Computation Offloading Strategy Optimization with Multiple Heterogeneous Servers in Mobile Edge Computing

Keqin Li, *Fellow, IEEE*

Abstract—Computation offloading from a user equipment (UE) to a mobile edge cloud (MEC) is an effective way to ease the computational burden of mobile devices, to improve the performance of mobile applications, to reduce the energy consumption, and to extend the battery lifetime of mobile user equipments. In this paper, we consider computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing. Queueing models are established for a UE and multiple heterogeneous servers from different MECs, and the average task response time of the UE and each MEC server and the average response time of all offloadable and non-offloadable tasks generated on the UE are rigorously analyzed. Three multi-variable optimization problems are formulated, i.e., minimization of average response time with average power consumption constraint, minimization of average power consumption with average response time constraint, and minimization of cost-performance ratio, so that computation offloading strategy optimization, power-performance tradeoff, as well as power-time product can all be studied in the context of load balancing. An efficient numerical method (which consists of a series of fast numerical algorithms) is developed to solve the problems of minimization of average response time with average power consumption constraint, minimization of average power consumption with average response time constraint, and minimization of cost-performance ratio. Numerical examples and data are also demonstrated to show the effectiveness of our method and to show the power-performance tradeoff, the power-time product, and the impact of various parameters. To the best of the author's knowledge, this is the first work in the literature that analytically addresses computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing.

Index Terms—Average response time, computation offloading strategy, cost-performance ratio, mobile edge cloud, mobile edge computing, power consumption, power-performance tradeoff, queueing model

1 Introduction

1.1 Motivation

A smart mobile device (e.g., smartphone, tablet, handheld computer, wearable device, and personal digital assistant) has been developed into a formidable equipment to provide much of the functionality of a laptop or a desktop computer. Mobile users expect to run pervasive and powerful applications, such as speech recognition, natural language processing, image processing, face detection and recognition, interactive gaming, reality augmentation, intelligent video acceleration, connected vehicles, and Internet of Things gateway [11]. However, due to limited computing capability, memory capacity, database storage, and due to finite battery lifetime, it is very challenging for a mobile device to support these novel but computation-intensive and energy-hungry applications.

As a newly emerged computing paradigm, mobile edge computing provides cloud computing capabilities and service environments at the edge of cellular networks and within the radio access networks in close proximity to mobile subscribers [2]. Mobile edge computing can increase performance compared to providing such services through cloud servers or through core network servers. An MEC platform has unique advantages and capabilities such as proximity to the users and network edge, location awareness and highly localized service, high bandwidth, ultra-low latency, and unparalleled quality of experience [21].

Computation offloading from a *user equipment* (UE, also called mobile user, mobile subscriber, or mobile device) to a *mobile edge cloud* (MEC) is an effective way to address the above challenge. Traditionally, *computation offloading* refers to the transfer of certain computing tasks to an external platform, such as a cluster, a grid, or a cloud. Computation offloading may be necessary due to hardware limitations of a computer system handling a particular task on its own. Computation offloading may also be employed to save energy consumption of a computer system. By utilizing MEC services, a mobile user equipment can benefit from an MEC's powerful computing resources, expedite its task execution, and save its battery power. Therefore, an MEC has the potential to ease the computational burden of mobile devices, to improve the performance of mobile applications, to reduce the energy consumption and to extend the battery lifetime of mobile user equipments [10], [13], [15], [23].

• *The author is with the College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561. E-mail: lik@newpaltz.edu.*

Manuscript received 29 Nov. 2017, revised 30 Jan. 2019, accepted 9 Mar. 2019, Date of publication 0 . 0000; date of current version 0 . 0000.
(Corresponding author: Keqin Li.)
Recommended for acceptance by W. Shi and C. Jiang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSUSC.2019.2904680

2377-3782 © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

IEEE Proof

However, computation offloading may not be beneficial due to possible long transmission delay from a mobile device to an MEC server and extra transmission energy for remote task execution. An offloadable task is typically a task which requires extensive computation time but much less data communication time. A non-offloadable task is typically a task which requires so much data communication time that offloading the task does not benefit at all. A computation offloading strategy should not only make offloading decisions (i.e., whether or not, fully or partially), but also determine communication (i.e., radio transmission power and channel bandwidth) and computing (i.e., CPU clock frequency and execution speed) resources, so that the combined processing time and energy consumption for both communication and computation can be optimized. Computation offloading strategy optimization with multiple heterogeneous MEC servers is even more challenging due to the additional issue of load balancing among the MEC servers, which also serve other users and application areas.

1.2 Our Contributions

In this paper, we consider computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing. The main contributions of the paper are summarized as follows. (1) We establish queueing models for a UE and multiple heterogeneous servers from different MECs, and rigorously analyze the average task response time of the UE and each MEC server and the average response time of all offloadable and non-offloadable tasks generated on the UE. (2) We formulate three multi-variable optimization problems, i.e., minimization of average response time with average power consumption constraint, minimization of average power consumption with average response time constraint, and minimization of cost-performance ratio, so that computation offloading strategy optimization, power-performance tradeoff, as well as power-time product can all be studied in the context of load balancing. (3) We develop an efficient numerical method (which consists of a series of fast numerical algorithms) to solve the problems of minimization of average response time with average power consumption constraint, minimization of average power consumption with average response time constraint, and minimization of cost-performance ratio. We also demonstrate numerical examples and data to show the effectiveness of our method and to show the power-performance tradeoff, the power-time product, and the impact of various parameters. To the best of the author's knowledge, this is the first work in the literature that analytically addresses computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing.

Compared with existing research, our investigation in this paper has the following new and unique features.

First, we consider multiple MECs and multiple heterogeneous MEC servers. Most existing studies assume that there is one MEC which has only one server. In [20], an MEC server is equipped with a multicore high-speed CPU, so that it can execute several applications in parallel; however, the processing latency at the MEC server is assumed to be negligible. In [6], a cloudlet is modeled as a set of homogeneous servers. In our study, there are multiple MECs or an MEC is equipped with multiple heterogeneous servers, and the performance of both the UE and the heterogeneous MEC servers are critical and carefully evaluated.

Second, each MEC server has its own preloaded tasks and performance commitment. All existing researches assume that an MEC server only processes offloaded tasks but nothing else. In this paper, in addition to offloaded tasks from the UE, each MEC server also has its own preloaded tasks, possibly from other UEs and application areas. Furthermore, each MEC server has its own commitment on performance guarantee, which means that the amount of offloaded tasks from a UE to an MEC server is limited.

Third, queueing models are established for both UE and MEC servers. In [22], an M/G/1 queueing model is established only for a mobile device. Although queueing models are established for both UE and MEC servers in [6], they are M/M/1 queueing systems. In our study, both UE and MEC serves are modeled as M/G/1 queueing systems, so that the average task response time of the UE and each MEC server can be obtained accurately and analytically and the average response time of all offloadable and non-offloadable tasks generated on the UE can be optimized.

Fourth, in addition to offloading decision and power allocation within a UE, we consider load balancing among the heterogeneous MEC servers with preloaded tasks and performance commitment, instead of task/transmission scheduling within a UE. Notice that since all servers in [6] are identical without preloaded tasks and performance commitment, there is no issue of load balancing, i.e., all homogeneous servers simply receive the same amount of offloadable tasks. Although multiple heterogeneous servers from multiple MECs are considered in [25], there is no issue of load balancing, since each mobile user has only one task.

Fifth, we consider power constrained performance optimization and performance constrained power optimization. Most existing studies try to minimize a weighted sum of execution time and energy consumption. The main concern of this method is that time (measured by seconds) and energy (measured by Joules) are very different in nature and it makes little sense to consider a weighted sum. Our approach in this paper is minimization of average response time with average power consumption constraint and minimization of average power consumption with average response time constraint, i.e., optimizing one metric while fixing the other. Furthermore, we also minimize the cost-performance ratio (i.e., the power-time product).

2 Related Research

Computation offloading in mobile edge computing has been a hot research topic in recent years, and extensive investigation has been conducted. The reader is referred to [3], [17], [27] for recent comprehensive surveys. These research can be grouped into several categories based on the number of mobile users and the number of tasks each user has.

Single User with Single Task. There is one user with a single task. Wang et al. investigated partial computation offloading for a single application by jointly optimizing the computational speed and transmit power of a smart mobile device and offloading ratio with two system design objectives, i.e., energy consumption minimization and application execution latency minimization [26].

IEEE Proof

Single User with Multiple Tasks. There is one user with multiple tasks. Mao et al. investigated a green MEC system with a single energy harvesting device and developed an effective computation offloading strategy, where the execution cost includes both execution latency and task failure, by proposing a dynamic computation offloading algorithm, which jointly decides the offloading decision, the CPU frequencies for mobile execution, and the transmit power for computation offloading [18]. Mao et al. jointly optimized task offloading scheduling and transmit power allocation for an MEC system with multiple independent tasks from a single-user [19]. Shah-Mansouri et al. formulated a utility maximization problem for a single mobile device, which takes energy consumption, delay, and price of cloud service into account, where a mobile device is characterized by two M/G/1 queuing systems, one for the local CPU and another for the wireless interface [22].

Multiple Users with Single Task. There are multiple users, each has a single task. Cao and Cai investigated the problem of multi-user computation offloading for cloudlet based mobile cloud computing in a multi-channel wireless contention environment, by formulating the multi-user computation offloading decision making problem as a non-cooperative game, where each mobile device user has one computation task with the same number of CPU cycles and attempts to minimize a weighted sum of execution time and energy consumption [5]. Chen formulated a decentralized computation offloading decision making problem among mobile device users as a decentralized computation offloading game, where each mobile device user has a computationally intensive and delay sensitive task and minimizes a weighted sum of computational time and energy consumption [8]. Chen et al. studied the multi-user computation offloading problem for mobile-edge cloud computing in a multi-channel wireless interference environment, and showed that it is NP-hard to compute a centralized optimal solution, and hence adopted a game theoretic approach to achieving efficient computation offloading in a distributed manner [9]. Ma et al. researched computation offloading strategies of multiple users via multiple wireless access points by taking energy consumption and delay (including computing and transmission delay) into account, and presented a game-theoretic analysis of the computation offloading problem while mimicking the selfish nature of the individuals [16]. Tao et al. investigated the problem of energy optimization for multiple users with performance guarantee [24]. You et al. studied optimal resource allocation for a multi-user mobile-edge computation offloading system, where each user has one task, by minimizing the weighted sum of mobile energy consumption under the constraint on computation latency, under the assumption of negligible cloud computing and result downloading time [28]. Zhang et al. studied energy-efficient computation offloading mechanisms for MEC in 5G heterogeneous networks by formulating an optimization problem to minimize the energy consumption of an offloading system with multiple mobile devices, where each device has a computation task to be completed within certain delay constraint, and the energy cost of both task computing and file transmission are taken into consideration [30].

Multiple Users with Multiple Tasks. There are multiple users, each has multiple tasks. Cardellini et al. considered a usage scenario where multiple non-cooperative mobile users share the limited computing resources of a close-by cloudlet and can selfishly decide to send their computations to any of the three tiers, i.e., a local tier of mobile nodes, a middle tier (cloudlets) of nearby computing nodes, and a remote tier of distant cloud servers [6]. Mao et al. investigated the tradeoff between two critical but conflicting objectives in multi-user MEC systems, namely, the power consumption of mobile devices and the execution delay of computation tasks, by considering a stochastic optimization problem, for which, the CPU frequency, the transmit power, as well as the bandwidth allocation should be determined for each device in each time slot [20].

Multiple MECs. All the above studies are for a single MEC. There has been investigation concerning multiple MECs. Tran and Pompili studied the problem of joint task offloading and resource allocation in a multi-cell and multi-server MEC system in order to maximize users task offloading gains, which are measured by the reduction in task completion time and energy consumption, by considering task offloading decision, uplink transmission power of mobile users, and computing resource allocation in the MEC servers [25].

Our investigation in this paper belongs to the category of a single user with multiple (actually, infinite) tasks in a multiple MECs environment, quite different from all the existing researches. It is clear that the benefits of considering multiple MECs are two-fold. First, multiple MECs enhance the processing power of mobile edge computing. Second, multiple MECs increase the flexibility of a UE in choosing an appropriate MEC. We are not interested in offloading one task or a group of tasks, but a stream of tasks. Our performance and cost metrics are the average response time of all tasks (offloadable and non-offloadable) generated on the UE and the average power consumption of the UE for both computation and communication, as well as the cost-performance ratio, i.e., the product of the above two metrics, which has rarely been considered before.

3 Preliminaries

In this section, we present the preliminaries, including a queueing model and two power consumption models. (For reader's convenience, Section 1 of the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSUSC.2019.2904680, gives a summary of notations and their definitions in the order introduced in the paper.)

3.1 A Queueing Model

To analytically study computation offloading strategy optimization in mobile edge computing, we need to establish mathematical models. Throughout the paper, we use $\overline{x}$ to represent the expectation of a random variable x.

Assume that there is a mobile UE and n MECs, i.e., MEC_1, $\text{MEC}_2, \ldots, \text{MEC}_n$ (see Fig. 1, which contains $n+1$ M/G/1 queueing systems). Let p_i be the probability that MEC_i is preferred for offloading when a new offloadable task is generated, because the UE is in the vicinity of MEC_i, or there is a communication channel or special processing capability available in MEC_i, for all $1 \le i \le n$. Clearly, we have $p_1 + p_2 + \cdots + p_n = 1$. When MEC_i is preferred for offloading, the UE can offload its computations to MEC_i, not other MECs.

IEEE Proof

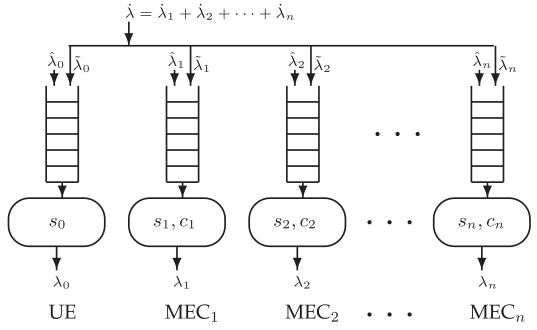

Fig. 1. Computation task offloading from a UE to n MECs.

In this paper, a UE is treated as an M/G/1 queueing system. Thus, the UE is actually a server. There is a Poisson stream of computation tasks with arrival rate $\check{\lambda}$ (measured by the number of arrival tasks per unit of time, e.g., second), i.e., the inter-arrival times are independent and identically distributed (i.i.d.) exponential random variables with mean $1/\check{\lambda}$. Note that a Poisson stream can be divided into substreams and multiple Poisson streams can be combined into a single Poisson stream. The arrival rate $\check{\lambda}$ is decomposed into $\check{\lambda} = \hat{\lambda}_0 + \dot{\lambda}$. That is, there is a Poisson stream of non-offloadable computation tasks with arrival rate $\hat{\lambda}_0$, and there is a Poisson stream of offloadable computation tasks with arrival rate $\dot{\lambda}$. The stream of offloadable computation tasks is further divided into n substreams with arrival rates $\dot{\lambda}_1, \dot{\lambda}_2, \ldots, \dot{\lambda}_n$ respectively, where $\dot{\lambda}_i = p_i\dot{\lambda}$, for all $1 \le i \le n$. Hence, we have $\dot{\lambda} = \dot{\lambda}_1 + \dot{\lambda}_2 + \cdots + \dot{\lambda}_n$. The ith substream is generated when MEC$_i$ is preferred for offloading, where $1 \le i \le n$. The ith substream is further divided into two sub-substreams, i.e., $\dot{\lambda}_i = \tilde{\lambda}_i + \ddot{\lambda}_i$, such that the sub-substream with arrival rate $\tilde{\lambda}_i$ is offloaded to MEC$_i$ and processed remotely in MEC$_i$, while the sub-substream with arrival rate $\ddot{\lambda}_i$ is processed locally in the UE, where $1 \le i \le n$. The vector $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$, where $\tilde{\lambda}_i \le \dot{\lambda}_i$, $1 \le i \le n$, is actually a *computation offloading strategy* of the UE. Note that the offloading strategy does not specify which specific tasks are offloaded to MEC$_i$, but how a substream is divided into two sub-substreams of offloaded and non-offloaded tasks.

Let $\lambda_0 = \hat{\lambda}_0 + \tilde{\lambda}_0$ be the total arrival rate of computation tasks that are processed locally in the UE, where $\tilde{\lambda}_0 = \ddot{\lambda}_1 + \ddot{\lambda}_2 + \cdots + \ddot{\lambda}_n$ is the total arrival rate of offloadable computation tasks that are processed locally in the UE. Let $\tilde{\lambda} = \tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n$ be the total arrival rate of computation tasks that are offloaded to the n MECs. Since $\tilde{\lambda}_0 = \dot{\lambda} - \tilde{\lambda} = \dot{\lambda} - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n)$, we get $\lambda_0 = \check{\lambda} - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n) = \hat{\lambda} + \dot{\lambda} - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n) = \hat{\lambda} + (\dot{\lambda}_1 + \dot{\lambda}_2 + \cdots + \dot{\lambda}_n) - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n)$.

Each MEC is also treated as an M/G/1 queueing system. Thus, an MEC is actually a server. There is a Poisson stream of computation tasks with arrival rate $\hat{\lambda}_i$ to MEC$_i$. This stream of tasks is already there and has nothing to do with the UE. As mentioned above, MEC$_i$ also accepts the ith sub-substream with arrival rate $\tilde{\lambda}_i$ from the UE. Therefore, the total arrival rate of computation tasks that are processed by MEC$_i$ is $\lambda_i = \hat{\lambda}_i + \tilde{\lambda}_i$, where $1 \le i \le n$.

Each M/G/1 queueing system maintains a queue with infinite capacity for waiting tasks when the server is busy in processing other tasks. The first-come-first-served (FCFS) queueing discipline is adopted.

The execution requirements (measured by the number of processor cycles or the number of billion instructions (BI) to be executed) of the non-offloadable computation tasks generated on the UE are i.i.d. random variables r_0 with an arbitrary probability distribution. We assume that its mean $\overline{r_0}$ and second moment $\overline{r_0^2}$ are available. The execution requirements of the offloadable computation tasks generated on the UE are i.i.d. random variables r with an arbitrary probability distribution. We assume that its mean $\overline{r}$ and second moment $\overline{r^2}$ are available. The execution requirements of the tasks already received and processed on MEC$_i$ and not offloaded from the UE are i.i.d. random variables r_i with an arbitrary probability distribution. We assume that its mean $\overline{r_i}$ and second moment $\overline{r_i^2}$ are available, where $1 \le i \le n$.

The amount of data (measured by the number of million bits (MB)) to be communicated between the UE and the MECs for offloadable tasks are i.i.d. random variables d with an arbitrary probability distribution. We assume that its mean $\overline{d}$ and second moment $\overline{d^2}$ are available.

The UE has execution speed s_0 (measured by GHz or the number of billion instructions that can be executed in one second). MEC$_i$ has execution speed s_i, where $1 \le i \le n$. The communication speed (measured by the number of million bits that can be communicated in one second) between the UE and MEC$_i$ is c_i, where $1 \le i \le n$.

We would like to mention that the above queueing models can also be applicable to the situation when there is one MEC with multiple heterogeneous MEC servers $S_1, S_2, \ldots, S_n$, where each S_i corresponds to MEC$_i$, for all $1 \le i \le n$. The condition $\tilde{\lambda}_i \le \dot{\lambda}_i$ can be translated into an equivalent condition $\rho_i \le \rho_i^*$, which states that the utilization of S_i cannot exceed certain bound ρ_i^*, since S_i has performance commitment for its users, where $1 \le i \le n$.

To summarize, the heterogeneous servers are different in vicinity (p_i), execution speed (s_i), communication speed (c_i), amount ($\hat{\lambda}_i$) and characteristics ($\overline{r_i}, \overline{r_i^2}$) of preloaded tasks.

3.2 Power Consumption Models

To analytically study energy consumption in mobile edge computing, we need to establish server speed and power consumption models.

Power dissipation and circuit delay in digital CMOS circuits can be accurately modeled by simple equations, even for complex microprocessor circuits. CMOS circuits have dynamic, static, and short-circuit power dissipation; however, the dominant component in a well designed circuit is dynamic power consumption P_d (i.e., the switching component of power) of the UE, which is approximately $P_d = aCV^2f$, where a is an activity factor, C is the loading capacitance, V is the supply voltage, and f is the clock frequency [7]. In the ideal case, the supply voltage and the clock frequency are related in such a way that $V \propto f^{\phi}$ for some constant $\phi > 0$ [29]. The execution speed s_0 of the server in the UE is usually linearly proportional to the clock frequency, namely, $s_0 \propto f$. For ease of discussion, we will assume that $V = bf^{\phi}$ and $s_0 = cf$, where b and c are some constants. Hence, we know that the dynamic power consumption of the UE is $P_d = aCV^2f = ab^2Cf^{2\phi+1} = (ab^2C/c^{2\phi+1})s_0^{2\phi+1} = \xi s_0^{\alpha}$, where $\xi = ab^2C/c^{2\phi+1}$ and $\alpha = 2\phi + 1$. For instance, by setting $b = 1.16$, $aC = 7.0$, $c = 1.0$, $\phi = 0.5$, $\alpha = 2\phi + 1 = 2.0$, and $\xi = ab^2C/c^{\alpha} = 9.4192$, the value of P_d calculated by the

IEEE Proof

equation $P_d = aCV^2 f = \xi s_0^\alpha$ is reasonably close to that in [12] for the Intel Pentium M processor.

The server in the UE still consumes some amount of power P_s even when it is idle, which includes static power dissipation, short circuit power dissipation, and other leakage and wasted power [1].

We will consider two types of server speed and power consumption models. In the *idle-speed model*, the server in the UE runs at zero speed when there is no task to perform. Thus, the dynamic power consumption needs to take server utilization into consideration. The average power consumption for computation is $P = \rho_0 P_d + P_s = \rho_0 \xi s_0^\alpha + P_s$, where ρ_0 is the utilization of the server in the UE, which will be available shortly. In the *constant-speed model*, the server in the UE still runs at speed s_0 even if there is no task to perform. Hence, the power consumption for computation is $P = P_d + P_s = \xi s_0^\alpha + P_s$, which is independent of server utilization.

In addition to the above power consumption of the server, the UE also has a data transmission unit which also consumes power. Let P_i be the transmission power of the UE for MEC$_i$, where $1 \le i \le n$. The data transmission rate c_i from the UE to the MEC$_i$ is $c_i = W \log_2(1 + \beta_i P_i)$, where W is the channel bandwidth and β_i is a combined quantity which summarizes various factors such as the channel gain between the UE and MEC$_i$, the interference on the communication channel caused by other devices' data transmission to the same MEC, and the background noise power. Since the average communication time for one offloaded task on MEC$_i$ is $\overline{d}/c_i$, the average energy consumption to complete data transmission for one offloaded task on MEC$_i$ is $P_i(\overline{d}/c_i)$. Thus, the average energy consumption to complete data transmission for one offloaded task on all MECs is

$$\begin{aligned} J &= \sum_{i=1}^{n} \frac{\tilde{\lambda}_i}{\tilde{\lambda}} P_i \frac{\overline{d}}{c_i} = \sum_{i=1}^{n} \frac{\tilde{\lambda}_i}{\tilde{\lambda}} P_i \frac{\overline{d}}{W \log_2(1 + \beta_i P_i)} \\ &= \frac{\overline{d}}{W\tilde{\lambda}} \sum_{i=1}^{n} \tilde{\lambda}_i \frac{P_i}{\log_2(1 + \beta_i P_i)}. \end{aligned}$$

For ease of discussion, we assume that P_i is adjusted in such a way that $P_i / \log_2(1 + \beta_i P_i)$ is a constant γ for all $1 \le i \le n$. Therefore, the average energy consumption to complete data transmission for one task is also a constant $J = \gamma(\overline{d}/W)$ (measured by Joule). Since there are $\tilde{\lambda}$ tasks offloaded per second, the average energy consumption per second (i.e., the average power consumption) for data communication is $\tilde{\lambda} J$, which should be taken into account. Thus, the *average power consumption* of the UE (measured by Watt) for both computation and communication is $P = \rho_0 \xi s_0^\alpha + P_s + \tilde{\lambda} J$ for the idle-speed model, and $P = \xi s_0^\alpha + P_s + \tilde{\lambda} J$ for the constant-speed model. Notice that P is the main cost metric in mobile edge computing.

4 Problem Definitions

Before we define our optimization problems, we derive the average response time of all tasks (offloadable and non-offloadable) generated on the UE. This is the main performance metric in mobile edge computing.

Theorem 1. *The average response time of all tasks generated on the UE is*

$$\begin{aligned} T = {} & \frac{\lambda_0}{\check{\lambda}} \Bigg(\frac{\hat{\lambda}_0}{\lambda_0} \cdot \frac{\overline{r_0}}{s_0} + \frac{\tilde{\lambda}_0}{\lambda_0} \cdot \frac{\overline{r}}{s_0} \\ & + \frac{\hat{\lambda}_0(\overline{r_0^2}/s_0^2) + \tilde{\lambda}_0(\overline{r^2}/s_0^2)}{2(1 - (\hat{\lambda}_0(\overline{r_0}/s_0) + \tilde{\lambda}_0(\overline{r}/s_0)))} \Bigg) \\ & + \sum_{i=1}^{n} \frac{\tilde{\lambda}_i}{\check{\lambda}} \Bigg(\left(\frac{\overline{r}}{s_i} + \frac{\overline{d}}{c_i} \right) \\ & + \frac{\hat{\lambda}_i(\overline{r_i^2}/s_i^2) + \tilde{\lambda}_i(\overline{r^2}/s_i^2 + 2\overline{r}\overline{d}/(s_i c_i) + \overline{d^2}/c_i^2)}{2(1 - (\hat{\lambda}_i(\overline{r_i}/s_i) + \tilde{\lambda}_i(\overline{r}/s_i + \overline{d}/c_i)))} \Bigg). \end{aligned}$$

Proof. Based on the queueing model for the UE in Section 3.1, we know that the execution times of non-offloadable tasks on the UE are i.i.d. random variables with mean $\overline{r_0}/s_0$ and second moment $\overline{r_0^2}/s_0^2$, and that the execution times of offloadable tasks on the UE are i.i.d. random variables with mean $\overline{r}/s_0$ and second moment $\overline{r^2}/s_0^2$. Therefore, the execution times of all tasks on the UE are i.i.d. random variables x_0 with mean

$$\overline{x_0} = \frac{\hat{\lambda}_0}{\lambda_0} \cdot \frac{\overline{r_0}}{s_0} + \frac{\tilde{\lambda}_0}{\lambda_0} \cdot \frac{\overline{r}}{s_0},$$

and second moment

$$\overline{x_0^2} = \frac{\hat{\lambda}_0}{\lambda_0} \cdot \frac{\overline{r_0^2}}{s_0^2} + \frac{\tilde{\lambda}_0}{\lambda_0} \cdot \frac{\overline{r^2}}{s_0^2},$$

where we notice that $\hat{\lambda}_0/\lambda_0$ is the percentage of non-offloadable tasks on the UE, while $\tilde{\lambda}_0/\lambda_0$ is the percentage of offloadable tasks on the UE. The utilization of the server in the UE is

$$\rho_0 = \lambda_0 \overline{x_0} = \hat{\lambda}_0 \frac{\overline{r_0}}{s_0} + \tilde{\lambda}_0 \frac{\overline{r}}{s_0}.$$

The average waiting time of the tasks on the UE is ([14], p. 190)

$$W_0 = \frac{\lambda_0 \overline{x_0^2}}{2(1 - \rho_0)},$$

where

$$\lambda_0 \overline{x_0^2} = \hat{\lambda}_0 \frac{\overline{r_0^2}}{s_0^2} + \tilde{\lambda}_0 \frac{\overline{r^2}}{s_0^2}.$$

The average response time of the tasks on the UE is

$$\begin{aligned} T_0 &= \overline{x_0} + W_0 = \overline{x_0} + \frac{\lambda_0 \overline{x_0^2}}{2(1 - \rho_0)} \\ &= \frac{\hat{\lambda}_0}{\lambda_0} \cdot \frac{\overline{r_0}}{s_0} + \frac{\tilde{\lambda}_0}{\lambda_0} \cdot \frac{\overline{r}}{s_0} \\ &\quad + \frac{\hat{\lambda}_0(\overline{r_0^2}/s_0^2) + \tilde{\lambda}_0(\overline{r^2}/s_0^2)}{2(1 - (\hat{\lambda}_0(\overline{r_0}/s_0) + \tilde{\lambda}_0(\overline{r}/s_0)))}. \end{aligned}$$

Furthermore, based on the queueing model for the MECs in Section 3.1, we know that the execution times of the tasks already processed on MEC$_i$ and not offloaded from the UE are i.i.d. random variables with mean $\overline{r_i}/s_i$ and second moment $\overline{r_i^2}/s_i^2$. The execution times of the tasks offloaded from the UE are i.i.d. random variables

IEEE Proof

$r/s_i + d/c_i$, where r/s_i is the computation time and d/c_i is the communication time. These random variables have mean $\bar{r}/s_i + \bar{d}/c_i$ and second moment $\overline{r^2}/s_i^2 + 2\bar{r}\bar{d}/(s_ic_i) + \overline{d^2}/c_i^2$. Therefore, the execution times of all tasks on MEC$_i$ are i.i.d. random variables x_i with mean

$$\overline{x_i} = \frac{\hat{\lambda}_i}{\lambda_i}\cdot\frac{\overline{r_i}}{s_i} + \frac{\tilde{\lambda}_i}{\lambda_i}\left(\frac{\bar{r}}{s_i} + \frac{\bar{d}}{c_i}\right),$$

and second moment

$$\overline{x_i^2} = \frac{\hat{\lambda}_i}{\lambda_i}\cdot\frac{\overline{r_i^2}}{s_i^2} + \frac{\tilde{\lambda}_i}{\lambda_i}\left(\frac{\overline{r^2}}{s_i^2} + \frac{2\bar{r}\bar{d}}{s_ic_i} + \frac{\overline{d^2}}{c_i^2}\right),$$

where we notice that $\hat{\lambda}_i/\lambda_i$ is the percentage of tasks already processed on MEC$_i$ and not offloaded from the UE, while $\tilde{\lambda}_i/\lambda_i$ is the percentage of tasks offloaded from the UE. The utilization of the server in MEC$_i$ is

$$\rho_i = \lambda_i\overline{x_i} = \hat{\lambda}_i\frac{\overline{r_i}}{s_i} + \tilde{\lambda}_i\left(\frac{\bar{r}}{s_i} + \frac{\bar{d}}{c_i}\right).$$

The average waiting time of the tasks on MEC$_i$ is

$$W_i = \frac{\lambda_i\overline{x_i^2}}{2(1-\rho_i)},$$

where

$$\lambda_i\overline{x_i^2} = \hat{\lambda}_i\frac{\overline{r_i^2}}{s_i^2} + \tilde{\lambda}_i\left(\frac{\overline{r^2}}{s_i^2} + \frac{2\bar{r}\bar{d}}{s_ic_i} + \frac{\overline{d^2}}{c_i^2}\right).$$

The average response time of offloaded tasks on MEC$_i$ is

$$\begin{aligned} T_i &= \left(\frac{\bar{r}}{s_i} + \frac{\bar{d}}{c_i}\right) + W_i = \left(\frac{\bar{r}}{s_i} + \frac{\bar{d}}{c_i}\right) + \frac{\lambda_i\overline{x_i^2}}{2(1-\rho_i)} \\ &= \left(\frac{\bar{r}}{s_i} + \frac{\bar{d}}{c_i}\right) \\ &\quad + \frac{\hat{\lambda}_i(\overline{r_i^2}/s_i^2) + \tilde{\lambda}_i(\overline{r^2}/s_i^2 + 2\bar{r}\bar{d}/(s_ic_i) + \overline{d^2}/c_i^2)}{2(1-(\hat{\lambda}_i(\overline{r_i}/s_i) + \tilde{\lambda}_i(\bar{r}/s_i + \bar{d}/c_i)))}, \end{aligned}$$

for all $1 \le i \le n$.

The *average response time* of all offloadable and non-offloadable tasks generated on the UE is

$$T = \frac{\lambda_0}{\lambda}T_0 + \frac{\tilde{\lambda}_1}{\lambda}T_1 + \frac{\tilde{\lambda}_2}{\lambda}T_2 + \cdots + \frac{\tilde{\lambda}_n}{\lambda}T_n.$$

The theorem is proved by substituting all the T_i's into the last equation, where $0 \le i \le n$. □

In addition to the cost metric P and the performance metric T in mobile edge computing, we can also define the *cost-performance ratio* (i.e., the power-time product) $R = PT$.

Now, we are ready to formally describe our optimization problems to be solved in this paper, which are multi-variable optimization problems.

Minimization of Average Response Time with Average Power Consumption Constraint. Given a UE specified by the parameters $p_1, p_2, \ldots, p_n, \hat{\lambda}_0, \dot{\lambda}, \overline{r_0}, \overline{r_0^2}, \bar{r}, \overline{r^2}, \bar{d}, \overline{d^2}, \xi, \alpha, P_s, J$, and n MECs specified by the parameters $\hat{\lambda}_i, \overline{r_i}, \overline{r_i^2}, s_i, c_i$, where $1 \le i \le n$, and power constraint P^*, find a computation offloading strategy $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$, such that T is minimized, subject to the conditions that $P \le P^*$, $\tilde{\lambda}_i \le \dot{\lambda}_i$, for all $1 \le i \le n$, and $\rho_i < 1$, for all $0 \le i \le n$.

Minimization of Average Power Consumption with Average Response Time Constraint. Given a UE specified by the parameters $p_1, p_2, \ldots, p_n, \hat{\lambda}_0, \dot{\lambda}, \overline{r_0}, \overline{r_0^2}, \bar{r}, \overline{r^2}, \bar{d}, \overline{d^2}, \xi, \alpha, P_s, J$, and n MECs specified by the parameters $\hat{\lambda}_i, \overline{r_i}, \overline{r_i^2}, s_i, c_i$, where $1 \le i \le n$, and performance constraint T^*, find a computation offloading strategy $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$, such that P is minimized, subject to the conditions that $T \le T^*$, $\tilde{\lambda}_i \le \dot{\lambda}_i$, for all $1 \le i \le n$, and $\rho_i < 1$, for all $0 \le i \le n$.

Minimization of Cost-Performance Ratio. Given a UE specified by the parameters $p_1, p_2, \ldots, p_n, \hat{\lambda}_0, \dot{\lambda}, \overline{r_0}, \overline{r_0^2}, \bar{r}, \overline{r^2}, \bar{d}, \overline{d^2}, \xi, \alpha, P_s, J$, and n MECs specified by the parameters $\hat{\lambda}_i, \overline{r_i}, \overline{r_i^2}, s_i, c_i$, where $1 \le i \le n$, find a computation offloading strategy $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$, such that R is minimized, subject to the conditions that $\tilde{\lambda}_i \le \dot{\lambda}_i$, for all $1 \le i \le n$, and $\rho_i < 1$, for all $0 \le i \le n$.

Notice that a computation offloading strategy $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$ is also a *load distribution* of offloadable tasks on the multiple heterogeneous MEC servers. Our optimization problems are actually load balancing problems, such that desired objectives are optimized.

We would like to emphasize that once an optimization problem is solved, i.e., an optimal computation offloading strategy $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$ is found, it can be easily implemented in a real mobile edge computing environment. When a new offloadable task which belongs to $\dot{\lambda}_i$ is generated, it is offloaded to MEC$_i$ with probability $\tilde{\lambda}_i/\dot{\lambda}_i$.

5 The Approach

In this section, we give an outline of our method to solve the three optimization problems.

We would like to mention that the above optimization problems include a subproblem, i.e., the determination of s_0, the execution speed of the UE, which depends on the power constraint P^* or the performance constraint T^*, and the computation offloading strategy $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$, among other factors. However, if we simply view s_0 as a function of $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$, then the problems will be more sophisticated and challenging to solve. We devise a unique method in the following discussion, and based on that, we develop a sequence of efficient numerical algorithms to solve the three optimization problems.

Let us rewrite T in Theorem 1 as a function of $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$

$$\begin{aligned} &T(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n) \\ &= \frac{\dot{\lambda} - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n)}{\dot{\lambda}}\left(\frac{\hat{\lambda}_0}{\dot{\lambda} - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n)}\cdot\frac{\overline{r_0}}{s_0}\right. \\ &\quad + \frac{\dot{\lambda} - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n)}{\dot{\lambda} - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n)}\cdot\frac{\bar{r}}{s_0} \\ &\quad \left. + \frac{\hat{\lambda}_0(\overline{r_0^2}/s_0^2) + (\dot{\lambda} - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n))(\overline{r^2}/s_0^2)}{2(1-(\hat{\lambda}_0(\overline{r_0}/s_0) + (\dot{\lambda} - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n))(\bar{r}/s_0)))}\right) \\ &\quad + \sum_{i=1}^{n}\frac{\tilde{\lambda}_i}{\dot{\lambda}}\left(\left(\frac{\bar{r}}{s_i} + \frac{\bar{d}}{c_i}\right)\right. \\ &\quad \left. + \frac{\hat{\lambda}_i(\overline{r_i^2}/s_i^2) + \tilde{\lambda}_i(\overline{r^2}/s_i^2 + 2\bar{r}\bar{d}/(s_ic_i) + \overline{d^2}/c_i^2)}{2(1-(\hat{\lambda}_i(\overline{r_i}/s_i) + \tilde{\lambda}_i(\bar{r}/s_i + \bar{d}/c_i)))}\right). \end{aligned}$$

IEEE Proof

For the idle-speed model, we represent the UE power consumption P as a function of $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$

$$P(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n) = \left(\hat{\lambda}_0 \frac{\overline{r_0}}{s_0} + (\dot{\lambda} - (\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n)) \frac{\overline{r}}{s_0}\right) \xi s_0^{\alpha} + P_s + \tilde{\lambda} J.$$

For a fixed $\tilde{\lambda} = \tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n$, we know that the condition $P(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n) = P^*$ implies that

$$\left(\hat{\lambda}_0 \frac{\overline{r_0}}{s_0} + (\dot{\lambda} - \tilde{\lambda}) \frac{\overline{r}}{s_0}\right) \xi s_0^{\alpha} + P_s + \tilde{\lambda} J = P^*,$$

which yields

$$s_0 = \left(\frac{P^* - P_s - \tilde{\lambda} J}{\xi(\hat{\lambda}_0 \overline{r_0} + (\dot{\lambda} - \tilde{\lambda}) \overline{r})}\right)^{1/(\alpha-1)}.$$

For the constant-speed model, we also represent the UE power consumption P as a function of $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$

$$P(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n) = \xi s_0^{\alpha} + P_s + \tilde{\lambda} J.$$

For a fixed $\tilde{\lambda}$, we know that the condition $P(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n) = P^*$ implies that

$$\xi s_0^{\alpha} + P_s + \tilde{\lambda} J = P^*,$$

which yields

$$s_0 = \left(\frac{P^* - P_s - \tilde{\lambda} J}{\xi}\right)^{1/\alpha}.$$

Hence, for a given $\tilde{\lambda}$, s_0 becomes available.

Furthermore, we have

$$\begin{aligned} T(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n) &= \frac{\check{\lambda} - \tilde{\lambda}}{\check{\lambda}} T_0 \\ &+ \sum_{i=1}^{n} \frac{\tilde{\lambda}_i}{\tilde{\lambda}} \left(\left(\frac{\overline{r}}{s_i} + \frac{\overline{d}}{c_i} \right) \right. \\ &+ \left. \frac{\hat{\lambda}_i(\overline{r_i^2}/s_i^2) + \tilde{\lambda}_i(\overline{r^2}/s_i^2 + 2\overline{r}\overline{d}/(s_i c_i) + \overline{d^2}/c_i^2)}{2(1 - (\hat{\lambda}_i(\overline{r_i}/s_i) + \tilde{\lambda}_i(\overline{r}/s_i + \overline{d}/c_i)))} \right), \end{aligned} \tag{1}$$

where

$$T_0 = \frac{\hat{\lambda}_0}{\check{\lambda} - \tilde{\lambda}} \cdot \frac{\overline{r_0}}{s_0} + \frac{\dot{\lambda} - \tilde{\lambda}}{\check{\lambda} - \tilde{\lambda}} \cdot \frac{\overline{r}}{s_0} + \frac{\hat{\lambda}_0(\overline{r_0^2}/s_0^2) + (\dot{\lambda} - \tilde{\lambda})(\overline{r^2}/s_0^2)}{2(1 - (\hat{\lambda}_0(\overline{r_0}/s_0) + (\dot{\lambda} - \tilde{\lambda})(\overline{r}/s_0)))},$$

which is entirely known for a fixed $\tilde{\lambda}$. Therefore, for both power consumption models, we essentially need to find $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$, which minimize $T(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$ in Eq. (1), under the constraint that

$$\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n = \tilde{\lambda},$$

for a given $\tilde{\lambda}$. Then we decide the value of $\tilde{\lambda}$, such that $T(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$ is optimized.

All our main algorithms in this paper (i.e., Algorithms 5, 6, 7) are based on several basic algorithms (i.e., Algorithm 1 for finding $\tilde{\lambda}_i$, Algorithm 2 for finding ϕ, and Algorithms 3 and 4 for finding $\tilde{\lambda}$).

6 Minimization of Average Response Time with Average Power Consumption Constraint

In this section, we solve the problem of minimization of average response time with average power consumption constraint.

6.1 A Numerical Method

Our optimization problem to minimize the average response time with average power consumption constraint can be solved by using the method of Lagrange multiplier, namely, $\nabla T(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n) = \phi \nabla F(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$, where $F(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n) = \tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n = \tilde{\lambda}$, that is, $\partial T/\partial \tilde{\lambda}_i = \phi \partial F/\partial \tilde{\lambda}_i = \phi$, for all $1 \le i \le n$, where ϕ is a Lagrange multiplier. Notice that

$$\begin{aligned} \frac{\partial T}{\partial \tilde{\lambda}_i} = \frac{1}{\tilde{\lambda}} \Bigg(& \left(\left(\frac{\overline{r}}{s_i} + \frac{\overline{d}}{c_i} \right) \right. \\ &+ \left. \frac{\hat{\lambda}_i(\overline{r_i^2}/s_i^2) + \tilde{\lambda}_i(\overline{r^2}/s_i^2 + 2\overline{r}\overline{d}/(s_i c_i) + \overline{d^2}/c_i^2)}{2(1 - (\hat{\lambda}_i(\overline{r_i}/s_i) + \tilde{\lambda}_i(\overline{r}/s_i + \overline{d}/c_i)))} \right) \\ &+ \frac{\tilde{\lambda}_i}{\tilde{\lambda}} \left(\frac{\overline{r^2}/s_i^2 + 2\overline{r}\overline{d}/(s_i c_i) + \overline{d^2}/c_i^2}{2(1 - (\hat{\lambda}_i(\overline{r_i}/s_i) + \tilde{\lambda}_i(\overline{r}/s_i + \overline{d}/c_i)))} \right. \\ &+ \left. \frac{(\overline{r}/s_i + \overline{d}/c_i)(\hat{\lambda}_i(\overline{r_i^2}/s_i^2) + \tilde{\lambda}_i(\overline{r^2}/s_i^2 + 2\overline{r}\overline{d}/(s_i c_i) + \overline{d^2}/c_i^2))}{2(1 - (\hat{\lambda}_i(\overline{r_i}/s_i) + \tilde{\lambda}_i(\overline{r}/s_i + \overline{d}/c_i)))^2} \right), \end{aligned}$$

for all $1 \le i \le n$.

In the following, we develop a numerical method (which consists of a series of numerical algorithms) to solve the problem of minimization of average response time with average power consumption constraint.

Algorithm 1. Find $\tilde{\lambda}_i$

Input: $\check{\lambda}, \overline{r}, \overline{r^2}, \overline{d}, \overline{d^2}$, and $\hat{\lambda}_i, \overline{r_i}, \overline{r_i^2}, s_i, c_i$, and ϕ.
Output: $\tilde{\lambda}_i$ such that $\partial T/\partial \tilde{\lambda}_i = \phi$.
Initialize the search interval of $\tilde{\lambda}_i$ as $(0, \tilde{\lambda}_i^*)$;
while (the length of the search interval is $\ge \epsilon$) **do**
  $\tilde{\lambda}_i \leftarrow$ the middle point of the search interval;
  Calculate $\partial T/\partial \tilde{\lambda}_i$;
  if $(\partial T/\partial \tilde{\lambda}_i < \phi)$ **then**
    Change the search interval to the right half;
  else
    Change the search interval to the left half;
  end if
end do;
$\tilde{\lambda}_i \leftarrow$ the middle point of the search interval;
Calculate $\partial T/\partial \tilde{\lambda}_i$;
return $\tilde{\lambda}_i$.

First, for a given ϕ, our numerical algorithm to find $\tilde{\lambda}_i$ such that $\partial T/\partial \tilde{\lambda}_i = \phi$ is given in Algorithm 1. The algorithm uses the classical bisection method based on the observation that $\partial T/\partial \tilde{\lambda}_i$ is an increasing function of $\tilde{\lambda}_i$. As shown in the proof of Theorem 2, $\tilde{\lambda}_i$ is in the range $(0, \tilde{\lambda}_i^*)$, where

$$\tilde{\lambda}_i^* = \left(1 - \hat{\lambda}_i \frac{\overline{r_i}}{s_i}\right)\left(\frac{\overline{r}}{s_i} + \frac{\overline{d}}{c_i}\right)^{-1}.$$

IEEE Proof

(The standard bisection method is described in [4], p. 22). The algorithm terminates when the search interval is shorter than ϵ. We set $\epsilon = 10^{-10}$ in this paper. Let I denote the maximum length of all initial search intervals in this paper. Then, the time complexity of Algorithm 1 is $O(\log(I/\epsilon))$.

Algorithm 2. Find ϕ and $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$

Input: $\tilde{\lambda}, \overline{r}, \overline{r^2}, \overline{d}, \overline{d^2}$, and $\hat{\lambda}_i, \overline{r_i}, \overline{r_i^2}, s_i, c_i$, for all $1 \le i \le n$, and $\tilde{\lambda}$.
Output: ϕ and $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$, such that $\partial T/\partial\tilde{\lambda}_i = \phi$, for all $1 \le i \le n$, and $\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n = \tilde{\lambda}$.

1: Initialize the search interval of ϕ as (0, *ub*), where *ub* is sufficiently large;
2: **while** (the length of the search interval is $\ge \epsilon$) **do**
3: $\phi \leftarrow$ the middle point of the search interval;
4: **for** $i \leftarrow 1$ **to** n **do**
5: Find $\tilde{\lambda}_i$ so that $\partial T/\partial\tilde{\lambda}_i = \phi$ using Algorithm 1;
6: **end do**;
7: **if** ($\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n < \tilde{\lambda}$) **then**
8: Change the search interval to the right half;
9: **else**
10: Change the search interval to the left half;
11: **end if**
12: **end do**;
13: $\phi \leftarrow$ the middle point of the search interval;
14: **for** $i \leftarrow 1$ **to** n **do**
15: Find $\tilde{\lambda}_i$ such that $\partial T/\partial\tilde{\lambda}_i = \phi$ using Algorithm 1;
16: **end do**;
17: **return** ϕ and $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$.

Second, for a given $\tilde{\lambda}$, our numerical algorithm to find ϕ and $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$, such that $\partial T/\partial\tilde{\lambda}_i = \phi$, for all $1 \le i \le n$, and $\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n = \tilde{\lambda}$, is given in Algorithm 2. Again, the algorithm uses the classical bisection method based on the observation that $\tilde{\lambda}_i$ is an increasing function ϕ, and thus $\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_n$ is also an increasing function ϕ. Due to the nested loops and the calling of Algorithm 1, the time complexity of Algorithm 2 is $O(n(\log(I/\epsilon))^2)$.

Algorithm 3. Find $\tilde{\lambda}$

Input: $p_1, p_2, \ldots, p_n, \hat{\lambda}_0, \dot{\lambda}, \overline{r_0}, \overline{r_0^2}, \overline{r}, \overline{r^2}, \overline{d}, \overline{d^2}, \xi, \alpha, P_s, J$, and $\hat{\lambda}_i, \overline{r_i}, \overline{r_i^2}, s_i, c_i$, where $1 \le i \le n$, and power constraint P^*.
Output: $\tilde{\lambda}$ such that T is minimized.

1: Initialize the search interval of $\tilde{\lambda}$ by using Theorem 2;
2: **while** (the length of the search interval is $\ge \epsilon$) **do**
3: $\tilde{\lambda} \leftarrow$ the middle point of the search interval;
4: Call Algorithm 2 to get $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$;
5: Calculate s_0;
6: Calculate $\partial T/\partial\tilde{\lambda}$;
7: **if** ($\partial T/\partial\tilde{\lambda} < 0$) **then**
8: Change the search interval to the right half;
9: **else**
10: Change the search interval to the left half;
11: **end if**
12: **end do**;
13: $\tilde{\lambda} \leftarrow$ the middle point of the search interval;
14: **return** $\tilde{\lambda}$.

Third, we view $T(\tilde{\lambda})$ as a function of $\tilde{\lambda}$. Our numerical algorithm to find $\tilde{\lambda}$ such that $\partial T/\partial\tilde{\lambda} = 0$ (i.e., T is minimized) is given in Algorithm 3. Again, the algorithm uses the classical bisection method based on the observation that T is a convex function, and $\partial T/\partial\tilde{\lambda}$ is an increasing function $\tilde{\lambda}$. Since there is no analytical form of $\partial T/\partial\tilde{\lambda}$, the value of $\partial T/\partial\tilde{\lambda}$ is obtained by calculating

$$\frac{\partial T}{\partial\tilde{\lambda}} = \frac{T(\tilde{\lambda}+\Delta) - T(\tilde{\lambda})}{\Delta},$$

for sufficiently small Δ. We set $\Delta = 10^{-7}$ in this paper. Due to the nested loops and the calling of Algorithm 2, the time complexity of Algorithm 3 is $O(n(\log(I/\epsilon))^3)$.

The determination of the search interval of $\tilde{\lambda}$ is critical to satisfy the many constraints in our optimization problem. The following theorem gives the initial search interval of $\tilde{\lambda}$ in Algorithm 3.

Theorem 2. *$\tilde{\lambda}$ is in the range*

$$\tilde{\lambda} \in \left(\tilde{\lambda}^*, \min\left(\frac{P^* - P_s}{J}, \sum_{i=1}^{n} \min\left(\dot{\lambda}_i, \left(1 - \hat{\lambda}_i \frac{\overline{r_i}}{s_i}\right)\left(\frac{\overline{r}}{s_i} + \frac{\overline{d}}{c_i}\right)^{-1}\right)\right)\right),$$

where $\tilde{\lambda}^*$ is the solution to the equation

$$\hat{\lambda}_0\overline{r_0} + (\dot{\lambda} - \tilde{\lambda})\overline{r} = \left(\frac{P^* - P_s - \tilde{\lambda}J}{\xi}\right)^{1/\alpha},$$

in the range $\tilde{\lambda}^* \in (0, \dot{\lambda})$.

Proof. There are several conditions which $\tilde{\lambda}$ must satisfy.

First, recall that

$$s_0 = \left(\frac{P^* - P_s - \tilde{\lambda}J}{\xi(\hat{\lambda}_0\overline{r_0} + (\dot{\lambda} - \tilde{\lambda})\overline{r})}\right)^{1/(\alpha-1)},$$

for the idle-speed model, and

$$s_0 = \left(\frac{P^* - P_s - \tilde{\lambda}J}{\xi}\right)^{1/\alpha},$$

for the constant-speed model. Since $s_0 > 0$, we get

$$\tilde{\lambda} < \frac{P^* - P_s}{J}.$$

The above condition essentially states that $\tilde{\lambda}$ cannot be too large; otherwise, the UE will not be able to finish data transmissions for offloaded tasks.

Second, the condition $\rho_0 < 1$ requires that

$$\hat{\lambda}_0\frac{\overline{r_0}}{s_0} + (\dot{\lambda} - \tilde{\lambda})\frac{\overline{r}}{s_0} < 1,$$

that is

$$\hat{\lambda}_0\overline{r_0} + (\dot{\lambda} - \tilde{\lambda})\overline{r} < s_0.$$

For the idle-speed model, we have

$$\hat{\lambda}_0\overline{r_0} + (\dot{\lambda} - \tilde{\lambda})\overline{r} < \left(\frac{P^* - P_s - \tilde{\lambda}J}{\xi(\hat{\lambda}_0\overline{r_0} + (\dot{\lambda} - \tilde{\lambda})\overline{r})}\right)^{1/(\alpha-1)},$$

IEEE Proof

that is

$$\hat{\lambda}_0\overline{r_0} + (\dot{\lambda} - \tilde{\lambda})\overline{r} < \left(\frac{P^* - P_s - \tilde{\lambda}J}{\xi}\right)^{1/\alpha}.$$

For the constant-speed model, we get the same inequality. If $\tilde{\lambda}^*$ is the solution to the equation

$$\hat{\lambda}_0\overline{r_0} + (\dot{\lambda} - \tilde{\lambda})\overline{r} = \left(\frac{P^* - P_s - \tilde{\lambda}J}{\xi}\right)^{1/\alpha},$$

in the range $\tilde{\lambda}^* \in (0, \dot{\lambda})$, then we have $\tilde{\lambda} > \tilde{\lambda}^*$.

Third, the condition $\rho_i < 1$ implies that

$$\hat{\lambda}_i\frac{\overline{r_i}}{s_i} + \tilde{\lambda}_i\left(\frac{\overline{r}}{s_i} + \frac{\overline{d}}{c_i}\right) < 1,$$

which yields

$$\tilde{\lambda}_i < \left(1 - \hat{\lambda}_i\frac{\overline{r_i}}{s_i}\right)\left(\frac{\overline{r}}{s_i} + \frac{\overline{d}}{c_i}\right)^{-1},$$

which, together with the condition $\tilde{\lambda}_i \le \dot{\lambda}_i$, gives

$$\tilde{\lambda}_i < \min\left(\dot{\lambda}_i, \left(1 - \hat{\lambda}_i\frac{\overline{r_i}}{s_i}\right)\left(\frac{\overline{r}}{s_i} + \frac{\overline{d}}{c_i}\right)^{-1}\right),$$

for all $1 \le i \le n$. Consequently, we have

$$\tilde{\lambda} < \sum_{i=1}^{n}\min\left(\dot{\lambda}_i, \left(1 - \hat{\lambda}_i\frac{\overline{r_i}}{s_i}\right)\left(\frac{\overline{r}}{s_i} + \frac{\overline{d}}{c_i}\right)^{-1}\right).$$

To summarize the above discussion, we know that $\tilde{\lambda}$ is in the range given in the theorem. □

It is clear that $\tilde{\lambda}^*$ does not accommodate a closed-form expression. However, it can be easily obtained numerically, as shown in Algorithm 4, where we observe that

$$f(\tilde{\lambda}) = (\hat{\lambda}_0\overline{r_0} + (\dot{\lambda} - \tilde{\lambda})\overline{r}) \Bigg/ \left(\frac{P^* - P_s - \tilde{\lambda}J}{\xi}\right)^{1/\alpha},$$

is a decreasing function of $\tilde{\lambda}$. It is clear that the time complexity of Algorithm 4 is $O(\log(I/\epsilon))$.

Algorithm 4. Find $\tilde{\lambda}^*$

Input: $\hat{\lambda}_0, \dot{\lambda}, \overline{r_0}, \overline{r}, P^*, P_s, J, \xi, \alpha$.
Output: $\tilde{\lambda}^*$ such that $f(\tilde{\lambda}^*) = 1$.
1: Initialize the search interval of $\tilde{\lambda}^*$ as $(0, \dot{\lambda})$;
2: **while** (the length of the search interval is $\ge \epsilon$) **do**
3: $\quad\tilde{\lambda}^* \leftarrow$ the middle point of the search interval;
4: $\quad$Calculate $f(\tilde{\lambda}^*)$;
5: $\quad$**if** $(f(\tilde{\lambda}^*) > 1)$ **then**
6: $\quad\quad$Change the search interval to the right half;
7: $\quad$**else**
8: $\quad\quad$Change the search interval to the left half;
9: $\quad$**end if**
10: **end do**;
11: $\tilde{\lambda}^* \leftarrow$ the middle point of the search interval;
12: **return** $\tilde{\lambda}^*$.

Notice that line (6) of Algorithm 3 uses a numerical method to calculate $\partial T/\partial\tilde{\lambda}$. The following theorem gives the accuracy of this method.

Theorem 3. *The $\tilde{\lambda}$ found by Algorithm 3 deviates less than 2Δ from the real value of $\tilde{\lambda}$.*

Proof. Let $\tilde{\lambda}_m$ be the value set in line (3) and $\tilde{\lambda}_r$ be the real value of $\tilde{\lambda}$. The correctness of the Algorithm 3 depends on the decision in line (7). We say that Algorithm 3 makes a correct decision if the outcome of line (7) is consistent with the real sign of $\partial T/\partial\tilde{\lambda}_m$. If every time line (7) makes a correct decision on whether $\partial T/\partial\tilde{\lambda}_m > 0$, then the $\tilde{\lambda}$ found by Algorithm 3 is exactly the real value $\tilde{\lambda}_r$. However, Algorithm 3 may make an incorrect decision due to the calculation of $\partial T/\partial\tilde{\lambda}_m = (T(\tilde{\lambda}_m + \Delta) - T(\tilde{\lambda}_m))/\Delta$. This may happen when $\tilde{\lambda}_m < \tilde{\lambda}_r$. Clearly, $\partial T/\partial\tilde{\lambda}_m < 0$, and the search interval is changed to the right half. However, if $\tilde{\lambda}_r < \tilde{\lambda}_m + \Delta$ and $T(\tilde{\lambda}_m + \Delta) > T(\tilde{\lambda}_m)$, then $\partial T/\partial\tilde{\lambda}_m > 0$ by using the numerical method, and the search interval is changed to the left half. This means that Algorithm 3 will never find $\tilde{\lambda}_r$, i.e., the real value of $\tilde{\lambda}$. Algorithm 3 continues the search and makes correct decisions until the search interval is less than 2Δ. In this case, the middle point $\tilde{\lambda}_m$ of the search interval in line (3) can again be too close to $\tilde{\lambda}_r$, so that Algorithm 3 makes an incorrect decision. However, since the search interval is small enough, the $\tilde{\lambda}$ found by Algorithm 3 deviates less than 2Δ from the real value of $\tilde{\lambda}$. The theorem is proven. □

Finally, we are ready to present our main algorithm for minimization of average response time with average power consumption constraint, which is described in Algorithm 5. The overall time complexity of Algorithm 5 is $O(n(\log(I/\epsilon))^3)$. Algorithm 5 is able to produce management decisions, i.e., an optimal computation offloading strategy $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$, with time complexity independent of the types of tasks.

We would like to mention that due to the efficiency of the bisection method, all our algorithms are extremely fast. The reason is that for all real search intervals, the value of $\log(I/\epsilon)$ is just a small constant.

We also emphasize that Algorithm 5 will be employed to solve the other two optimization problems in Sections 7 and 8. In other words, the three optimization problems in this paper are very closely related.

Algorithm 5. Minimize Average Response Time with Average Power Consumption Constraint

Input: $p_1, p_2, \ldots, p_n, \hat{\lambda}_0, \dot{\lambda}, \overline{r_0}, \overline{r_0^2}, \overline{r}, \overline{r^2}, \overline{d}, \overline{d^2}, \xi, \alpha, P_s, J$, and $\hat{\lambda}_i, \overline{r_i}, \overline{r_i^2}, s_i, c_i$, where $1 \le i \le n$, and power constraint P^*.
Output: $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$ and the minimized T.
1: Call Algorithm 4 to get $\tilde{\lambda}^*$;
2: Call Algorithm 3 to get $\tilde{\lambda}$;
3: Call Algorithm 2 to get $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$;
4: Calculate T;
5: **return** $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$ and T.

6.2 Numerical Examples and Data

In this section, we demonstrate numerical examples and data.

IEEE Proof

TABLE 1
Numerical Data for Minimizing Average Response Time with Average Power Consumption Constraint (Idle-Speed Model)

	0	1	2	3	4	5	6	7
p_i	—	0.0828571	0.1028571	0.1228571	0.1428571	0.1628571	0.1828571	0.2028571
$\dot{\lambda}_i$	—	0.3728571	0.4628571	0.5528571	0.6428571	0.7328571	0.8228571	0.9128571
$\hat{\lambda}_i$	1.0000000	1.5000000	1.4500000	1.4000000	1.3500000	1.3000000	1.2500000	1.2000000
$\overline{r_i}$	0.5000000	1.0000000	1.0500000	1.1000000	1.1500000	1.2000000	1.2500000	1.3000000
$\overline{r_i^2}$	0.4000000	1.3500000	1.5435000	1.7545000	1.9837500	2.2320000	2.5000000	2.7885000
s_i	1.2926435	2.5000000	2.6000000	2.7000000	2.8000000	2.9000000	3.0000000	3.1000000
$\overline{r}/s_i$	1.1604128	0.6000000	0.5769231	0.5555556	0.5357143	0.5172414	0.5000000	0.4838710
$\overline{r_i}/s_i$	0.3868043	0.4000000	0.4038462	0.4074074	0.4107143	0.4137931	0.4166667	0.4193548
$\hat{\lambda}_i(\overline{r_i}/s_i)$	0.3868043	0.6000000	0.5855769	0.5703704	0.5544643	0.5379310	0.5208333	0.5032258
$\tilde{\lambda}_i^*$	—	0.3728571	0.4628571	0.5528571	0.6428571	0.7328571	0.8228571	0.8858407
c_i	—	10.0000000	10.5000000	11.0000000	11.5000000	12.0000000	12.5000000	13.0000000
$\overline{d}/c_i$	—	0.1000000	0.0952381	0.0909091	0.0869565	0.0833333	0.0800000	0.0769231
$\tilde{\lambda}_i$	0.3543583	0.3728571	0.4628571	0.5528571	0.6145553	0.6625006	0.7132343	0.7667800
λ_i	1.3543583	1.8728571	1.9128571	1.9528571	1.9645553	1.9625006	1.9632343	1.9667800
ρ_i	0.7980062	0.8237143	0.8526099	0.8775132	0.8836903	0.8806038	0.8774505	0.8742484
T_i	2.7566227	2.6903135	3.5453376	4.9879970	5.7203121	5.6276726	5.5339270	5.4392547

TABLE 2
Numerical Data for Minimizing Average Response Time with Average Power Consumption Constraint (Constant-Speed Model)

	0	1	2	3	4	5	6	7
p_i	—	0.0828571	0.1028571	0.1228571	0.1428571	0.1628571	0.1828571	0.2028571
$\dot{\lambda}_i$	—	0.3728571	0.4628571	0.5528571	0.6428571	0.7328571	0.8228571	0.9128571
$\hat{\lambda}_i$	1.0000000	1.5000000	1.4500000	1.4000000	1.3500000	1.3000000	1.2500000	1.2000000
$\overline{r_i}$	0.5000000	1.0000000	1.0500000	1.1000000	1.1500000	1.2000000	1.2500000	1.3000000
$\overline{r_i^2}$	0.4000000	1.3500000	1.5435000	1.7545000	1.9837500	2.2320000	2.5000000	2.7885000
s_i	1.1986849	2.5000000	2.6000000	2.7000000	2.8000000	2.9000000	3.0000000	3.1000000
$\overline{r}/s_i$	1.2513714	0.6000000	0.5769231	0.5555556	0.5357143	0.5172414	0.5000000	0.4838710
$\overline{r_i}/s_i$	0.4171238	0.4000000	0.4038462	0.4074074	0.4107143	0.4137931	0.4166667	0.4193548
$\hat{\lambda}_i(\overline{r_i}/s_i)$	0.4171238	0.6000000	0.5855769	0.5703704	0.5544643	0.5379310	0.5208333	0.5032258
$\tilde{\lambda}_i^*$	—	0.3728571	0.4628571	0.5528571	0.6428571	0.7328571	0.8228571	0.8858407
c_i	—	10.0000000	10.5000000	11.0000000	11.5000000	12.0000000	12.5000000	13.0000000
$\overline{d}/c_i$	—	0.1000000	0.0952381	0.0909091	0.0869565	0.0833333	0.0800000	0.0769231
$\tilde{\lambda}_i$	0.3348746	0.3728571	0.4628571	0.5528571	0.6190294	0.6672357	0.7182359	0.7720529
λ_i	1.3348746	1.8728571	1.9128571	1.9528571	1.9690294	1.9672357	1.9682359	1.9720529
ρ_i	0.8361763	0.8237143	0.8526099	0.8775132	0.8860872	0.8830529	0.8799513	0.8767998
T_i	3.6100259	2.6903135	3.5453376	4.9879970	5.9748127	5.8782116	5.7804314	5.6816622

Let us consider a UE with $\hat{\lambda}_0 = 1.0$ tasks/second, $\dot{\lambda} = 4.5$ tasks/second, $\tilde{\lambda} = \hat{\lambda}_0 + \dot{\lambda} = 5.5$ tasks/second, $\overline{r_0} = 0.5$ BI, $\overline{r_0^2} = 0.4$ BI^2, $\overline{r} = 1.5$ BI, $\overline{r^2} = 3.0$ BI^2, $\overline{d} = 1.0$ MB, $\overline{d^2} = 1.5$ MB^2, $\xi = 1.5$, $\alpha = 3.0$, $P_s = 2.0$ Watts, $J = 0.1$ Joules, and $P^* = 5.0$ Watts.

There are $n = 7$ MECs with $p_1 = 1.0/n - 0.06$, $p_2 = 1.0/n - 0.04$, $p_3 = 1.0/n - 0.02$, $p_4 = 1.0/n$, $p_5 = 1.0/n + 0.02$, $p_6 = 1.0/n + 0.04$, $p_7 = 1.0/n + 0.06$. The n MECs are set as $\dot{\lambda}_i = p_i\dot{\lambda}$ tasks/second, $\hat{\lambda}_i = 1.50 - 0.05(i-1)$ tasks/second, $\overline{r_i} = 1.0 + 0.05(i-1)$ BI, $\overline{r_i^2} = (1.35 + 0.05(i-1))\overline{r_i}^2$ BI^2, $s_i = 2.5 + 0.1(i-1)$ GHz, $c_i = 10.0 + 0.5(i-1)$ MB/second, for all $1 \le i \le n$.

In Table 1, we show numerical data for minimizing average response time with average power consumption constraint for the idle-speed model. First, we show p_i and $\dot{\lambda}_i$ for all $1 \le i \le n$. Second, we show $\hat{\lambda}_i$, $\overline{r_i}$, $\overline{r_i^2}$, s_i, for all $0 \le i \le n$. Third, we show $\overline{r}/s_i$ (i.e., the average computation time of an offloadable task on the UE or an MEC$_i$), $\overline{r_i}/s_i$ (i.e., the average computation time of a non-offloadable task on the UE or a preloaded task on MEC$_i$), and $\hat{\lambda}_i(\overline{r_i}/s_i)$ (i.e., the utilization due to non-offloadable tasks on the UE or preloaded tasks on MEC$_i$), for all $0 \le i \le n$. Fourth, we show $\tilde{\lambda}_i^*$ in Algorithm 1 and obtained in the proof of Theorem 2, and c_i, $\overline{d}/c_i$, for all $1 \le i \le n$. Finally, we display the output of our algorithms, including $\tilde{\lambda}_i$ (i.e., the optimal computation offloading strategy), λ_i (i.e., the actual workload on each server), ρ_i (i.e., the utilization of each server), and T_i (i.e., the average response time of each server), for all $0 \le i \le n$.

The search interval of $\tilde{\lambda}$ found by Algorithm 4 is $(4.0328485, 4.4729836)$, and the optimal choice of $\tilde{\lambda}$ found by Algorithm 3 is $\tilde{\lambda} = 4.1456417$ tasks/second. The minimized average response time of all offloadable and non-offloadable tasks generated on the UE obtained by Algorithm 5 is $T = 4.4539410$ seconds.

In Table 2, we show numerical data for minimizing average response time with average power consumption constraint for the constant-speed model. All the data are displayed in the same way as that of Table 1. The search interval of $\tilde{\lambda}$ found by Algorithm 4 is $(4.0328485, 4.4729836)$, and the optimal choice of $\tilde{\lambda}$ found by Algorithm 3 is $\tilde{\lambda} = 4.1651254$ tasks/second. The minimized average response time of all offloadable and non-offloadable tasks generated on the UE obtained by Algorithm 5 is $T = 4.7963025$ seconds.

From both Tables 1 and 2, we make the following observations. (1) MEC$_1$, MEC$_2$, MEC$_3$ receive all the offloadable tasks designated to them, i.e., $\hat{\lambda}_i = \dot{\lambda}_i$, for all $1 \le i \le 3$, due to their

IEEE Proof

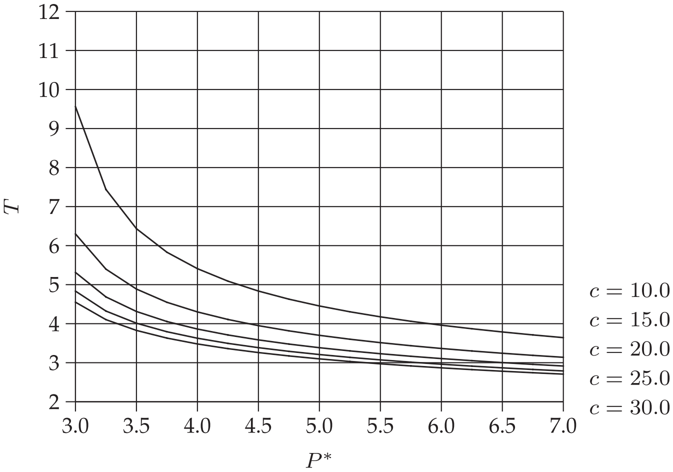

Fig. 2. The average response time T versus the average power consumption P^* (varying c_i, idle-speed model).

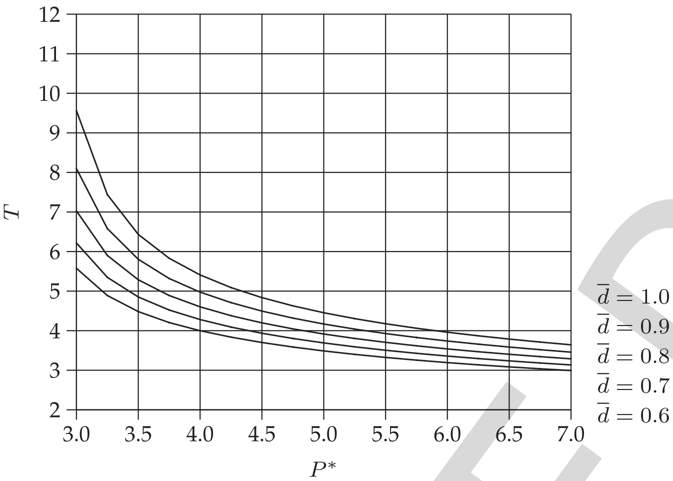

Fig. 3. The average response time T versus the average power consumption P^* (varying $\overline{d}$, idle-speed model).

relatively low $\dot{\lambda}_i$. (2) MEC$_4$, MEC$_5$, MEC$_6$, MEC$_7$ do not receive all the offloadable tasks designated to them, and the remaining offloadable tasks are processed by the UE itself, i.e., $\hat{\lambda}_i < \dot{\lambda}_i$, for all $4 \le i \le 7$, due to their relatively high $\dot{\lambda}_i$. (3) Given the same power constraint P^*, compared with the idle-speed model, the constant-speed model results in reduced s_0 and $\tilde{\lambda}_0$, increased $\tilde{\lambda}$, increased T_i for all $i = 0, 4, 5, 6, 7$, and increased T.

6.3 Power-Performance Tradeoff

In this section, we show the power-performance tradeoff and the impact of various parameters for the idle-speed model.

In Fig. 2, we examine the impact of the speed of data communication on the average response time of all offloadable and non-offloadable tasks generated on the UE for the idle-speed model. We show T as a function of P^* for $c_i = c + 0.5(i-1)$ MB/second, where $c = 10.0, 15.0, 20.0, 25.0, 30.0$ MD/second.

In Fig. 3, we examine the impact of the amount of data communication on the average response time of all offloadable and non-offloadable tasks generated on the UE for the idle-speed model. We show T as a function of P^* for $\overline{d} = 0.6, 0.7, 0.8, 0.9, 1.0$ MD.

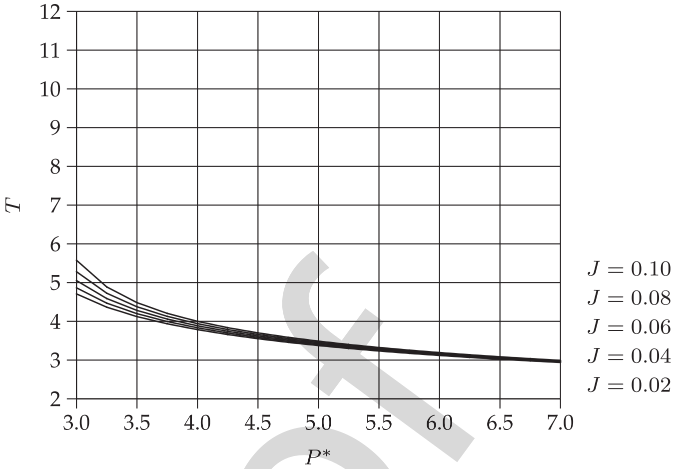

Fig. 4. The average response time T versus the average power consumption P^* (varying J, idle-speed model).

In Fig. 4, we examine the impact of the energy consumption of data communication on the average response time of all offloadable and non-offloadable tasks generated on the UE for the idle-speed model. We show T as a function of P^* for $J = 0.02, 0.04, 0.06, 0.08, 0.10$ Joules.

We have the following observations. (1) These figures all demonstrate the power-performance tradeoff, i.e., more (less, respectively) average power consumption results in shorter (longer, respectively) average response time. (2) These figures also demonstrate the impact of various parameters. Figs. 2 and 3 show that for the same power constraint, increasing the speed of data communication or decreasing the amount of data communication results in noticeable decrement in the average response time. The reason is that the processing times of offloaded tasks on all the MECs are reduced. (3) However, Fig. 4 shows that decreasing the energy consumption of data communication only increases the speed of the UE and slightly decreases the average response time.

(Due to space limitation, our numerical data to show the power-performance tradeoff and the impact of various parameters for the constant-speed model are moved to Section 2.1 of the supplementary file, available online.)

7 Minimization of Average Power Consumption with Average Response Time Constraint

In this section, we solve the problem of minimization of average power consumption with average response time constraint.

7.1 A Numerical Algorithm

Our optimization problem to minimize average power consumption with average response time constraint can be solved by using the algorithms in Section 6.1. Our numerical method is given in Algorithm 6. The algorithm uses the classical bisection method based on the observation that T obtained by Algorithm 5 is a decreasing function of P^*. The overall time complexity of Algorithm 6 is $O(n(\log(I/\epsilon))^4)$.

IEEE Proof

TABLE 3
Numerical Data for Minimizing Average Power Consumption with Average Response Time Constraint (Idle-Speed Model)

	0	1	2	3	4	5	6	7
p_i	—	0.0828571	0.1028571	0.1228571	0.1428571	0.1628571	0.1828571	0.2028571
$\dot{\lambda}_i$	—	0.3728571	0.4628571	0.5528571	0.6428571	0.7328571	0.8228571	0.9128571
$\hat{\lambda}_i$	1.0000000	1.5000000	1.4500000	1.4000000	1.3500000	1.3000000	1.2500000	1.2000000
$\overline{r_i}$	0.5000000	1.0000000	1.0500000	1.1000000	1.1500000	1.2000000	1.2500000	1.3000000
$\overline{r_i^2}$	0.4000000	1.3500000	1.5435000	1.7545000	1.9837500	2.2320000	2.5000000	2.7885000
s_i	1.4382873	2.5000000	2.6000000	2.7000000	2.8000000	2.9000000	3.0000000	3.1000000
$\overline{r}/s_i$	1.0429071	0.6000000	0.5769231	0.5555556	0.5357143	0.5172414	0.5000000	0.4838710
$\overline{r_i}/s_i$	0.3476357	0.4000000	0.4038462	0.4074074	0.4107143	0.4137931	0.4166667	0.4193548
$\hat{\lambda}_i(\overline{r_i}/s_i)$	0.3476357	0.6000000	0.5855769	0.5703704	0.5544643	0.5379310	0.5208333	0.5032258
$\tilde{\lambda}_i^*$	—	0.3728571	0.4628571	0.5528571	0.6428571	0.7328571	0.8228571	0.8858407
c_i	—	10.0000000	10.5000000	11.0000000	11.5000000	12.0000000	12.5000000	13.0000000
$\overline{d}/c_i$	—	0.1000000	0.0952381	0.0909091	0.0869565	0.0833333	0.0800000	0.0769231
$\tilde{\lambda}_i$	0.4168637	0.3728571	0.4628571	0.5528571	0.6002005	0.6473098	0.6971892	0.7498654
λ_i	1.4168637	1.8728571	1.9128571	1.9528571	1.9502005	1.9473098	1.9471892	1.9498654
ρ_i	0.7823858	0.8237143	0.8526099	0.8775132	0.8760003	0.8727464	0.8694279	0.8660639
T_i	2.3854845	2.6903135	3.5453376	4.9879970	5.0370935	4.9551026	4.8722000	4.7885371

TABLE 4
Numerical Data for Minimizing Average Power Consumption with Average Response Time Constraint (Constant-Speed Model)

	0	1	2	3	4	5	6	7
p_i	—	0.0828571	0.1028571	0.1228571	0.1428571	0.1628571	0.1828571	0.2028571
$\dot{\lambda}_i$	—	0.3728571	0.4628571	0.5528571	0.6428571	0.7328571	0.8228571	0.9128571
$\hat{\lambda}_i$	1.0000000	1.5000000	1.4500000	1.4000000	1.3500000	1.3000000	1.2500000	1.2000000
$\overline{r_i}$	0.5000000	1.0000000	1.0500000	1.1000000	1.1500000	1.2000000	1.2500000	1.3000000
$\overline{r_i^2}$	0.4000000	1.3500000	1.5435000	1.7545000	1.9837500	2.2320000	2.5000000	2.7885000
s_i	1.4281480	2.5000000	2.6000000	2.7000000	2.8000000	2.9000000	3.0000000	3.1000000
$\overline{r}/s_i$	1.0503113	0.6000000	0.5769231	0.5555556	0.5357143	0.5172414	0.5000000	0.4838710
$\overline{r_i}/s_i$	0.3501038	0.4000000	0.4038462	0.4074074	0.4107143	0.4137931	0.4166667	0.4193548
$\hat{\lambda}_i(\overline{r_i}/s_i)$	0.3501038	0.6000000	0.5855769	0.5703704	0.5544643	0.5379310	0.5208333	0.5032258
$\tilde{\lambda}_i^*$	—	0.3728571	0.4628571	0.5528571	0.6428571	0.7328571	0.8228571	0.8858407
c_i	—	10.0000000	10.5000000	11.0000000	11.5000000	12.0000000	12.5000000	13.0000000
$\overline{d}/c_i$	—	0.1000000	0.0952381	0.0909091	0.0869565	0.0833333	0.0800000	0.0769231
$\tilde{\lambda}_i$	0.4419413	0.3728571	0.4628571	0.5508388	0.5949043	0.6417054	0.6912701	0.7436259
λ_i	1.4419412	1.8728571	1.9128571	1.9508388	1.9449043	1.9417054	1.9412701	1.9436259
ρ_i	0.8142796	0.8237143	0.8526099	0.8763919	0.8731630	0.8698476	0.8664684	0.8630448
T_i	2.8427462	2.6903135	3.5453376	4.9037552	4.8260847	4.7473874	4.6678381	4.5875794

Algorithm 6. Minimize Average Power Consumption with Average Response Time Constraint

Input: $p_1, p_2, \ldots, p_n, \hat{\lambda}_0, \dot{\lambda}, \overline{r_0}, \overline{r_0^2}, \overline{r}, \overline{r^2}, \overline{d}, \overline{d^2}, \xi, \alpha, P_s, J$, and $\hat{\lambda}_i, \overline{r_i}, \overline{r_i^2}, s_i, c_i$, where $1 \le i \le n$, and performance constraint T^*.
Output: $(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n)$ and the minimized P.

```
Initialize the search interval of P as (lb, ub);
while (the length of the search interval is ≥ ϵ) do
    P* ← the middle point of the search interval;
    Get T by using Algorithm 5;
    if (T > T*) then
        Change the search interval to the right half;
    else
        Change the search interval to the left half;
    end if
end do;
P ← the middle point of the search interval;
return (λ̃_1, λ̃_2, ..., λ̃_n) and P.
```

To find the *lb* in the algorithm, we notice that since $s_0 > 0$, we need $P^* > P_s + \tilde{\lambda}J$. Because $\tilde{\lambda} > \tilde{\lambda}^*$, where $\tilde{\lambda}^*$ is defined in the proof of Theorem 2 and can be found by using Algorithm 4, we set $\mathrm{lb} = P_s + \tilde{\lambda}^* J$.

7.2 Numerical Examples and Data

In this section, we demonstrate numerical examples and data.

We use the same UE and MEC parameter setting in Section 6.2. Let $T^* = 4.0$ seconds.

In Tables 3 and 4, we show numerical data for minimizing average power consumption with average response time constraint for the idle-speed model and the constant-speed model respectively. For the idle-speed model, we get $P = 5.9001117$ Watts and $\tilde{\lambda} = 4.0831363$ tasks/second. For the constant-speed model, we get $P = 6.7750964$ Watts and $\tilde{\lambda} = 4.0580587$ tasks/second. As expected, to achieve the same performance goal T^*, the constant-speed model consumes more power than the idle-speed model.

From both Tables 3 and 4, we make the following observations. (1) Lower indexed MECs receive all the offloadable tasks designated to them, due to their relatively low $\dot{\lambda}_i$. (2) Higher indexed MECs do not receive all the offloadable tasks designated to them, and the remaining offloadable tasks are processed by the UE itself, due to their relatively high $\dot{\lambda}_i$. (3) Given the same performance constraint T^*, compared with the idle-speed model, the constant-speed model results in reduced s_0, increased $\tilde{\lambda}_0$ and increased T_0, reduced $\tilde{\lambda}$, reduced T_i for all $i = 3, 4, 5, 6, 7$, and increased P.

IEEE Proof

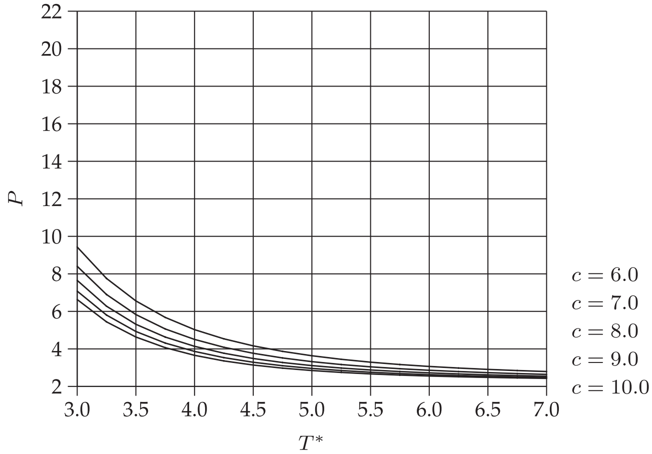

Fig. 5. The average power consumption P versus the average response time T^* (varying c_i, idle-speed model).

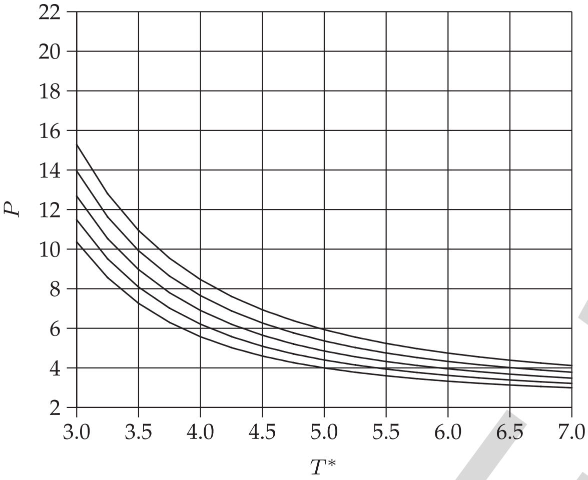

Fig. 6. The average power consumption P versus the average response time T^* (varying $\overline{d}$, idle-speed model).

7.3 Power-Performance Tradeoff

In this section, we show the power-performance tradeoff and the impact of various parameters for the idle-speed model.

In Fig. 5, we examine the impact of the speed of data communication on the average power consumption of all offloadable and non-offloadable tasks generated on the UE for the idle-speed model. We show P as a function of T^* for $c_i = c + 0.5(i-1)$ MB/second, where $c = 6.0, 7.0, 8.0, 9.0, 10.0$ MD/second.

In Fig. 6, we examine the impact of the amount of data communication on the average power consumption of all offloadable and non-offloadable tasks generated on the UE for the idle-speed model. We show P as a function of T^* for $\overline{d} = 1.0, 1.1, 1.2, 1.3, 1.4$ MD.

In Fig. 7, we examine the impact of the energy consumption of data communication on the average power consumption of all offloadable and non-offloadable tasks generated on the UE for the idle-speed model. We show P as a function of T^* for $J = 0.10, 0.12, 0.14, 0.16, 0.18$ Joules.

We have the following observations. (1) These figures all demonstrate the power-performance tradeoff, i.e., longer (shorter, respectively) average response time results in less (more, respectively) average power consumption. (2) These figures also demonstrate the impact of various parameters.

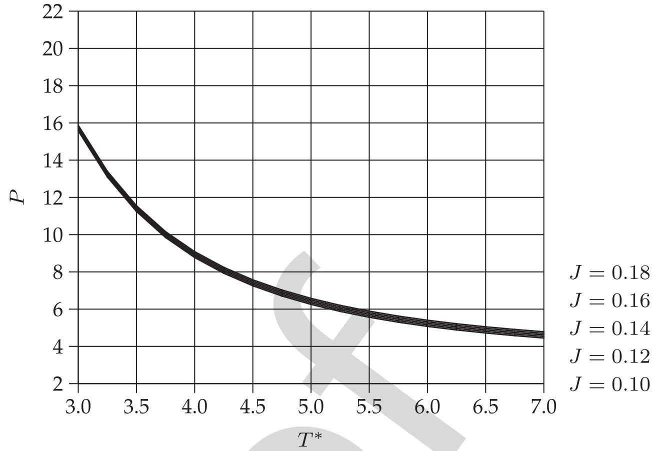

Fig. 7. The average power consumption P versus the average response time T^* (varying J, idle-speed model).

Figs. 5 and 6 show that for the same performance constraint, decreasing the speed of data communication or increasing the amount of data communication results in noticeable increment in the average power consumption. The reason is that the processing times of offloaded tasks on all the MECs are increased. To keep the same average response time, the speed of the UE should be significantly enhanced to handle increased amount of offloadable but not offloaded tasks. (3) However, Fig. 7 shows that increasing the energy consumption of data communication only slightly increases the average power consumption of the UE.

(Due to space limitation, our numerical data to show the power-performance tradeoff and the impact of various parameters for the constant-speed model are moved to Section 2.2 of the supplementary file, available online.)

8 Minimization of Cost-Performance Ratio

In this section, we solve the problem of minimization of the cost-performance ratio. (Due to space limitation, this section is moved to Section 3 of the supplementary file, available online.)

9 Concluding Remarks

We have considered computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing. Our approach is to establish a queueing model for a UE and multiple MECs, and also power consumption models for the UE, so that the discussion on cost and performance can be carried out rigorously. In particular, we have formulated and investigated three multi-variable optimization problems, i.e., minimization of average response time with average power consumption constraint, minimization of average power consumption with average response time constraint, and minimization of cost-performance ratio. We have developed effective and efficient numerical algorithms to solve the three problems. We have also demonstrated numerical examples and data to show the effectiveness of our method and to show the power-performance tradeoff, the power-time product, and the impact of various parameters. Since our models involve parameters easily available for any UE and MECs in any real-world scenario and our methods

IEEE Proof

are computationally very efficient, the proposed computation offloading strategy optimizations in this paper can be applied to any real-world applications in mobile edge computing and fog computing.

There are still some issues worth of further investigation. First, in our model, it is assumed that an MEC server performs communication with the UE and then does the computation for each offloaded task. It is clear that enhanced performance can be achieved by overlapping communication and computation, i.e., communication is performed while a task is waiting. However, it is very challenging to analytically study such a sophisticated situation. Second, it is interesting to consider the case when each MEC server is a multicore server and requires a multiserver queueing model (e.g., M/G/m). Finally, it is of practical interest and importance to implement the optimizing techniques developed in this paper in real application environments.

Acknowledgments

The author is thankful to the four anonymous reviewers for their comments. The research was supported in part by National Natural Science Foundation of China under Grant No. 61876061, the Key Program of National Natural Science Foundation of China under Grant No. 61432005, and the National Key Research and Development Program of China under Grant No. 2018YFB1003401.

References

[1] (2019, Jan.). [Online]. Available: http://en.wikipedia.org/wiki/CMOS

[2] (2019, Jan.). [Online]. Available: https://en.wikipedia.org/wiki/Mobile_edge_computing

[3] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *Proc. 10th IEEE Int. Conf. Intell. Syst. Control*, Jan. 7/8, 2016, pp. 1–8.

[4] R. L. Burden, J. D. Faires, and A. C. Reynolds, *Numerical Analysis*, 2nd ed. Boston, MA, USA: Prindle, Weber & Schmidt, 1981.

[5] H. Cao and J. Cai, "Distributed multi-user computation offloading for Cloudlet based mobile cloud computing: A game-theoretic machine learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 752–764, Jan. 2018.

[6] V. Cardellini, V. De Nitto Personé, V. Di Valerio, F. Facchinei, V. Grassi, F. L. Presti, and V. Piccialli, "A game-theoretic approach to computation offloading in mobile cloud computing," *Math. Program.*, vol. 157, no. 2, pp. 421–449, 2016.

[7] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[8] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, Apr. 2015.

[9] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[10] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Services*, Jun. 15–18, 2010, pp. 49–62.

[11] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing – A key technology towards 5G," ETSI White Paper No. 11, European Telecommunications Standards Institute, Sep. 2015.

[12] Intel, *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor – White Paper*, Mar. 2004.

[13] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," *Mobile Computing, Applications, and Services*, M. Griss and G. Yang, Eds. Berlin, Germany: Springer, 2012, pp. 59–79.

[14] L. Kleinrock, *Queueing Systems: Theory*, vol. 1. New York, NY, USA: Wiley, 1975.

[15] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan./Feb. 2018.

[16] X. Ma, C. Lin, X. Xiang, and C. Chen, "Game-theoretic analysis of computation offloading for Cloudlet-based mobile cloud computing," in *Proc. 18th ACM Int. Conf. Model. Anal. Simul. Wireless Mobile Syst.*, Nov. 2–6, 2015, pp. 271–278.

[17] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Jul.–Sep. 2017.

[18] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[19] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Mar. 19–22, 2017, pp. 1–6.

[20] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Power-delay tradeoff in multi-user mobile-edge computing systems," in *Proc. IEEE Global Commun. Conf.*, Dec. 4–8, 2016, pp. 1–6.

[21] M. Patel, et al., "Mobile-edge computing – Introductory technical white paper," Sep. 2014. [Online]. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf

[22] H. Shah-Mansouri, V. W. S. Wong, and R. Schober, "Joint optimal pricing and task scheduling in mobile cloud computing systems," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 5218–5232, Aug. 2017.

[23] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "COSMOS: Computation offloading as a service for mobile devices," in *Proc. 15th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Aug. 11–14, 2014, pp. 287–296.

[24] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, "Performance guaranteed computation offloading for mobile-edge cloud computing," *IEEE Wireless Commun. Lett.*, vol. 6, no. 6, pp. 774–777, Dec. 2017.

[25] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *arXiv:1705.00704v1*, May 1, 2017. [Online]. Available: https://arxiv.org/abs/1705.00704

[26] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.

[27] H. Wu, "Multi-objective decision-making for mobile cloud offloading: A survey," *IEEE Access*, vol. 6, pp. 3962–3976, 2018.

[28] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.

[29] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "Theoretical and practical limits of dynamic voltage scaling," in *Proc. 41st Des. Autom. Conf.*, 2004, pp. 868–873.

[30] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

Keqin Li is a SUNY distinguished professor of computer science with the State University of New York. He is also a distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous comuting systems, big data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent, and soft computing. He has published more than 630 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He currently serves or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is a fellow of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.