e.,

$$C^t_{ser-i,a} = \log\left(Int^{\alpha_{ser}}_{i,x} \cdot r^{\beta_{ser}}_{i-ser} \cdot a^{\gamma_{ser}}_{comp}\right), \tag{19}$$

where α_{ser}, β_{ser} and γ_{ser} are server cost parameters. Then the expected cost of task r_i on a server is given by

$$C^t_{ser-i} = (C^t_{ser-i,1}, ..., C^t_{ser-i,k}) \cdot \boldsymbol{\sigma}^{t\top}_i. \tag{20}$$

4.3 *Kernel tasks*

Kernel tasks are critical for the successful delivery of application services and they should be guaranteed to be completed with high priority. To this end, a kernel task is required to be assigned to a group of d_i devices, where d_i is a random integer chosen from $(1, \sqrt{r_{i-prio}}]$. When $d_i > 1$, there are a group of task copies $(r^{(1)}_i, ..., r^{(d_i)}_i)$, and each of them can be regarded as an independent task. As long as one of them is completed on time, the kernel task is deemed to be successfully executed. Its latency is accordingly set to be the minimum one, i.e., $L^t_i = \min(L^t_{i(1)}, ..., L^t_{i(d_i)})$. At the same time the cost is the sum of all successful executions of copies, i.e., $C^t_i = \sum_{j=1}^{d_i} C^t_{i(j)} \cdot P^t_{i(j)}$, where $P^t_{i(j)}$ is the task completion indicator and $P^t_{i(j)} = 0$ if $L^t_{i(j)} > L^*$; otherwise $P^t_{i(j)} = 1$.

4.4 *Lyapunov optimisation*

At the first stage, we optimise the latency spent on the device by solving a nonlinear subproblem described in equation (7), and obtain the task offloading profile. At the second stage, according to the set of devices covered by each edge server, the task uploading profile is also obtained by means of a random selection of servers within the communication scope. Having offloading profile and uploading profile as input, we can solve the scheduling problem defined in equation (2) that is also a nonlinear 0-1 programming.

However, as the number of periods grows, i.e., $T \to \infty$, it is of challenging to solve the proposed scheduling problem due to the coupling between task scheduling profiles and the long-term latency constraint in each period. To this end, we propose to solve the problem by leverage Lyapunov optimisation technique that is a principled method for analysing dynamic systems.

To satisfy the latency constraint, we create a virtual queue that is used to describe the latency deficit and provide the guidance for the following task scheduling, and then define a function $Q_i(t)$ for each task r_i to denote the queue backlog at period t. The queue's dynamics evolves in a recursive manner, i.e.,

$$Q_i(t+1) = \left[Q_i(t) + L^t_i - L^*\right]^+, \tag{21}$$

where $[value]^+ = \max(value, 0)$ and $Q_i(0) = 0$. We also define a Lyapunov function as follows:

$$\Psi(Q_i(t)) \triangleq \frac{1}{2} \sum_{r_i \in R^t} Q^2_i(t). \tag{22}$$

Here $\Psi(Q_i(t))$ is used to reflect the queue's stability, and a lower value implies a more stable queue. For $\forall t \leq T$, we define a function of $Q_i(t)$ to capture the drift between two adjacent periods,

$$o(Q_i(t)) \triangleq \mathrm{E}\left[\Psi(Q_i(t+1)) - \Psi(Q_i(t)) | Q_i(t)\right]. \tag{23}$$

Through applying Lyapunov optimisation, the objective of long-term task scheduling becomes equivalent to minimising the supremum bound on the following inequality during each period, i.e.,

$$\begin{aligned} &o(Q_i(t)) + \omega \mathrm{E}\left[C_i^t | Q_i(t)\right] \\ &\leq \Lambda + Q_i(t) \mathrm{E}\left[L_i^t - L^* | Q_i(t)\right] + \omega \mathrm{E}\left[C_i^t | Q_i(t)\right], \end{aligned} \tag{24}$$

where $\Lambda = \frac{1}{2}\left(\sum_{r_i \in R^t} \left(L_i^t - L^*\right)\right)^2$ is a weight designed to control the tradeoff of optimisation between cost and latency. Therefore, the scheduling problem is redefined as follows:

Lyapunov optimisation: Given the same input of the original problem, and the profiles $\boldsymbol{\pi^t}$ and $\boldsymbol{\eta^t}$ obtained from the previous stages of each period t, the objective is to find $\boldsymbol{\sigma}^t$ such that it minimises cost and latency, i.e.,

$$\arg\min_{\boldsymbol{\sigma}^t} \sum_{r_i \in R^t} \left(\omega C_i^t + Q_i(t) L_i^t\right), t \in [0, T-1]. \tag{25}$$

The above definition converts the long-term shifting problem to multiple minimisation problems at each period, and they can be solved separately by having the output of offloading and uploading at each period. For each subproblem, the additional term $Q_i(t)L_i^t$ is set to control the latency deficit. Whenever $Q_i(t)$ increases, the deficit must be minimised with a higher priority.

We propose an algorithm named crowd sensing task scheduling (CSTS) to implement the staged scheduling and the pseudocode is described by Algorithm 1. It runs on the cloud and collects the device status sent from individuals in the current period as the input of problem formalised by equation (7). It solves the offloading profile $\boldsymbol{\pi}^t$ based on the branch-and-bound method, so as to ensure that the average latency is less than a given L^*_{dev}. At the second stage, the algorithm first solves the uploading profile $\boldsymbol{\eta}^t$ by randomly selecting a communicable edge server for every device. After that, it solves the nonlinear 0-1 programming defined in equation (25) by means of the branch-and-bound method, and updates the queue backlog for the next period.

5 Experimental evaluation

In this section, we carry out experiments to evaluate the performance of our proposed algorithm. We first introduce the experiment settings including system simulation and baselines and then present the evaluation results.

5.1 Settings

To build the experimental environment, we simulate a MEC-enabled crowd sensing system consisting of $n = 20$ edge servers with computation capacity between 10 and 20, $k = 200$ edge devices with computation capacity between 1 and 5. During each period, 100 tasks are initialised with specified requirements, i.e., sensing time $r_{i-time} \in [1, 1.5] \times 10^4$ ms, computation demands per unit $r_{i-dev} \in [0.5, 1]$, $r_{i-ser} \in [1, 5]$, and priority $r_{prio} = 9$ for kernel tasks. Each device has a set of edge servers available for communication, the set is built by randomly selecting servers from set S^t, and the set size is set to be an integer between 1 and 3. The raw data size is proportional to sensing time, and the factor is set to 2, while the intermediate data size is also proportional to the corresponding raw data size and the factor is set to 0.4. Other parameter settings are listed in Table 2.

Algorithm 1 CSTS algorithm

```
Require: $R^t$, $D^t$, $S^t$, $L^*$, $L^*_{ser}$;
Ensure: $\boldsymbol{\pi}^t$, $\boldsymbol{\eta}^t$, $\boldsymbol{\sigma}^t$;
1: for $r_i \in R^t$ do
2:     $Q_i(0) \leftarrow 0$;
3: end for
4: for $t \leftarrow 0$ to $T-1$ do
5:     receive the current status of edge devices in $D^t$;
6:     solve a nonlinear 0-1 programming formalized by Eq. 7, and obtain $\boldsymbol{\pi}^t$; /* task offloading */
7:     $\boldsymbol{\eta}^t \leftarrow zero\,matrix$;
8:     for $x \in D^t$ do
9:         randomly select an edge server $a$ within the communication region as the uploading target;
10:        $\eta^t_{i,x,a} \leftarrow 1, \forall \pi^t_{i,x} = 1$;
11:        update $\boldsymbol{\eta}^t$ by appending $\pi^t_{i,x}$; /* task uploading */
12:    end for
13:    solve a nonlinear 0-1 programming formalized by Eq. 25, and obtain $\boldsymbol{\sigma}^t$; /* task shifting */
14:    for $r_i \in R^t$ do
15:        calculate $Q_i(t+1)$ based on Eq. 21;
16:    end for
17: end for
18: return the set of ($\boldsymbol{\pi}^t$, $\boldsymbol{\eta}^t$, $\boldsymbol{\sigma}^t$);
```

Table 2 Simulation parameters and their default values

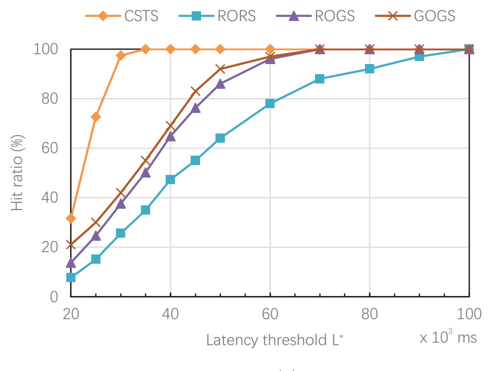
Parameter	*Value*
n	20
k	200
m	100
T	100
L^*	3.5×10^4 ms
L^*_{dev}	2.2×10^4 ms
r_{i-time}	[1, 1.5] $\times 10^4$ ms
r_{i-dev}	[0.5, 1]
r_{i-ser}	[1, 5]
r_{i-prio} for kernel tasks	9
$Raw_{i,x}$	$2 \cdot r_{i-time}$ MB
$Int_{i,x}$	$0.4 \cdot Raw_{i,x}$ MB
x_{comp}	[1, 5]
a_{comp}	[10, 20]
$(\alpha_{dev}, \beta_{dev}, \gamma_{dev})$	(10^{-4}, 0.5, 3)
α_{up}	1
α_{sh}	0.6
$(\alpha_{ser}, \beta_{ser}, \gamma_{ser})$	(1, 1, 2)

We compare our algorithm with the following baslines:

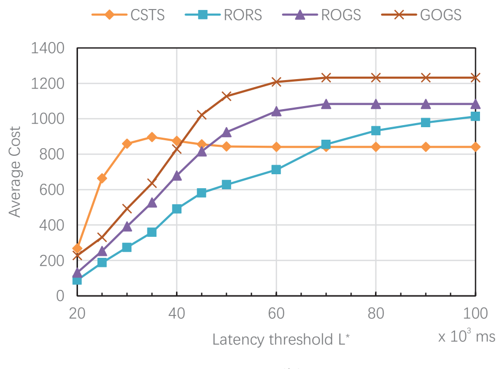
- *Random offloading random shifting (RORS)*: It randomly assigns tasks to edge devices for task offloading. For task shifting, it randomly decides whether and where each task will be shifted.

- *Random offloading greedy shifting (ROGS)*: It generates the task offloading profile in a random manner similar to RORS, but generates the task shifting profile in a greedy manner. Specifically, for task shifting, it first classifies edge servers into two groups by estimating the waiting time, i.e., light-load servers and heavy-load servers. Then for each task r_i, if r_i resides on a heavy-load server, it will be shifted to the server that currently has the lowest workload.
- *Greedy offloading greedy shifting (GOGS)*: For task offloading, it always assigns tasks with highest computation demands to the devices with the best status, e.g, the most powerful computation capacity and the lowest workload. For task shifting, it also conducts greedy shifting similar to ROGS.

Figure 4 Performance comparison under different latency thresholds: (a) hit ratio under different latency thresholds and (b) average cost under different latency thresholds (see online version for colours)



(a)



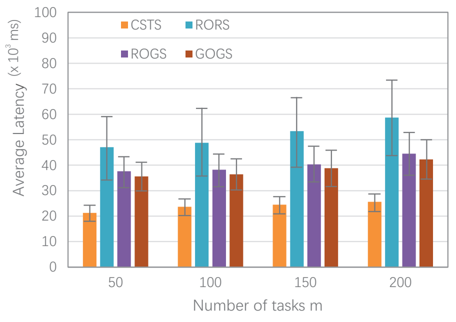
(b)

5.2 Results

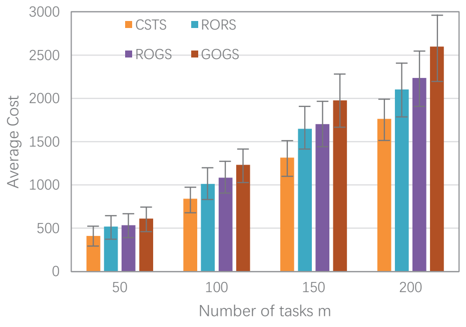
Evaluation under varied latency thresholds: In this group of experiments, we focus on the objective achievement under latency thresholds varied from 2×10^4 ms to 10^5 ms. Specifically, we evaluate the average hit ratio and average cost of all the compared algorithms during 100 periods, where hit ratio is proposed to denote the fraction of completed tasks against total tasks per period. As shown in Figure 4(a), CSTS's hit ratio is always no less than the others. As a baseline, RORS randomly generates offloading and shifting proflies, which results in the longest time to complete all tasks, e.g., 10^5 ms. Greedy offloading reduces the time spent on devices by scheduling high-demand tasks to strong-capacity devices, and greedy shifting reduces the time spent on servers by scheduling tasks from heavy-load servers to light-load ones. Thus both ROGS and GOGS achieve a better performance than RORS. In contrast, CSTS reduces latencies at all stages except uploading by optimising task scheduling. Therefore, it is able to ensure all tasks being completed successfully within an even lower constraint, e.g., 3.5×10^4 ms.

Figure 4(b) shows that almost all algorithms except CSTS have an sustained increasing trend on cost along with the growth of threshold. We notice that CSTS's cost is higher than the others when $L^* \leq 4 \times 10^4$ ms. This is because the hit ratios of others are far below 100% when $L^* \leq 4 \times 10^4$ and the costs are only paid for the small fraction of tasks that have been completed successfully. We also notice that the cost of CSTS decreases as L^* increases from 3.5×10^4 to 6×10^4. It is because high threshold constraint allows more room for optimisation, at the same time the merits will disappear as L^* increases. Therefore, the cost of CSTS does not continue to decline.

Figure 5 Performance comparison with different number of tasks: (a) average latency with different number of tasks and (b) average cost with different number of tasks (see online version for colours)



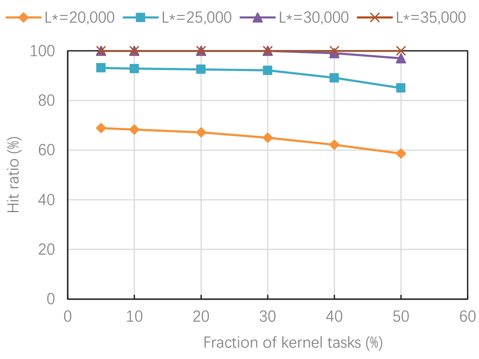
(a)



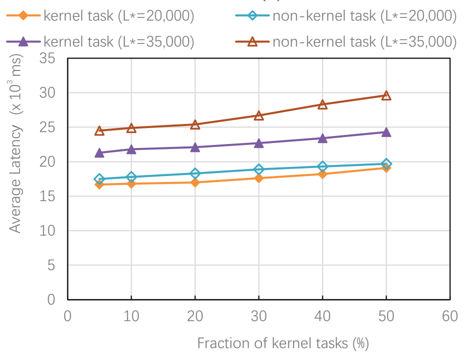
(b)

Evaluation under different number of tasks: In this group of experiments, we evaluate the impact of tasks upon the latency and cost. The latency threshold is set to 10^5 ms to ensure that all tasks could be accomplished before the deadline no matter which algorithm is adopted. Figure 5(a) shows the average latencies and the corresponding bounds of all algorithms as the number of tasks varies. Compared to baselines, CSTS optimises the long-term task scheduling across devices and servers so that it achieves the lowest latency and tightest bounds under different numbers of tasks. Figure 5(b) shows the average cost per period and the corresponding bounds, where the average cost is found to be approximately linear with the number of tasks, meanwhile the bounds become larger as the number of tasks increases.

Figure 6 Self-performance comparison under different fraction of kernel tasks: (a) hit ratio under different fractions of kernel tasks and (b) average latency under different fractions of kernel tasks



(a)



(b)

Evaluation under different kernel fractions: In this group of experiments, we explore the impact of kernel tasks upon the performance by varying the fraction of kernel tasks from 5% to 50%. From the results shown in Figure 6(a), we notice that when the latency threshold is set too low, e.g., $L^* < 3.5 \times 10^4$ ms, the hit ratio of kernel tasks will decline as the fraction increases. For each kernel task, there will be multiple independent task copies issued and the total number of tasks increases. Then the average latency of kernel tasks increases accordingly as shown in Figure 6(b). The latency of a kernel task takes the value of the minimum of its task copies, so that kernel tasks have the lower average latency compared with non-kernel tasks. When the fraction of kernel tasks is high (e.g., exceeds 30%) but the latency threshold is low (e.g., less than 3.5×10^4 ms in our system settings), there may be a tiny number of kernel tasks and their latencies exceeds the threshold. As a result, in order to avoid the impact of timeout of kernel tasks upon the sensing service quality, it is advised to lower the fraction of kernel tasks in each batch of tasks, otherwise, we have to raise the latency constraint to a proper threshold.

6 Conclusion

In this paper, we investigated the MEC-enabled crowd sensing service system with a focus on task scheduling optimisation from the service provider's perspective. We built a workflow framework for crowd sensing services and it captures the task execution logic across varied stages. To address the scheduling problem for many latency-sensitive crowd sensing applications, we first defined a formalised scheduling problem based on the proposed framework and then proposed a staged scheduling scheme. The proposed scheme decouples the original problem by converting it into two sub-problems at different stages. Moreover, we applied Lyapunov optimisation technique to address the long-term latency-constraint cost minimisation. We also evaluated the proposed algorithm by extensive experiments.

Acknowledgement

This work is supported by National Natural Science Foundation of China (No. 61972272, No. U1905211, No. 61672370), the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing (No. 2019A04), Jiangsu Planned Projects for Postdoctoral Research Funds (No. 1701173B), Open Project Program of State Key Lab. for Novel Software Technology (No. KFKT2019B16) and Jiangsu Overseas Visiting Scholar Program for University Prominent Yong & Middle-aged Teachers and Presidents.

References

Cao, C., Wang, J., Wang, J., Lu, K., Zhou, J., Jukan, A. and Zhao, W. (2019) 'Optimal task allocation and coding design for secure coded edge computing', *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019*, Dallas, TX, USA, 7–10 July, pp.1083–1093.

Capponi, A., Fiandrino, C., Kantarci, B., Foschini, L., Kliazovich, D. and Bouvry, P. (2019) 'A survey on mobile crowdsensing systems: Challenges, solutions, and opportunities', *IEEE Commun. Surv. Tutorials*, Vol. 21, No. 3, pp.2419–2465.

Chan, W., Lu, T-C. and Chen, R-J. (1997) 'Pollaczek-khinchin formula for the m/g/i queue in discrete time with vacations', *IEE Proceedings – Computers and Digital Techniques*, Vol. 144, No. 4, pp.222–226.

Chen, X., Jiao, L., Li, W. and Fu, X. (2016a) 'Efficient multi-user computation offloading for mobile-edge cloud computing', *IEEE/ACM Transactions on Networking*, Vol. 24, No. 5, pp.2795–2808.

Chen, Y., Chen, J. and Tseng, Y. (2016b) 'Inference of conversation partners by cooperative acoustic sensing in smartphone networks', *IEEE Trans. Mob. Comput.*, Vol. 15, No. 6, pp.1387–1400.

Dai, W., Wang, Y., Jin, Q. and Ma, J. (2016) 'An integrated incentive framework for mobile crowdsourced sensing', *Tsinghua Science and Technology*, Vol. 21, No. 2, pp.146–156.

Devarakonda, S., Sevusu, P., Liu, H., Liu, R., Iftode, L. and Nath, B. (2013) 'Real-time air quality monitoring through mobile sensing in metropolitan areas', *ACM SIGKDD International Workshop on Urban Computing, UrbComp*, Chicago, IL, USA, pp.15:1–15:8.

Dinh, T.Q., Tang, J., La, Q.D. and Quek, T.Q.S. (2017), 'Offloading in mobile edge computing: task allocation and computational frequency scaling', *IEEE Trans. Communications*, Vol. 65, No. 8, pp.3571–3584.

Fang, X., Yang, M. and Wu, W. (2018) 'Security cost aware data communication in low-power iot sensors with energy harvesting', *Sensors*, Vol. 18, No. 12, p.4400.

Ge, X., Tu, S., Mao, G., Wang, C. and Han, T. (2016) '5g ultra-dense cellular networks', *IEEE Wireless Commun.*, Vol. 23, No. 1, pp.72–79.

Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M. (2013), 'Internet of things (IoT): a vision, architectural elements, and future directions', *Future Generation Comp. Syst.*, Vol. 29, No. 7, pp.1645–1660.

Hansen, P. (1979) 'Methods of nonlinear 0-1 programming', *Annals of Discrete Mathematics*, Vol. 5, pp.53–70.

Ilarri, S., Wolfson, O. and Delot, T. (2014) 'Collaborative sensing for urban transportation', *IEEE Data Eng. Bull.*, Vol. 37, No. 4, pp.3–14.

Lin, L., Liao, X., Jin, H. and Li, P. (2019) 'Computation offloading toward edge computing', *Proceedings of the IEEE*, Vol. 107, No. 8, pp.1584–1607.

Liu, C., Du, R., Wang, S. and Bie, R. (2018a) 'Cooperative stackelberg game based optimal allocation and pricing mechanism in crowdsensing', *International Journal of Sensor Networks*, Vol. 28, No. 1, pp.57–68.

Liu, J., Shen, H., Narman, H.S., Chung, W. and Lin, Z. (2018*b*) 'A survey of mobile crowdsensing techniques: a critical component for the internet of things', *ACM Transactions on Cyber-Physical Systems*, Vol. 2, No. 3, pp.18:1–18:26.

Liu, W., Chen, J., Wang, Y., Gao, P., Lei, Z. and Ma, X. (2020) 'Quantum-based feature selection for multiclassification problem in complex systems with edge computing', *Complexity*, Vol. 2020, pp.8216874:1–8216874:12.

Mao, Y., Zhang, J. and Letaief, K.B. (2016) 'Dynamic computation offloading for mobile-edge computing with energy harvesting devices', *IEEE Journal on Selected Areas in Communications*, Vol. 34, No. 12, pp.3590–3605.

Merlino, G., Arkoulis, S., Distefano, S., Papagianni, C.A., Puliafito, A. and Papavassiliou, S. (2016) 'Mobile crowdsensing as a service: A platform for applications on top of sensing clouds', *Future Gener. Comput. Syst.*, Vol. 56, pp.623–639.

Sardellitti, S., Scutari, G. and Barbarossa, S. (2015) 'Joint optimization of radio and computational resources for multicell mobile-edge computing', *IEEE Trans. Signal and Information Processing over Networks*, Vol. 1, No. 2, pp.89–103.

Satyanarayanan, M., Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., Hu, W. and Amos, B. (2015) 'Edge analytics in the internet of things', *IEEE Pervasive Computing*, Vol. 14, No. 2, pp.24–31.

Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016) 'Edge computing: vision and challenges', *IEEE Internet of Things Journal*, Vol. 3, No. 5, pp.637–646.

Tan, H., Han, Z., Li, X. and Lau, F. C. M. (2017) 'Online job dispatching and scheduling in edge-clouds', *IEEE Conference on Computer Communications, INFOCOM*, Atlanta, GA, USA, pp.1–9.

Wang, B., Sun, Y., Liu, D., Nguyen, H.M. and Duong, T.Q. (2020) 'Social-aware uav-assisted mobile crowd sensing in stochastic and dynamic environments for disaster relief networks', *IEEE Trans. Vehicular Technology*, Vol. 69, No. 1, pp.1070–1074.

Wang, J., Wang, L., Wang, Y., Zhang, D. and Kong, L. (2018a) 'Task allocation in mobile crowd sensing: State-of-the-art and future opportunities', *IEEE Internet of Things Journal*, Vol. 5, No. 5, pp.3747–3757.

Wang, L., Jiao, L., Kliazovich, D. and Bouvry, P. (2016), Reconciling task assignment and scheduling in mobile edge clouds', *24th IEEE International Conference on Network Protocols, ICNP*, Singapore, pp.1–6.

Wang, Y., Tong, X., Wang, K., Fan, B., He, Z. and Yin, G. (2018b) 'A novel task recommendation model for mobile crowdsourcing systems', *International Journal of Sensor Networks*, Vol. 28, No. 3, pp.139–148.

Xiao, Z., Xiao, Y. and Chen, H. (2014) 'An accountable framework for sensing-oriented mobile cloud computing', *Journal of Internet Technology*, Vol. 15, No. 5, pp.813–822.

Zhai, S., Tang, Z., Wang, D., Li, Z., Chen, X., Fang, D. and Chen, F. (2017) 'Coverage hole detection and recovery in wireless sensor networks based on rssi-based localization', *2017 IEEE International Conference on Computational Science and Engineering, CSE 2017, and IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2017*, Guangzhou, China, 21–24 July, Vol. 2, pp.250–257.

Zhang, D., Ma, Y., Zhang, Y., Lin, S., Hu, X.S. and Wang, D. (2018) 'A real-time and non-cooperative task allocation framework for social sensing applications in edge computing systems', *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, Porto, Portugal, pp.316–326.