Received: 11 July 2020 | Revised: 11 November 2020 | Accepted: 10 December 2020

DOI: 10.1002/spe.2951

SPECIAL ISSUE PAPER

WILEY

Server configuration optimization in mobile edge computing: A cost-performance tradeoff perspective

Zhenli He[1,2] | **Kenli Li**[1] | **Keqin Li**[1,3] | **Wei Zhou**[2]

[1]College of Computer Science and Electronic Engineering & National Supercomputing Center in Changsha, Hunan University, Changsha, China

[2]College of Software, Yunnan University, Kunming, China

[3]Department of Computer Science, State University of New York, New Paltz, New York, USA

Correspondence
Keqin Li, College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China.
Email: lik@newpaltz.edu

Funding information
National Natural Science Foundation of China, Grant/Award Number: 61876061; Yunnan Applied Basic Research Projects, Grant/Award Number: 202001BB050034

Abstract

Before service providers build up an mobile edge computing (MEC) platform, an important issue that needs to be considered is the configuration of computing resources on edge servers. Since the computing resources on an edge server are limited compared with a cloud server and the service provider's deployment budget is limited, it would be unrealistic to equip all edge servers with abundant computing resources. In addition, the edge servers have different computation demands due to their different geographies. Therefore, this article investigates the problem of server configuration optimization in an MEC environment based on a given computation demand statistics of the selected deployment locations. Our strategy is to treat each edge server as an M/M/m queueing model, and then establish the performance and cost models for the system. Two optimization problems, including cost constrained performance optimization, and performance constrained cost optimization are formulated based on our models and solved by a series of fast numerical algorithms. We also conduct extensive numerical simulation examples to show the effectiveness of the proposed algorithms. MEC service providers can use our strategy to get the appropriate type of processor and obtain the optimal processor number for each edge server to achieve two different goals: (1) deliver the highest-quality services with a given cost constraint; (2) minimize the investment cost with a service-quality guarantee. Our research is of great significance for service providers to control the tradeoff between investment cost and service quality.

KEYWORDS

cost-performance tradeoff, edge server, mobile edge computing, queueing model, server configuration

1 | INTRODUCTION

1.1 | Motivation

In recent years, mobile applications have greatly facilitated people's daily life. People use various portable services provided by mobile applications to socialize with others, conduct business transactions, handle official business, and entertain themselves.[1] In conjunction with this, mobile devices (MDs, including smartphones, handheld computers, and

other vertical application devices) have been becoming increasingly smarter and powerful to provide more computing capability for mobile applications. Nevertheless, the computing capability of MDs is still limited due to the constraints of portable devices in terms of space, battery capacity, weigh, and heat dissipation.[2] With the increasing demand for complex functionalities of portable services, mobile applications have been becoming gradually energy-hungry and computation-intensive.[3,4] Offloading the computation-intensive tasks of mobile applications from MDs to resource-rich clouds is an effective approach to address this problem, such that mobile users can obtain the high-performance services and at the same time effectively extend the battery life of devices.

However, the data centers of clouds are usually located at the core of the Internet which are geographically far away from mobile users, the increasing mobile data traffic make huge burden on the backbone network.[5] With the prevalence of MDs and mobile applications, the transmission delays introduced by the network have become a new obstacle.[6] This is unbearable for some powerful mobile applications (e.g., multiplayer online games, large-scale image processing, personal assistant) that are sensitive to task response time. In fact, this challenge will be more severe. According to a recent forecast by Cisco, the total number of mobile users will grow from 5.1 billion (66% of the global population) in 2018 to 5.7 billion (71% of the global population) by 2023.[7]

As a promising computing paradigm, mobile edge computing (MEC) has been proposed to overcome this challenge. MEC offers high-quality, high-bandwidth, and low-latency services at the edge of mobile networks by deploying edge servers within or in close proximity to mobile base stations.[8,9] MDs connect directly to edge servers through mobile networks, and offload parts of their tasks (i.e., offloadable tasks) to edge servers within one hop, thus releasing the load on the backbone network and extending their battery life.[10] The services provided by an MEC platform have unparalleled quality of experience when compared to the services provided by a cloud computing platform.[11,12] Therefore, MEC is attracting increasing attention from academia and industry.

Before service providers build up an MEC platform, an important issue that needs to be considered is the configuration of computing resources on edge servers.[13] Since the computing resources on an edge server are limited compared with a cloud server[14] and the service provider's deployment budget is limited, it would be unrealistic to equip all edge servers with abundant computing resources. In addition, edge servers have different computation demands due to their different geographies. Some available computing resources will be idling when the computing resources equipped by the edge server exceed its computing requirements. These idle computing resources will generate a lot of wasteful power consumption, thereby increasing the operating cost of the service provider. According to a recent study, the basic energy consumption of the server in the idle state accounts for more than 60% of the server energy consumption in the full state.[15] Therefore, how to optimize the computing resource configuration of edge servers (i.e., server configuration optimization) is of great significance for MEC service providers to control the tradeoff between investment cost and service quality, and is a critically key problem to be solved in the development of MEC.

1.2 | Our contributions

In this article, we investigate the problem of server configuration optimization in an MEC environment, where the main objective is to optimize the number and type of processors that will be configured on edge servers based on a given computation demand statistics. We consider such a typical application scenario from the perspective of a service provider that we will deploy edge servers at selected base stations to establish an MEC platform. Our investment budget is limited, and we hope that the system can provide a service-quality guarantee after it is established. Therefore, we mainly consider two key factors, namely, the system performance and the investment cost.

We only focus on system performance rather than application performance in this work. This is mainly because the portable services provided by mobile applications are various in an MEC environment, and the MEC service provider need to consider the performance from an overall perspective, that is, system performance. We use the average response time as the system performance metric. On the other hand, the service provider's investment cost consists of multiple parts, such as equipment purchases, operating cost, maintenance cost, utility cost, and so on. These costs have different dimensional units. In order to solve this problem, we assume that the economic life of an MEC platform is 3 years, and then use the unit cost as the investment cost metric in this article.

However, optimizing both performance and cost may be conflicting requirements. The improvement of performance is often accompanied by an increase in power consumption, and the power consumption is an important part of our investment cost. Hence, we consider the tradeoff between investment cost and system performance, that is, minimization of average response time with a given system unit cost constraint, and minimization of system unit cost with a given

average response time constraint. The fundamental purpose of our study is to find an optimal configuration for edge servers to reduce redundant investment cost and energy consumption caused by idle computing resources, and to meet the potential computation demands of different deployment locations as much as possible to reduce communication delay caused by task migration among multiple edge servers.

It should be noted that the computation demand statistics of different deployment locations are assumed to be known in this work. Although the workloads of edge servers are different and dynamically changed due to their different geographies and the mobility of MDs, we think that a time-series analysis of the historical mobile data traffic of base stations can be used to predict the stable computation demands of edge servers over a long period of time, which is not within the scope of this article. In fact, our strategy is to solve a static optimization problem of server configuration in order to obtain the optimal server configuration scheme before system deployment, and we can use the scheme for a long time until the environment changes significantly. When the computation demands faced by edge servers have changed a lot after a period of time, we can reuse the strategy to obtain another scheme and use it again.

Our major contributions in this article can be summarized as follows.

- We establish an M/M/m queueing model with infinite waiting queue capacity to characterize multiple heterogeneous edge servers, and then establish a performance model and a cost model for the MEC system respectively, such that the performance and investment cost of the system can be calculated analytically.
- We formulate two optimization problems, that is, *cost constrained performance optimization* (i.e., minimization of average response time with a given system unit cost constraint), and *performance constrained cost optimization* (i.e., minimization of system unit cost with a given average response time constraint), such that the server configuration optimization problem of edge servers from the perspective of cost-performance tradeoff can be studied. We also extend these two problems to the case where the service provider has multiple types of processors provided by different hardware suppliers for selection before constructing the MEC system.
- We design a series of fast numerical algorithms based on the bisection algorithm to address the above problems. MEC service providers can use our algorithms to get the appropriate type of processor and obtain the optimal processor number for each edge server to achieve two different goals: (1) deliver the highest-quality services with a given cost constraint; (2) minimize the investment cost with a service-quality guarantee.
- We also conduct extensive numerical simulation examples to show the effectiveness of the proposed algorithms.

To the best of our knowledge, this work is the first research of server configuration optimization in MEC which considers the cost-performance tradeoff. Therefore, the results of our research provide theoretical and practical contributions for the computing resource configuration problem in MEC, which can be applied to the deployment of MEC platforms.

The rest of the article is arranged as follows. In Section 2, we review related work and summarize the unique features of our research. In Section 3, we define two optimization problems (i.e., *cost constrained performance optimization* and *performance constrained cost optimization*) to be studied in the context of server configuration and then establish system models based on queuing theory. In Sections 4, we formulate the above optimization problems, and then develop a numerical method and a series of algorithms for each problem. In Sections 5, we conduct extensive numerical simulation examples to demonstrate the effectiveness of the proposed algorithms. In Section 6, we conclude this article and show some prospects for future study.

2 | RELATED RESEARCH

MEC has two fundamental purposes, namely, improving performance and reducing cost or power consumption,[16] since low performance can limit the functionality of mobile applications and reduce the quality of user experience, while high energy consumption can cause severe economic losses and environmental problems. Specifically, there are several essential factors that can affect the performance and power consumption of an MEC platform, such as the amount of available computing resources, the amount of idle computing resources, the amount of task offloading, the scheduling of offloading tasks, the CPU-cycle frequencies of edge servers and MDs, and the transmission/communication speed for computation offloading, and so on. Therefore, there is a huge body of literature studied how to optimize performance and energy consumption by considering different factors from different entry points, such as *computation offloading optimization*, *power allocation optimization*, *dynamic resource allocation optimization*, and so on. In this section, we mainly review related

research from these three entry points. The reader is referred to References 11-15,17,18 for related research based on other entry points and detailed surveys.

Computation offloading optimization. The principle of computation offloading is to weigh the benefits of executing tasks on MDs and edge servers to partition, schedule tasks reasonably, thereby speeding up computing and saving energy.[6] There is a lot of literature studied this issue with respect to task partitioning, allocation, and execution. Lyu et al.[19] studied computation offloading strategy on a three-layer architecture of cloud, edge server, and devices. The authors developed a computation offloading algorithm that can minimizes the energy consumption of devices while meeting the latency requirements of MDs. Tham et al.[20] proposed a load balancing scheme in an MEC environment to minimize the response time of offloadable tasks while still satisfying the wireless channel capacity and link contention constraints. In Reference 21, the authors implemented a task offloading strategy based on power consumption and bandwidth capacity of different time slots by using KKT conditions. In Reference 22, the authors designed a distributed algorithm for multiuser computation offloading to minimize the system-wide computation overhead. Huang et al.[23] designed a computation offloading algorithm to minimize system power consumption while ensuring the quality of service. A similar problem were studied in References 24,25. In Reference 26, Li established an M/G/1 queueing model with infinite waiting queue capacity to characterize multiple heterogeneous MDS and edge servers, such that the performance and energy consumption of the MEC platform can be calculated analytically. He studied computation offloading strategy optimization in an MEC environment to balance power and performance. This work used two different CPU core speed models, that is, the idle-speed model and the constant-speed model, to analyze the energy consumption of MDs, thereby increasing the application scope the strategy. He also studied computation offloading strategy in the case of multiple mobile users playing noncooperative game in an MEC environment, and designed an algorithm that can find the Nash equilibrium to address this problem.[27]

Power allocation optimization. In an MEC environment, it is important to focus not only on the energy consumption of remote infrastructures but also on the energy consumption of MDs which is related to their battery life. Therefore, it is necessary to make reasonably power allocation for both edge servers and MDs. Mao et al.[28] developed an effective strategy aimed at improving system performance and achieving green computing to jointly obtain optimal scheduling for offloadable tasks, optimal power allocation for MDs, and optimal transmission power allocation. Their algorithms are implemented by dynamically adjusting the CPU-cycle frequencies of MDs, the offloading decision, and transmit power of MDs. In Reference 29, the authors proposed an optimization framework of offloading which jointly optimizes the task allocation and the CPU-cycle frequencies of MDs to minimize both response time and MD's power consumption. Zhang et al.[30] developed a task scheduling algorithm to jointly obtain optimal frequencies for MDs' processor and optimal transmission power allocation, thereby managing the power-performance tradeoff. Li[16] established an M/G/1 queueing model to characterize multiple MDs and an M/G/m queueing model to characterize an edge server. He conducted a quantitative study on the stabilization of a competitive mobile edge computing environment by established a noncooperative game framework. In his work, all MDs and the mobile edge cloud can obtain an optimal action that minimize their payoff by dynamically adjusting the CPU-cycle frequencies, the offloading decision, and transmission power allocation.

Dynamic resource allocation optimization. Dynamic resource allocation optimization in MEC is also an effective way to improve performance and reduce cost. There are many types of resources that can be dynamically allocated in an MEC environment, including server caches, communication resource (e.g., wireless bandwidth, communication channel), and computing resource (e.g., containers and virtual machines). Zhang et al.[31] proposed a strategy to manage the tradeoff between energy and latency in the MEC network through joint optimization of computation and communication resource allocation. Xu et al.[32] studied the problem of service caching optimization in the MEC network and implemented an algorithm to determine which services need to be cached on edge servers to improve system performance with a given energy consumption constraint. In Reference 33, the authors developed an algorithm to improve offloading gains by jointly optimizing task scheduling, transmission power of MDs, and resource allocation of edge servers. However, the computing resource only abstractly represented by computational rates of edge servers in their work. Ma et al.[34] designed an algorithm of optimal resource provisioning which jointly determines the utilization of cloud resources and the optimal edge computation capacity to minimize the cost of service provisioning. However, this work only uses a simple parameter to abstract edge computation capacity. Mao et al.[35] developed an algorithm of dynamic resource allocation to jointly obtain optimal transmit power and bandwidth allocation for computation offloading to make task offloading effective. Zhou et al.[36] designed a strategy to maximize energy efficiency and minimize SLA violation rates by dynamically adjusting CPU and memory resources during VM deployment. Paščinski et al.[37] designed a new autonomic orchestration architecture and implemented a global cluster manager (GCM) to monitor the selected QoS metrics of related applications in real time, thereby automatically selecting the best geographically available computing resources within the software-defined data centers (SDDCs), and finally achieving high QoS.

TABLE 1 A comparative table with existing research

	Applicable stage	Main constraints	Optimization objectives
Computation offloading optimization	MEC has been established	Performance constraint, energy consumption constraint	Amount of task offloading, scheduling of offloading tasks, offloading decision
Power allocation optimization	MEC has been established	Performance constraint, energy consumption constraint	CPU-cycle frequencies, transmit power of MDs
Dynamic resource allocation optimization	MEC has been established	Resource constraint, performance constraint, energy consumption constraint	Service caching policies, channel allocation, placement or capability of VMs or containers
Our investigation	Before deployment	Performance constraint, cost constraint	Number and type of processors that will be configured on all edge servers

From the above literature review, we find that most of existing research focus on different optimization objectives based on the premise that the MEC platform has been deployed. Our work is different from the above studies in that we consider server configuration strategy from the perspective of a service provider before system deployment. In order to clearly position our investigation and highlight the difference between our study and existing research, we make a comparison in Table 1.

Our investigation has the following unique features.

- We consider an application scenario from the perspective of service providers, that is, we will deploy multiple edge servers in selected base stations to build up an MEC platform. Since our investment budget is limited and the system needs to provide a service-quality guarantee after it is established, we need to obtain the optimal number and type of processors that will be configured on edge servers based on a well-analyzed and predicted computation demand statistics.
- We establish an M/M/m queueing model to characterize multiple edge servers, such that to conduct a rigorous analysis of the average response time of offloading tasks, including task waiting time and task processing time.
- We analyze the investment cost from multiple aspects including equipment purchases (i.e., processors), utility cost, maintenance cost, and operating cost (i.e., energy consumption of edge servers' processors), and use unit cost to unify the different dimensional units of these costs.
- Our strategy can provide an optimal server configuration scheme to reduce redundant investment cost and energy consumption caused by idle computing resources, and communication delay caused by task migration among multiple edge servers.

We would like to mention that our strategy does not conflict with other optimization methods (including computation offloading optimization, power allocation optimization, and dynamic resource allocation optimization), since our strategy and other optimization methods are applied in different stages of the MEC system implementation. Before system deployment, MEC service providers can control the tradeoff between investment cost and system performance according to our strategy, and can quantitatively analyze the performance and cost of the system. After all edge servers are deployed with appropriate computing resources (i.e., the MEC platform has been established), other optimization methods can be applied to further improve system performance or save energy based on other footholds.

3 | PROBLEM DEFINITION AND MODELS

3.1 | Problem definition

We consider such a typical application scenario from the perspective of a service provider that we will deploy n edge servers $S_1, S_2, \ldots, S_n$ at the n selected base stations to establish an MEC platform. In addition, we assume that the stable computation demand statistics of n base stations $\lambda_1, \lambda_2, \ldots, \lambda_n$ are obtained through a time-series prediction analysis based on the historical mobile data traffic. Our investment budget is limited, and we hope that the system can provide a service-quality guarantee after it is established. We aim to address the following two problems.

- *Cost constrained performance optimization*. Under the given cost constraint $\tilde{C}$ and the above conditions, find the optimal number and type of processors that will be configured on each edge server, such that the average response time of the system is minimized.
- *Performance constrained cost optimization*. Under the given performance constraint $\tilde{T}$ and the above conditions, find the optimal number and type of processors that will be configured on each edge server, such that the unit cost of system is minimized.

For example, Figure 1 indicates a small MEC environment. In the environment, we will deploy two edge servers (represented by $S1$ and $S2$ in Figure 1) at the selected base stations (represented by A and B in Figure 1) to establish the system. These selected base stations are located at regions with high population density (such as commercial and residential areas) and have higher potential computation demands. The base stations (represented by C and D in Figure 1) that do not deploy edge servers will migrate tasks from MDs within their coverage to the nearest edge server for computing (illustrated by the red dotted arrows in Figure 1). Assume that the stable computation demand of A (including the task migration of C) is λ_A, and the stable computation demand of B (including the task migration of D) is λ_B. Our main objective is to find the optimal number and type of processors for the two edge servers based on its potential computing demands λ_A, λ_B, such that the average response time of the system is minimized while the investment cost does not exceed the given cost constraint, or the unit cost of system is minimized while the average response time does not exceed the given performance constraint.

In order to solve the above two problems, and to conduct a rigorous analysis of the performance and investment cost of the MEC platform, we first need to establish mathematical models.

3.2 | Edge server model

In the MEC environment, each edge server is configured with different numbers of processors and has different computing capabilities, thus the edge servers are heterogeneous. We use an M/M/m queueing model to characterize multiple heterogeneous edge servers. Taking m_i as the number of identical processors configured on server S_i, that is, server processors have the same execution speed s, which is measured in units of billion instructions per second (BIPS). The edge servers are modeled as follows.

Each edge server maintains a first-in-first-out (FIFO) queue with infinite capacity to cache waiting tasks when the server is busy and adopts the first-come-first-served (FCFS) queueing discipline for arrival tasks. Since the memory of servers has become more and more sufficient with the development of modern computer technology, and the task response times in an MEC system are not very large, the overflow of server buffer might be ignored to certain extent. In addition, the arrival time of offloadable tasks are randomness, and it is difficult to clearly know their probability distribution. However, the occurrence of many things in nature obeys the Poisson distribution, and many other probability distributions can also be approximated by the Poisson distribution. Thus, we assume that each edge server accepts an

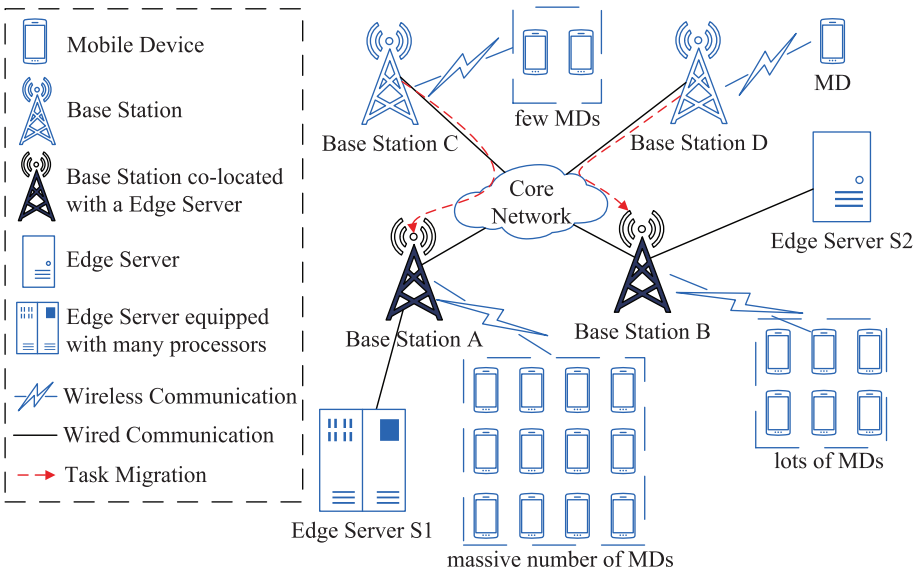

FIGURE 1 An example of an MEC environment

independent stream of tasks (including user tasks within its coverage area and the tasks migrated from other base stations) with a Poisson distribution, that is, the average arrival rate of tasks to S_i is λ_i and the interarrival times of tasks to S_i are a sequence of independent and identically distributed (i.i.d.) exponential random variables with the average value $1/\lambda_i$. The number of instructions (measured in units of billion instructions) to be executed for offloadable tasks are a sequence of i.i.d. exponential random variables r with the average value $\overline{r}$. Thus, the task execution times of processors (measured in seconds) have an average value of $\overline{x} = \overline{r}/s$, that is, an offloadable task is assigned to an idle processor for execution until it is completed.

According to queuing theory, we can get the average service rate of each edge server as $\mu = 1/\overline{x} = s/\overline{r}$. The value of μ represents the average number of executed tasks per processor per second. We also obtain the server utilization of S_i as:

$$\rho_i = \frac{\lambda_i}{m_i\mu} = \frac{\lambda_i\overline{x}}{m_i} = \frac{\lambda_i\overline{r}}{m_i s}, \tag{1}$$

which represents the average fraction of the time that S_i is busy. Notice that the value of $m_i\rho_i = \lambda_i\overline{x} = \lambda_i\overline{r}/s$ represents the average number of processors on S_i that are executing tasks.

3.3 | Performance model

In this section, we discuss how to characterize the performance of an MEC system. Specifically, we use the average response time as the performance metric.

According to queuing theory [38, p. 102], the probability that there are k tasks with the states of waiting or executing in the M/M/m queueing system for the server S_i is

$$p_{i,k} = \begin{cases} p_{i,0}\frac{(m_i\rho_i)^k}{k!}, & k \le m_i; \\ p_{i,0}\frac{m_i^{m_i}\rho_i^k}{m_i!}, & k \ge m_i; \end{cases} \tag{2}$$

where

$$p_{i,0} = \left(\sum_{k=0}^{m_i-1}\frac{(m_i\rho_i)^k}{k!} + \frac{(m_i\rho_i)^{m_i}}{m_i!}\cdot\frac{1}{1-\rho_i}\right)^{-1}. \tag{3}$$

Then, we obtain the probability that a newly arrived task must be queued as

$$P_{q,i} = \sum_{k=m_i}^{\infty} p_{i,k} = \frac{p_{i,m_i}}{1-\rho_i} = p_{i,0}\frac{{m_i}^{m_i}}{m_i!}\cdot\frac{{\rho_i}^{m_i}}{1-\rho_i}. \tag{4}$$

Let L_i denote the average number of tasks in S_i. Then we have

$$L_i = \sum_{k=0}^{\infty} k p_{i,k} = m_i\rho_i + \frac{\rho_i}{1-\rho_i}P_{q,i}. \tag{5}$$

Applying Little's law, the average task response time of S_i is

$$T_i = \frac{L_i}{\lambda_i} = \overline{x} + \frac{P_{q,i}}{m_i(1-\rho_i)}\overline{x} = \overline{x}\left(1 + \frac{P_{q,i}}{m_i(1-\rho_i)}\right). \tag{6}$$

Through the above analysis, we can get the average response time of the system (measured in seconds) as

$$T = \sum_{i=1}^{n}\frac{\lambda_i}{\lambda}T_i = \frac{\lambda_1}{\lambda}T_1 + \frac{\lambda_2}{\lambda}T_2 + \ \dots\ + \frac{\lambda_n}{\lambda}T_n, \tag{7}$$

Notice that Equation (7) is actually a function of server sizes $m_1, m_2, \dots, m_n$, that is, $T(m_1, m_2, \dots, m_n)$.

3.4 | Cost model

In this section, an analytical model is established to characterize the investment cost of an MEC system.

The investment cost mainly consists of two parts, namely, the *static cost* and the *dynamic cost*. The *static cost* mainly comes from the purchase of hardware equipments and the site rentals of edge servers, which also includes maintenance cost and utility (such as A/C) cost. However, we do not consider the impact of the site rentals. This is mainly because the site rentals are determined when the deployment locations are selected. The *dynamic cost* mainly comes from the energy consumption of edge servers' processors. Notice that the *static cost* and *dynamic cost* have different dimensional units. In order to solve this problem, we assume that the economic life of an MEC platform is 3 years. We use the unit cost of the system as the cost metric in this article.

Let C_s denote the static cost of the system. We have

$$C_s = (1+\omega)c_p\sum_{i=1}^{n} m_i, \tag{8}$$

where c_p denotes the price of the processor with speed s (measured in CNY) and ω denotes the ratio coefficient of maintenance and utility costs (i.e., the maintenance and utility costs are linearly related to the sizes of edge servers). Thus, the static unit cost of the system is

$$\overline{C}_s = (1+\omega)c_p\sum_{i=1}^{n} m_i \cdot \frac{1}{\kappa}, \tag{9}$$

where $\kappa = 94{,}608{,}000$ is a constant (3 years $\cdot$ 365 days $\cdot$ 24 h $\cdot$ 60 min $\cdot$ 60 s) denotes the total number of seconds in 3 years.

The processor power consumption mainly consists of three parts, namely, dynamic, static, and short-circuits power dissipation. The dynamic power consumption is the dominant component and can be approximated as

$$P_d = aC_{es}{V_p}^2 f_p,$$

where a is an activity factor, C_{es}, V_p, and f_p are the effective switching capacitance, supply voltage, and clock frequency of the processor, respectively.[39] For the processor speed s, we have $s \propto f_p$. Since $P_d \propto f_p^{\alpha}$ where α is approximately equal to 3,[40] we use s^{α} to denote the dynamic power consumption of the processor with speed s for ease of discussion. Notice that we set $\alpha = 3$ in all numerical examples in this article.

In addition, the dynamic power consumption of processor is related to its working model. According to some existing research,[26,41,42] processors generally have two working modes namely, the *idle-speed model* and the *constant-speed model*. When there is no task to execute, the processor speed is zero for the idle-speed model, but is s for the constant-speed model. Since the processor type/working model that the service provider intend to use are uncertain, we use the two types of core speed models in this article to improve the applicability of our strategy.

Based on Equation (1), we formulate the average power consumption of a processor with speed s (measured in watts/second) as

$$P_i = m_i(\rho_i s^{\alpha} + P^*) = \lambda_i \overline{r} s^{\alpha-1} + m_i P^*, \tag{10}$$

for the idle-speed model and as

$$P_i = m_i(s^{\alpha} + P^*), \tag{11}$$

for the constant-speed model, where P^* denotes the basic power of the processor with speed s (including static and short-circuits power dissipation). Notice that the processor speed in Equation (10) is zero when the processor is idle, since s^{α} is multiplied by the utilization of S_i. Hence, we can get the dynamic unit cost of the system as

$$\overline{C}_d = c_e\sum_{i=1}^{n} P_i, \tag{12}$$

where c_e expresses the price of electricity per watt per second (measured in CNY). According to the above discussion, we can get the unit cost of the system (measured in CNY per second) as

$$\overline{C} = \overline{C}_s + \overline{C}_d = (1+\omega)c_p \sum_{i=1}^{n} m_i \cdot \frac{1}{\kappa} + c_e \sum_{i=1}^{n} P_i. \tag{13}$$

By straightforward algebraic manipulation, we get

$$\overline{C} = \sum_{i=1}^{n} \left(\frac{(1+\omega)m_i c_p}{\kappa} + \left(\lambda_i \overline{r} s^{\alpha-1} + m_i P^*\right) c_e \right) = \sum_{i=1}^{n} m_i \cdot \left(\frac{(1+\omega)c_p}{\kappa} + c_e P^* \right) + \lambda \overline{r} s^{\alpha-1} c_e, \tag{14}$$

for the idle-speed model and

$$\overline{C} = \sum_{i=1}^{n} \left(\frac{(1+\omega)m_i c_p}{\kappa} + m_i \left(s^{\alpha} + P^*\right) c_e \right) = \sum_{i=1}^{n} m_i \cdot \left(\frac{(1+\omega)c_p}{\kappa} + \left(s^{\alpha} + P^*\right) c_e \right), \tag{15}$$

for the constant-speed model, where $\lambda = \lambda_1 + \lambda_2 + \ldots + \lambda_n$ denotes the total task arrival rate of the system. Notice that Equations (14) and (15) is actually the functions of server sizes $m_1, m_2, \ldots, m_n$, that is, $\overline{C}(m_1, m_2, \ldots, m_n)$.

The mathematical notation that will be used throughout the following sections are shown in Table 2.

TABLE 2 Mathematical notations

Symbol	Definition
n	Number of edge servers
$S_1, S_2, \ldots, S_n$	Edge servers
m_i	Number of processors configured on S_i
s	Processor speed
λ_i	Arrival rate of offloadable tasks to S_i
λ	Total task arrival rate of the system
$\overline{r}$	Average task size
$\overline{x} = \overline{r}/s$	Average task execution time of processors
$\mu = 1/\overline{x}$	Average service rate
$\rho_i = \lambda_i \overline{x}/m_i$	Utilization of S_i, where $\rho_i < 1$, for all $1 \le i \le n$
$p_{i,k}$	Probability that S_i has k tasks
$P_{q,i}$	Probability of queueing for S_i
$\overline{N}_i$	Average number of tasks in S_i
T_i	Average task response time of S_i
T	$\frac{\lambda_1}{\lambda} T_1 + \frac{\lambda_2}{\lambda} T_2 + \ldots \frac{\lambda_n}{\lambda} T_n$.
c_p	Price of the processor with speed s
c_e	Price of electricity per watt per second
ω	Ratio coefficient of maintenance and utility costs
P^*	Basic power of the processor
κ	94,608,000
$\overline{C}$	Unit cost of the system
$\tilde{C}$	System unit cost constraint
$\tilde{T}$	Average response time constraint

4 | OUR SOLUTIONS

In this section, we formulate the two optimization problems and develop a numerical method and a series of algorithms to solve each problem.

4.1 | Cost constrained performance optimization

In this section, we solve the problem of cost constrained performance optimization, that is, to minimize

$$T(m_1, m_2, \ldots, m_n) = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda} T_i = \frac{\lambda_1}{\lambda} T_1 + \frac{\lambda_2}{\lambda} T_2 + \ldots + \frac{\lambda_n}{\lambda} T_n$$

(which represents the weighted mean of the task response time of all edge servers), subject to the constraints $\rho_i < 1$ (since the load of each edge server must not exceed its capacity), for all $1 \le i \le n$, that is, $m_i > \lambda_i \overline{x} = \lambda_i \overline{r}/s$; and $\overline{C} = \tilde{C}$ (which represents the investment cost must be close to the given cost constraint).

In order to facilitate the derivation, we make a simple transformation of Equations (14) and (15), and use a new function $G(m_1, m_2, \ldots, m_n)$ to represent $\overline{C} = \tilde{C}$, namely,

$$G(m_1, m_2, \ldots, m_n) = \sum_{i=1}^{n} m_i = \frac{\tilde{C} - \lambda \overline{r} s^{\alpha-1} c_e}{\frac{(1+\omega)c_p}{\kappa} + c_e P^*}, \tag{16}$$

for the idle-speed model, and

$$G(m_1, m_2, \ldots, m_n) = \sum_{i=1}^{n} m_i = \frac{\tilde{C}}{\frac{(1+\omega)c_p}{\kappa} + (s^\alpha + P^*) c_e}, \tag{17}$$

for the constant-speed model.

We utilize the Lagrange multiplier method to solve this optimization problem. However, this method views the server sizes $m_1, m_2, \ldots, m_n$, as a series of continuous values, but the number of processors can only be a positive integer. We will discuss how to solve this problem in the algorithm design, and temporarily treat the server size as a continuous variable.

Using the Lagrange multiplier method, we derive the Lagrange function of the optimization problem as

$$\nabla T(m_1, m_2, \ldots, m_n) = \phi \nabla G(m_1, m_2, \ldots, m_n),$$

that is, n equations

$$\frac{\partial T(m_1, m_2, \ldots, m_n)}{\partial m_i} = \phi \frac{\partial G(m_1, m_2, \ldots, m_n)}{\partial m_i},$$

for all $1 \le i \le n$, where ϕ is a Lagrange multiplier. Based on Equations (7), (16), and (17), we have

$$\frac{\partial T}{\partial m_i} = \frac{\lambda_i}{\lambda} \cdot \frac{\partial T_i}{\partial m_i} = \phi, \tag{18}$$

for all $1 \le i \le n$. Notice that the above $\partial G / \partial m_i = 1$, which is the same for both the idle-speed model and the constant-speed model.

We directly use the result from a previous work,[43] that is,

$$\frac{\partial T_i}{\partial m_i} = -\frac{\overline{r}}{s} \cdot \frac{\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i}\left(\frac{\rho_i+3}{2} - m_i(1-\rho_i)\ln\rho_i\right) + 1}{m_i{}^2(1-\rho_i)^2\left(\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i} + 1\right)^2} \tag{19}$$

(In order to ensure the readability of this article, we provide the detailed derivation of the above equation in the Appendix).

Based on Equations (18) and (19), we can get

$$\frac{\partial T}{\partial m_i} = -\frac{\lambda_i \overline{r}}{\lambda s} \cdot \frac{\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i}\left(\frac{\rho_i+3}{2} - m_i(1-\rho_i)\ln\rho_i\right)+1}{{m_i}^2(1-\rho_i)^2\left(\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i}+1\right)^2} = \phi. \tag{20}$$

It suffices to show that T is a descending function of m_i (i.e., $\partial T/\partial m_i < 0$), since each part of Equation (20) is positive except for the minus sign at the beginning, that is, $\lambda_i\overline{r}/\lambda s > 0$, $\sqrt{2\pi m_i}(1-\rho_i) > 0$, $(e^{\rho_i}/e\rho_i)^{m_i} > 0$, $(\rho_i+3)/2 > 0$, and $-m_i(1-\rho_i)\ln\rho_i > 0$, where $\rho_i < 1$ (i.e., $1-\rho_i > 0$ and $\ln\rho_i < 0$).

In addition, based on Equations (14) and (15), it is clear that $\overline{C}$ is an increasing function of m_i. Therefore, with the increase of m_i, the performance of the system will improve, but the investment cost will increase.

Notice that Equation (20) is actually a function of m_i, namely, $\phi(m_i)$. It is difficult to get the inverse function of $\phi(m_i)$. However, from our observations, we find that T is a convex function of m_i and $\phi(m_i)$ is an increasing function of m_i, since the growth rate of the numerator is lower than that of the denominator in Equation (20), and $\partial T/\partial m_i < 0$.

To further illustrate the above observations, we take the second partial derivative $\partial^2 T/\partial {m_i}^2$, that is,

$$\frac{\partial^2 T}{\partial {m_i}^2} = -\frac{\lambda_i \overline{r}}{\lambda s}\left(2F_i\frac{\partial F_i}{\partial m_i}Q_i + {F_i}^2\frac{\partial Q_i}{\partial m_i}\right), \tag{21}$$

where

$$Q_i = \sqrt{2\pi m_i}(1-\rho_i)H_i\left(\frac{\rho_i+3}{2} - m_i(1-\rho_i)\ln\rho_i\right)+1. \tag{22}$$

Since

$$\frac{\partial F_i}{\partial m_i} = -{F_i}^2 Q_i,$$

Equation (21) can rewrite as

$$\frac{\partial^2 T}{\partial {m_i}^2} = -\frac{\lambda_i \overline{r}}{\lambda s}\left(-2{F_i}^3{Q_i}^2 + {F_i}^2\frac{\partial Q_i}{\partial m_i}\right). \tag{23}$$

It is clear that

$$\begin{aligned}\frac{\partial Q_i}{\partial m_i} = {} & \sqrt{\frac{\pi}{2m_i}}(1-\rho_i)H_i\left(\frac{\rho_i+3}{2} - m_i(1-\rho_i)\ln\rho_i\right) + \frac{\sqrt{2\pi m_i}\rho_i}{m_i}H_i\left(\frac{\rho_i+3}{2} - m_i(1-\rho_i)\ln\rho_i\right) \\ & - H_i\ln\rho_i\cdot\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{\rho_i+3}{2} - m_i(1-\rho_i)\ln\rho_i\right) + \left(1-\ln\rho_i - \frac{\rho_i(2m_i+1)}{2m_i}\right)\sqrt{2\pi m_i}(1-\rho_i)H_i.\end{aligned} \tag{24}$$

In fact, $\partial^2 T/\partial {m_i}^2$ is a function of m_i. However, it is difficult to analyze the positive–negative of $\partial^2 T/\partial {m_i}^2$ based on mathematical derivation. Therefore, we assign 2.0 to λ_1, 4.0 to λ_2, … , 10.0 to λ_5, respectively, that is, $\lambda = 30$, and then obtain the corresponding $\partial^2 T/\partial {m_i}^2$ in the range of m_i from $\lceil \lambda_i\overline{r}/s \rceil$ (since $m_i > \lambda_i\overline{r}/s$) to 50. By drawing the points on the coordinate axis, we get Figure 2. We can observe that $\partial^2 T/\partial {m_i}^2$ approaches gradually to 0 from a positive value, with the increases of m_i. It is consistent with our previous observations.

Since it is difficult to obtain a closed-form solution for the optimization problem, in this section, we propose a numerical solution based on the bisection algorithm to search ϕ and m_i within a certain range, respectively.

From Equation (1), we get $m_i > \lambda_i\overline{r}/s$ for all $1 \le i \le n$. Thus, the lower bound of m_i is

$$lb_i = \lceil \lambda_i\overline{r}/s \rceil.$$

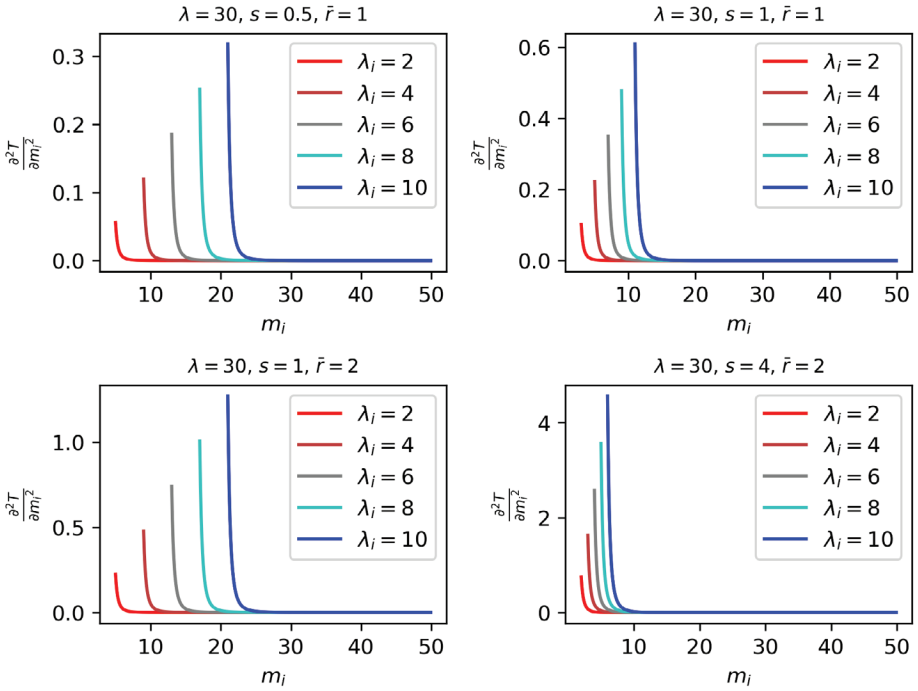

FIGURE 2 Several examples of $\partial^2 T/\partial m_i^2$

Since $\phi(m_i)$ is an increasing function of m_i, let lb_{max} denote the maximum of lb_i, that is,

$$lb_{max} = \max_{1 \le i \le n} (lb_i),$$

based on Equation (20), we obtain the lower bound of ϕ as

$$lb_\phi = \phi(lb_{max}).$$

The upper bound of m_i (namely, ub_i) and the upper bound of ϕ (namely, ub_ϕ) can be obtained by doubling the corresponding lower bound repeatedly until $\overline{C} > \tilde{C}$.

Summarizing the above discussion, given n, λ_i, s, $\overline{r}$, c_p, c_e, ω, P^*, and $\tilde{C}$, our algorithms for finding the optimal processor configuration $m_1, m_2, \ldots, m_n$ to minimize the average response time of the system with the given cost constraint $\tilde{C}$ can be described by Algorithms 1–4.

Algorithm 1 describes the steps that find the lower bound of ϕ, that is, lb_ϕ, and provides a basis for the subsequent algorithms.

Algorithm 1. *Find_lb$_\phi$*

Input: $n, \lambda_1, \lambda_2, \cdots, \lambda_n, s, \bar{r}$.
Output: lb_ϕ.
$lb_\phi, lb_{max} \leftarrow 0$; $\lambda \leftarrow \lambda_1 + \lambda_2 + \cdots + \lambda_n$; //Initialize the parameters
for i in *range*(n): //Find the maximum of lb_i, for all $1 \le i \le n$
  $lb_i \leftarrow \lceil \lambda_i \bar{r}/s \rceil$;
  if $lb_i > lb_{max}$:
    $lb_{max} \leftarrow lb_i$;
  end if
end for
$lb_\phi \leftarrow \phi(lb_{max})$; //This value is calculated by Eq. (20), where $m_i \leftarrow lb_{max}$
return lb_ϕ. //Return the lower bound of ϕ

Algorithm 2. *Find_m_i_with_ϕ*

Input: $\lambda_i, s, \bar{r}, \phi$.
Output: m_i.
$\varepsilon \leftarrow$ a very small quantity; $lb_i, ub_i \leftarrow \lceil \lambda_i \bar{r}/s \rceil$; //Initialize ε, lb_i, and ub_i
while $\phi(ub_i) < \phi$: //The value of $\phi(ub_i)$ is calculated by Eq. (20), where $m_i \leftarrow ub_i$
  $ub_i \leftarrow 2ub_i$; //Find the upper bound of m_i by doubling itself until the calculated value of ϕ is greater than the given value
end while
while $ub_i - lb_i > \varepsilon$: //Search m_i in $\left[lb_i, ub_i\right]$ through the bisection algorithm
  $mid \leftarrow (lb_i + ub_i)/2$;
  if $\phi(mid) < \phi$: //The value of $\phi(mid)$ is calculated by Eq. (20), where $m_i \leftarrow mid$
    $lb_i \leftarrow mid$;
  else:
    $ub_i \leftarrow mid$;
  end if
end while
$m_i \leftarrow (lb_i + ub_i)/2$;
return m_i. //Return the processor number of S_i when ϕ is given

Algorithm 3. *Find_config_with_cost_constraint*

Input: $\tilde{C}, n, \lambda_1, \lambda_2, \cdots, \lambda_n, s, \bar{r}, c_p, c_e, \omega, P^*, idle$. //The *idle* is a Boolean representing the working model of the processor
Output: $m_1, m_2, \cdots, m_n$. //A server configuration scheme with a set of continuous values
$\varepsilon \leftarrow$ a very small quantity; $lb_\phi, \phi \leftarrow Find\_lb_\phi(n, \lambda_1, \lambda_2, \cdots, \lambda_n, s, \bar{r})$; $\bar{C} \leftarrow 0$; //Initialize the parameters
while $\bar{C} < \tilde{C}$: //Find the upper bound of ϕ by doubling itself until system unit cost exceeds the given cost constraint
  $\phi \leftarrow \phi/2$; //Since $\phi < 0$, the doubling operation is actually a division by 2.
  for i in $range(n)$:
    $m_i \leftarrow Find\_m_i\_with\_\phi(\lambda_i, s, \bar{r}, \phi)$;
  end for
  $\bar{C} \leftarrow idle$? the value calculated by Eq. (14) : the value calculated by Eq. (15);
end while
$ub_\phi \leftarrow \phi$;
while $ub_\phi - lb_\phi > \varepsilon$: //Search ϕ in $[lb_\phi, ub_\phi]$ through the bisection algorithm
  $mid \leftarrow \left(lb_\phi + ub_\phi\right)/2$;
  for i in $range(n)$: //Obtain the server configuration scheme according to the new value of ϕ (i.e., mid)
    $m_i \leftarrow Find\_m_i\_with\_\phi(\lambda_i, s, \bar{r}, mid)$;
  end for
  $\bar{C} \leftarrow idle$? the value calculated by Eq. (14) : the value calculated by Eq. (15);
  if $\bar{C} > \tilde{C}$:
    $ub_\phi \leftarrow mid$;
  else:
    $lb_\phi \leftarrow mid$;
  end if
end while
$\phi \leftarrow \left(lb_\phi + ub_\phi\right)/2$;
for i in $range(n)$: //Obtain the server configuration scheme according to the determined ϕ
  $m_i \leftarrow Find\_m_i\_with\_\phi(\lambda_i, s, \bar{r}, \phi)$;
end for
return $m_1, m_2, \cdots, m_n$.

Algorithm 4. *Discretize_config_and_Calculate_T*

Input: $m_1, m_2, \cdots, m_n, \tilde{C}, n, \lambda_1, \lambda_2, \cdots, \lambda_n, s, \bar{r}, c_p, c_e, \omega, P^*, idle$.
Output: $m_1, m_2, \cdots, m_n$. //The final server configuration scheme
$\lambda \leftarrow \lambda_1 + \lambda_2 + \cdots + \lambda_n$; //Initialize the parameters
for i in $range(n)$:
  $m_i \leftarrow \lfloor m_i \rfloor$;
end for
do: //Get the final configuration
  $m_i \leftarrow Sort\_by\_T_i\_desc(m_1, m_2, \cdots, m_n)[0]$; //The value of T_i is calculated by Eq. (A7)
  $m_i \leftarrow m_i + 1$;
  $\bar{C} \leftarrow idle$? the value calculated by Eq. (14) : the value calculated by Eq. (15);
while $\bar{C} < \tilde{C}$ //Increase the number of processors system unit cost exceeds the given cost constraint
$T \leftarrow \frac{\lambda_1}{\lambda} T_1 + \frac{\lambda_2}{\lambda} T_2 + \cdots + \frac{\lambda_n}{\lambda} T_n$; //The value of T_i is calculated by Eq. (A7)
return $m_1, m_2, \cdots, m_n$, and T.

Algorithm 2 describes the details regarding how to find m_i for S_i (i.e., the processor number of S_i) through the bisection algorithm (lines 7–15) according to ϕ. In the algorithm, we obtain lb_i by $lb_i = \lceil \lambda_i \bar{r}/s \rceil$ (line 3), and get ub_i by doubling itself until $\phi(ub_i) > \phi$ (lines 4–6).

The details regarding how to find the server configuration scheme to optimize performance with the given cost constraint are described in Algorithm 3. The process of finding the lower and upper bounds of ϕ is described by lines 3–11. We obtain ub_ϕ by doubling itself until $\overline{C} > \tilde{C}$ (i.e., we obtain the upper bound of ϕ by doubling itself until system unit cost exceeds the given cost constraint). The algorithm searches for ϕ in an interval $[lb_\phi, ub_\phi]$ by using bisection algorithm (lines 12–24), and gets the processor configuration by lines 25–28. It should be noted that the conditional operator "?" in lines 9 and 17 calculates system unit cost based on the value of *idle*, since the energy consumption of processors is different under the two different working models. When *idle* is true, the processor working model supports the idle-speed model, otherwise only the constant-speed model is supported.

Notice that the output of Algorithm 3 is a set of continuous values, that is, a set of decimals. Such a result is not achievable in application environment. Therefore, we provide Algorithm 4 to discretize the configuration further. Basically, the main idea of the algorithm is to round down $m_1, m_2, \ldots, m_n$ (lines 4–6, the cost will be decreased and the response time will be increased), and then increase the number of processors on the edge server with the highest task response time repeatedly until $\overline{C} > \tilde{C}$ (lines 7–11). $Sort\_by\_T_i\_desc(m_1, m_2, \ldots, m_n)$ is a sorting function (line 8) that calculates the task response time of each edge server T_i according to m_i and sorts in descending order.

4.2 | Performance constrained cost optimization

In this section, we solve the problem of performance constrained cost optimization, that is, to minimize

$$\overline{C} = \sum_{i=1}^{n} m_i \cdot \left(\frac{(1+\omega)c_p}{\kappa} + c_e P^* \right) + \lambda \bar{r} s^{\alpha-1} c_e,$$

for the idle-speed model (which represents the unit cost of MEC system when the processors support the idle-speed model), and

$$\overline{C} = \sum_{i=1}^{n} m_i \cdot \left(\frac{(1+\omega)c_p}{\kappa} + (s^\alpha + P^*)\, c_e \right),$$

for the constant-speed model (which represents the unit cost of MEC system when the processors only support the constant-speed model), subject to the constraints $\rho_i < 1$ (since the load of each edge server must not exceed its capacity), for all $1 \le i \le n$, that is, $m_i > \lambda_i \bar{x} = \lambda_i \bar{r}/s$; and $T = \tilde{T}$ (which represents the system performance must be close to the given performance constraint).

In order to facilitate the derivation, we use a new function $J(m_1, m_2, \ldots, m_n)$ to represent $T = \tilde{T}$, namely,

$$J(m_1, m_2, \ldots, m_n) = \frac{\lambda_1}{\lambda} T_1 + \frac{\lambda_2}{\lambda} T_2 + \ldots + \frac{\lambda_n}{\lambda} T_n - \tilde{T}.$$

We also use the method of Lagrange multiplier to solve this optimization problem, we have

$$\nabla \overline{C}(m_1, m_2, \ldots, m_n) = \eta \nabla J(m_1, m_2, \ldots, m_n),$$

that is, n equations

$$\frac{\partial \overline{C}(m_1, m_2, \ldots, m_n)}{\partial m_i} = \eta \frac{\partial J(m_1, m_2, \ldots, m_n)}{\partial m_i},$$

for all $1 \le i \le n$, where η is a Lagrange multiplier. Taking the partial derivative $\partial \overline{C}/\partial m_i$, we obtain

$$\frac{\partial \overline{C}}{\partial m_i} = \frac{(1+\omega)c_p}{\kappa} + c_e P^*, \tag{25}$$

for the idle-speed model, and

$$\frac{\partial \overline{C}}{\partial m_i} = \frac{(1+\omega)c_p}{\kappa} + (s^\alpha + P^*)\, c_e, \tag{26}$$

for the constant-speed model. Taking the partial derivative $\partial J/\partial m_i$, we get

$$\frac{\partial J}{\partial m_i} = \frac{\lambda_i}{\lambda} \cdot \frac{\partial T_i}{\partial m_i} = \frac{\partial T}{\partial m_i}. \tag{27}$$

Based on Equations (20), (25), (26), and (27), we have

$$\frac{\frac{(1+\omega)c_p}{\kappa} + c_e P^*}{\eta} = -\frac{\lambda_i \bar{r}}{\lambda s} \cdot \frac{\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i}\left(\frac{\rho_i+3}{2} - m_i(1-\rho_i)\ln\rho_i\right) + 1}{m_i{}^2(1-\rho_i)^2\left(\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i} + 1\right)^2}, \tag{28}$$

for the idle-speed model, and

$$\frac{\frac{(1+\omega)c_p}{\kappa} + (s^\alpha + P^*)\, c_e}{\eta} = -\frac{\lambda_i \bar{r}}{\lambda s} \cdot \frac{\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i}\left(\frac{\rho_i+3}{2} - m_i(1-\rho_i)\ln\rho_i\right) + 1}{m_i{}^2(1-\rho_i)^2\left(\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i} + 1\right)^2}, \tag{29}$$

for the constant-speed model. Notice that Equations (28) and (29) are the functions of m_i.

Let us rewrite Equation (28) as

$$\eta = \left(\frac{(1+\omega)c_p}{\kappa} + c_e P^*\right)\left(-\frac{\lambda_i \bar{r}}{\lambda s} \cdot \frac{\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i}\left(\frac{\rho_i+3}{2} - m_i(1-\rho_i)\ln\rho_i\right) + 1}{m_i{}^2(1-\rho_i)^2\left(\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i} + 1\right)^2}\right)^{-1}, \tag{30}$$

namely, $\eta_{idle}(m_i)$. Also, Equation (29) becomes

$$\eta = \left(\frac{(1+\omega)c_p}{\kappa} + (s^\alpha + P^*)\, c_e\right)\left(-\frac{\lambda_i \bar{r}}{\lambda s} \cdot \frac{\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i}\left(\frac{\rho_i+3}{2} - m_i(1-\rho_i)\ln\rho_i\right) + 1}{m_i{}^2(1-\rho_i)^2\left(\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i} + 1\right)^2}\right)^{-1}, \tag{31}$$

namely, $\eta_{constant}(m_i)$. According to our analysis in Section 4.1, it is clear that $\eta_{idle}(m_i)$ and $\eta_{constant}(m_i)$ are decreasing functions of m_i.

Similarly, we use the bisection algorithm to search η and m_i within a certain range to obtain a numerical solution. Since the lower bound of m_i is

$$lb_i = \left\lceil \lambda_i \bar{r}/s \right\rceil,$$

we get the upper bound of η is

$$ub_\eta = \begin{cases} \eta_{idle}(lb_{max}), \text{ the idle-speed model;} \\ \eta_{constant}(lb_{max}), \text{ the constant-speed model;} \end{cases}$$

where

$$lb_{max} = \max_{1 \le i \le n} (lb_i).$$

The lower bound of η (namely, lb_η) can be obtained by halving ub_η repeatedly until $T < \tilde{T}$, since T is a descending function of m_i.

Summarizing the above discussion, given $n, \lambda_i, s, \bar{r}, c_p, c_e, \omega, P^*$, and $\tilde{T}$, our algorithms for finding the optimal processor configuration $m_1, m_2, \ldots, m_n$ to minimize the unit cost of the system with the performance constraint $\tilde{T}$ can be described by Algorithms 5–8.

Algorithm 5 describes the steps that find the upper bound of η, that is, ub_η, and provides a basis for the subsequent algorithms. We also use the Boolean *idle* to distinguish the two working modes of the processors (line 11), and calculate the upper bound of η through conditional operator.

Algorithm 6 describes the details regarding how to find m_i for S_i (i.e., the processor number of S_i) through the bisection algorithm (lines 9–19) according to η. In the algorithm, we obtain lb_i by $lb_i = \left\lceil \lambda_i \bar{r}/s \right\rceil$ (line 3), and get ub_i by doubling itself until $\eta_{cal} < \eta$ (lines 4–8) (i.e., find the upper bound of m_i by doubling itself until the calculated value of η is less than the given value).

The details regarding how to find the server configuration scheme to optimize cost with the given performance constraint are described in Algorithm 7. The process of finding the lower and upper bounds of η is described by lines 4–12. We obtain lb_η by halving itself until $T < \tilde{T}$ (i.e., we obtain the lower bound of η by halving itself until system performance meets the performance constraint). The algorithm searches for η in an interval $[lb_\eta, ub_\eta]$ by using bisection algorithm (lines 13–25), and gets the processor configuration by lines 26–29.

Algorithm 8 describes the steps that get the final configuration. The main idea of Algorithm 8 is to round up $m_1, m_2, \ldots, m_n$ (lines 4–6, the cost will be increased and the response time will be decreased), and then decrease the number of processors on the edge server with the lowest task response time repeatedly until $T > \tilde{T}$ (lines 7–11).

Algorithm 5. *Find_ub*$_\eta$

1: **Input:** $n, \lambda_1, \lambda_2, \cdots, \lambda_n, s, \bar{r}, c_p, c_e, \omega, P^*$, *idle*.
2: **Output:** ub_η.
3: $ub_\eta, lb_{max} \leftarrow 0$; $\lambda \leftarrow \lambda_1 + \lambda_2 + \cdots + \lambda_n$; //Initialize the parameters
4: **for** i in *range*(n): //Find the maximum of lb_i, for all $1 \le i \le n$
5: $lb_i \leftarrow \lceil \lambda_i \bar{r}/s \rceil$;
6: **if** $lb_i > lb_{max}$:
7: $lb_{max} \leftarrow lb_i$;
8: **end if**
9: **end for**
10: //The values of $\eta_{idle}(lb_{max})$ and $\eta_{constant}(lb_{max})$ are calculated by Eqs. (30) and (31), respectively, where $m_i \leftarrow lb_{max}$
11: $ub_\eta \leftarrow idle\ ?\ \eta_{idle}(lb_{max})\ :\ \eta_{constant}(lb_{max})$;
12: **return** ub_η. //Return the upper bound of η

Algorithm 6. *Find_m_i_with_η*

1: **Input:** $\lambda_i, s, \bar{r}, c_p, c_e, \omega, P^*, \eta, idle$.
2: **Output:** m_i.
3: $\varepsilon \leftarrow$ a very small quantity; $lb_i, ub_i \leftarrow \lceil \lambda_i \bar{r}/s \rceil$ //Initialize ε, lb_i, and ub_i
4: **do**:
5: $ub_i \leftarrow 2ub_i$;
6: //The values of $\eta_{idle}(ub_i)$ and $\eta_{constant}(ub_i)$ are calculated by Eqs. (30) and (31), respectively, where $m_i \leftarrow ub_i$
7: $\eta_{cal} \leftarrow idle\ ?\ \eta_{idle}(ub_i)\ :\ \eta_{constant}(ub_i)$
8: **while** $\eta_{cal} > \eta$ //Find the upper bound of m_i by doubling itself until the calculated value of η is less than the given value
9: **while** $ub_i - lb_i > \varepsilon$: //Search m_i in $\left[lb_i, ub_i\right]$ through the bisection algorithm
10: $mid \leftarrow (lb_i + ub_i)/2$;
11: //The values of $\eta_{idle}(mid)$ and $\eta_{constant}(mid)$ are calculated by Eqs. (30) and (31), respectively, where $m_i \leftarrow mid$
12: $\eta_{cal} \leftarrow idle\ ?\ \eta_{idle}(mid)\ :\ \eta_{constant}(mid)$
13: **if** $\eta_{cal} > \eta$:
14: $lb_i \leftarrow mid$;
15: **else**:
16: $ub_i \leftarrow mid$;
17: **end if**
18: **end while**
19: $m_i \leftarrow (lb_i + ub_i)/2$;
20: **return** m_i. //Return the processor number of S_i when η is given

Sort_by_T_i $(m_1, m_2, \ldots, m_n)$ is also a sorting function (line 8) that calculates the task response time of each edge server according to m_i and sorts in ascending order. The expression *Sort_by_T_i*$(m_1, m_2, \ldots, m_n)[0]$ is refers to taking the first element after sorting, that is, obtaining the edge server with the best performance per cycle, and reducing its number of processors by one (lines 8–9).

4.3 | Processor selection

In this section, we extend these two problems to the case where the service provider has multiple types of processors provided by different hardware suppliers for selection before constructing the MEC system, that is, each type of processor has different speed, price, and basic power. We give Algorithm 9 to solve the problem.

Let x denote the number of selectable processor types. Then, we use $s_1, s_2, \ldots s_x$; $c_p^1, c_p^2, \ldots, c_p^x$; and $P_1^*, P_2^*, \ldots, P_x^*$; representing the speeds, prices, and basic powers of processors with different types, respectively. The outline of our algorithm can be described as follows:

- Firstly, for each type of processor, we can use the algorithms of performance constrained cost optimization or the algorithms of cost constrained performance optimization to obtain an optical processor configuration scheme;
- Secondly, for the optimal configuration of each type of processor, we can calculate the average task response time and unit cost of the system, correspondingly;
- Finally, we can determine the processor type with the highest performance or the lowest unit cost according to these performance and cost metrics.

Specifically, we use a boolean parameter *op_performance* to control the algorithm outputs, that is, the algorithm outputs the processor type and configuration with the best performance under the given cost constraint when *op_performance* is true, and the algorithm outputs the processor type and configuration with the lowest cost under the given performance constraint when *op_performance* is false. T^i and $\overline{C}^i$ represent the average task response time and the unit cost of the system, respectively, after using the ith type of processor. The expressions *Sort_*$T^i(T^1, T^2, \ldots, T^x)[0]$ and *Sort_*$\overline{C}^i(\overline{C}^1, \overline{C}^2, \ldots, \overline{C}^x)[0]$ refer to sorting the input parameters and returning the index of the first element.

Algorithm 7. *Find_config_with_performance_constraint*

Input: $\tilde{T}, n, \lambda_1, \lambda_2, \cdots, \lambda_n, s, \bar{r}, c_p, c_e, \omega, P^*, idle$. //The *idle* is a Boolean representing the working model of the processor
Output: $m_1, m_2, \cdots, m_n$. //A server configuration scheme with a set of continuous values
//Initialize the parameters
$\varepsilon \leftarrow$ a very small quantity; $ub_\eta, \eta \leftarrow Find\_ub_\eta(n, \lambda_1, \lambda_2, \cdots, \lambda_n, s, \bar{r}, c_p, c_e, \omega, P^*, idle)$; $T \leftarrow$ a very big quantity;
while $T > \tilde{T}$: //Find the lower bound of η by halving itself until system performance meets the performance constraint
  $\eta \leftarrow 2\eta$; //Since $\eta < 0$, the halving operation is actually a multiplication by 2.
  for *i* in *range*(*n*):
    $m_i \leftarrow Find\_m_i\_with\_\eta(\lambda_i, s, \bar{r}, c_p, c_e, \omega, P^*, \eta, idle)$;
  end for
  $T \leftarrow \frac{\lambda_1}{\lambda} T_1 + \frac{\lambda_2}{\lambda} T_2 + \cdots + \frac{\lambda_n}{\lambda} T_n$; //The value of T_i is calculated by Eq. (A7)
end while
$lb_\eta \leftarrow \eta$;
while $ub_\eta - lb_\eta > \varepsilon$: //Search η in $[lb_\eta, ub_\eta]$ through the bisection algorithm
  $mid \leftarrow \left(lb_\eta + ub_\eta\right)/2$;
  for *i* in *range*(*n*): //Obtain the server configuration scheme according to the new value of η (i.e., *mid*)
    $m_i \leftarrow Find\_m_i\_with\_\eta(\lambda_i, s, \bar{r}, c_p, c_e, \omega, P^*, mid, idle)$;
  end for
  $T \leftarrow \frac{\lambda_1}{\lambda} T_1 + \frac{\lambda_2}{\lambda} T_2 + \cdots + \frac{\lambda_n}{\lambda} T_n$; //The value of T_i is calculated by Eq. (A7)
  if $T > \tilde{T}$:
    $ub_\eta \leftarrow mid$;
  else:
    $lb_\eta \leftarrow mid$;
  end if
end while
$\eta \leftarrow \left(lb_\eta + ub_\eta\right)/2$;
for *i* in *range*(*n*): //Obtain the server configuration scheme according to the determined η
  $m_i \leftarrow Find\_m_i\_with\_\eta(\lambda_i, s, \bar{r}, c_p, c_e, \omega, P^*, \eta, idle)$;
end for
return $m_1, m_2, \cdots, m_n$.

Algorithm 8. *Discretize_config_and_Calculate_*$\overline{C}$

Input: $m_1, m_2, \cdots, m_n, \tilde{T}, n, \lambda_1, \lambda_2, \cdots, \lambda_n, s, \bar{r}, c_p, c_e, \omega, P^*, idle$.
Output: $m_1, m_2, \cdots, m_n$. //The final server configuration scheme
$\lambda \leftarrow \lambda_1 + \lambda_2 + \cdots + \lambda_n$; //Initialize the parameters
for *i* in *range*(*n*):
  $m_i \leftarrow \lceil m_i \rceil$;
end for
do: //Get the final configuration
  $m_i \leftarrow Sort\_by\_T_i(m_1, m_2, \cdots, m_n)[0]$; //The value of T_i is calculated by Eq. (A7)
  $m_i \leftarrow m_i - 1$;
  $T \leftarrow \frac{\lambda_1}{\lambda} T_1 + \frac{\lambda_2}{\lambda} T_2 + \cdots + \frac{\lambda_n}{\lambda} T_n$; //The value of T_i is calculated by Eq. (A7)
while $T < \tilde{T}$
$\bar{C} \leftarrow idle$? the value calculated by Eq. (14) : the value calculated by Eq. (15);
return $m_1, m_2, \cdots, m_n$, and $\bar{C}$.

Algorithm 9. *Processor_selection*

Input:
$\tilde{C}, \tilde{T}, n, \lambda_1, \lambda_2, \cdots, \lambda_n, \bar{r}, c_e, \omega, idle, op\_performance, x, s_1, s_2, \cdots s_x, c_p^1, c_p^2, \cdots, c_p^x, P_1^*, P_2^*, \cdots, P_x^*$;
Output:
$index$; //The selected processor type ($1 \leq index \leq x$)
$m_1, m_2, \cdots, m_n$. //The server configuration scheme with the selected processor
$\lambda \leftarrow \lambda_1 + \lambda_2 + \cdots + \lambda_n$; //Initialize the parameters
for i in $range(x)$:
  if $op\_performance$:
    $m_1, m_2, \cdots, m_n \leftarrow$ Algorithm 3; //With s_i, c_p^i, P_i^*
    $m_1, m_2, \cdots, m_n \leftarrow$ Algorithm 4; //With s_i, c_p^i, P_i^*
    $T^i \leftarrow \frac{\lambda_1}{\lambda} T_1 + \frac{\lambda_2}{\lambda} T_2 + \cdots + \frac{\lambda_n}{\lambda} T_n$; //The value of T_i is calculated by Eq. (A7)
  else:
    $m_1, m_2, \cdots, m_n \leftarrow$ Algorithm 7; //With s_i, c_p^i, P_i^*
    $m_1, m_2, \cdots, m_n \leftarrow$ Algorithm 8; //With s_i, c_p^i, P_i^*
    $\bar{C}^i \leftarrow idle$? the value calculated by Eq. (14) : the value calculated by Eq. (15);
  end if
end for
$index \leftarrow op\_performance$? $Sort\_T^i(T^1, T^2, \cdots, T^x)[0]$: $Sort\_\bar{C}^i(\bar{C}^1, \bar{C}^2, \cdots, \bar{C}^x)[0]$;
return $index, m_1, m_2, \cdots, m_n$.

5 | NUMERICAL EXAMPLES

In this section, we illustrate some numerical examples to show the effectiveness of algorithms proposed above. Notice that all the parameters used in the examples are for illustrative purposes only, and they can be changed to any other actual values. Since our parameters have specific physical meanings and can be obtained through statistical methods. Therefore, our strategy has guiding significance for actual implementation.

5.1 | The parameters

Let us consider an example scenario from the perspective of a service provider that we need to deploy edge servers in 20 selected mobile base stations to build an MEC system. We mainly consider two critical constraints, that is, the unit cost of the system needs to be close to 0.004 CNY per second, and the average task response time of the system needs to be close to 1.01 s. In addition, an important factor we consider is that there may be multiple types of processors provided by different hardware suppliers to choose from before building up an MEC system. Our parameter settings are shown in Tables 3 and 4.

TABLE 3 The parameters of an example scenario

Parameters	Explanation
$\tilde{C} = 0.004$	The unit cost of the system needs to be close to 0.004 CNY per second
$\tilde{T} = 1.01$	The average task response time of the system needs to be close to 1.01 s
$n = 20$	We have 20 edge servers that need to be deployed in the selected base stations
$\bar{r} = 2$	The average task size is two billion instructions.
$c_e = 0.8/1000/3600$	The price of electricity per watt per second (measured in CNY)
$\omega = 0.3$	The ratio coefficient of maintenance and utility costs is 0.3
$\lambda_1, \lambda_2, \ldots, \lambda_n$ are a group of random numbers	The arrival rate of offloadable tasks to edge servers is a set of random numbers from a normal distribution with mean 15 and standard deviation 6

TABLE 4 The parameters of three selectable processors

Processor type	Speed	Price	Basic power	Working model
Processor I	2 BIPS	400 CNY	2 W	The processor only supports the constant-speed model
Processor II	3 BIPS	700 CNY	3 W	The processor supports the idle-speed model
Processor III	3.9 BIPS	900 CNY	6 W	The processor supports the idle-speed model

5.2 | Examples for minimizing average response time with unit cost constraint

In this section, we respectively show numerical data of server configuration schemes based on the above parameters for minimizing average response time with unit cost constraint. In the three numerical examples, the task arrival rates of edge servers are the same.

Tables 5, 6, and 7 respectively illustrate the server configuration scheme of 20 edge servers with Processor I, II, and III. Through Algorithms 1–3, we obtain the optimal configuration schemes for 20 edge servers that make the system have the best performance when the unit cost of the system is close to 0.004 CNY per second. However, the number of processors can only be a positive integer, such configuration schemes cannot be implemented in an actual environment. Through Algorithm 4, we obtain a set of positive integer results for the processor configuration of edge servers at the cost of a slight decrease in performance, that is, the average response time changes from 1.000140950 to 1.000148059 s when Processor I is selected for configuration, from 0.677000832 to 0.677493237 s when Processor II is selected for configuration, and from 0.900150395 to 0.992104602 s when Processor III is selected for configuration. It is reasonable because we considered the

TABLE 5 Numerical data for minimizing average response time with unit cost constraint (configure with Processor I)

i	λ_i	m_i (Algorithm 3)	m_i (round down $m_1, m_2, \ldots, m_n$)	m_i (Algorithm 4)
1	12.49945292	24.50535645	24	25
2	14.66239904	27.56910580	27	27
3	2.18282343	7.81499493	7	9
4	24.84162485	41.28288538	41	41
5	4.23938649	11.69081052	11	12
6	9.94951581	20.78450309	20	21
7	18.01728850	32.19686218	32	32
8	7.52827148	17.09917763	17	18
9	8.65228669	18.83243993	18	19
10	9.54595431	20.18215459	20	20
11	18.30872427	32.59296512	32	32
12	28.75324808	46.34957365	46	46
13	15.24923636	28.38850033	28	28
14	8.29244733	18.28220921	18	19
15	18.23434992	32.49196032	32	32
16	11.42304180	22.95105727	22	23
17	14.88521702	27.88077014	27	28
18	22.05000732	37.60963971	37	37
19	10.51277430	21.61846768	21	22
20	15.05415151	28.11661704	28	28
The unit cost $\overline{C}$		0.004	0.003921042	0.004005946
The average task response time T		1.000140950	1.000220429	1.000148059

TABLE 6 Numerical data for minimizing average response time with unit cost constraint (configure with Processor II)

i	λ_i	m_i (Algorithm 3)	m_i (round down $m_1, m_2, \ldots, m_n$)	m_i (Algorithm 4)
1	12.49945292	13.19975695	13	13
2	14.66239904	15.02722553	15	15
3	2.18282343	3.60478329	3	5
4	24.84162485	23.32746889	23	23
5	4.23938649	5.74844003	5	7
6	9.94951581	10.99902184	10	11
7	18.01728850	17.80884326	17	17
8	7.52827148	8.84476715	8	9
9	8.65228669	9.85433876	9	10
10	9.54595431	10.64500661	10	11
11	18.30872427	18.04796402	18	18
12	28.75324808	26.43037190	26	26
13	15.24923636	15.51800460	15	15
14	8.29244733	9.53310694	9	10
15	18.23434992	17.98697479	17	18
16	11.42304180	12.27771779	12	12
17	14.88521702	15.21380327	15	15
18	22.05000732	21.08855491	21	21
19	10.51277430	11.49026596	11	12
20	15.05415151	15.35507054	15	15
The unit cost $\overline{C}$		0.004	0.003897131	0.004010270
The average task response time T		0.677000832	0.682492201	0.677493237

TABLE 7 Numerical data for minimizing average response time with unit cost constraint (configure with Processor III)

i	λ_i	m_i (Algorithm 3)	m_i (round down $m_1, m_2, \ldots, m_n$)	m_i (Algorithm 4)
1	12.49945292	7.09249965	7	7
2	14.66239904	8.25845826	8	8
3	2.18282343	2.00000000	2	2
4	24.84162485	13.70178861	13	14
5	4.23938649	3.00000000	3	3
6	9.94951581	6.00000000	6	6
7	18.01728850	10.05921700	10	10
8	7.52827148	4.39020035	4	5
9	8.65228669	5.00481823	5	5
10	9.54595431	5.49174025	5	6
11	18.30872427	10.21527914	10	10
12	28.75324808	15.78081950	15	16
13	15.24923636	8.57406358	8	9
14	8.29244733	5.00000000	5	5
15	18.23434992	10.17545715	10	10
16	11.42304180	6.51041473	6	7
17	14.88521702	8.37832532	8	8
18	22.05000732	12.21445238	12	12
19	10.51277430	6.01704917	6	6
20	15.05415151	8.46917735	8	8
The unit cost $\overline{C}$		0.004	0.003926927	0.004009128
The average task response time T		0.900150395	1.753751666	0.992104602

performance balance of all edge servers, that is, the computing resources of the edge servers with lower performance are increased.

Through the data in Tables 5–7, we can observe that when the unit cost constraint is $\tilde{C} = 0.004$, Processor II is the best choice, which makes the system have the highest performance, that is, the average response time of all tasks is 0.677493237.

5.3 | Examples for minimizing unit cost with average response time constraint

In this section, we respectively show numerical data of server configuration schemes for minimizing unit cost with average response time constraint. In the following numerical examples, the task arrival rates of edge servers are the same as the previous section.

Tables 8, 9, and 10 respectively illustrate the server configuration scheme of 20 edge servers with Processor I, II, and III. Through Algorithms 5–7, we obtain the optimal configuration schemes for 20 edge servers that make the service provider have the lowest investment cost when the average task response time of the system is close to 1.01 s. Similarly, such configuration schemes cannot be implemented in an actual environment. Through Algorithm 8, we obtain a set of positive integer results for the processor configuration of edge servers at the cost of a slight increase in investment cost, that is, the unit cost of the system changes from 0.003096559 CNY to 0.003102872 CNY when Processor I is selected for configuration, from 0.00318866 CNY to 0.003208016 CNY when Processor II is selected for configuration, and from 0.003971932 CNY to 0.003995427 CNY when Processor III is selected for configuration. We also considered the performance balance of all

TABLE 8 Numerical data for minimizing unit cost with average response time constraint (configure with Processor I)

i	λ_i	m_i (Algorithm 7)	m_i (round up $m_1, m_2, \ldots, m_n$)	m_i (Algorithm 8)
1	12.49945292	18.72342979	19	19
2	14.66239904	21.38104858	22	21
3	2.18282343	4.92441938	5	5
4	24.84162485	33.50370378	34	33
5	4.23938649	7.96991813	8	8
6	9.94951581	15.53103398	16	16
7	18.01728850	25.43542011	26	25
8	7.52827148	12.41710829	13	13
9	8.65228669	13.87484029	14	14
10	9.54595431	15.01847436	16	15
11	18.30872427	25.78439593	26	26
12	28.75324808	38.05102575	39	37
13	15.24923636	22.09564394	23	22
14	8.29244733	13.41068489	14	14
15	18.23434992	25.69538139	26	26
16	11.42304180	17.38472873	18	18
17	14.88521702	21.65267309	22	22
18	22.05000732	30.22704434	31	30
19	10.51277430	16.24276115	17	16
20	15.05415151	21.85836685	22	22
The unit cost $\overline{C}$		0.003096559	0.003172339	0.003102872
The average task response time T		1.01	1.007343592	1.010059071

TABLE 9 Numerical data for minimizing unit cost with average response time constraint (configure with Processor II)

i	λ_i	m_i (Algorithm 7)	m_i (round up $m_1, m_2, \ldots, m_n$)	m_i (Algorithm 8)
1	12.49945292	9.28596191	10	9
2	14.66239904	10.80713281	11	11
3	2.18282343	2.00000000	2	2
4	24.84162485	17.90479710	18	18
5	4.23938649	3.38113867	4	4
6	9.94951581	7.48321097	8	8
7	18.01728850	13.15579450	14	13
8	7.52827148	6.00000000	6	6
9	8.65228669	6.56100564	7	7
10	9.54595431	7.19674087	8	7
11	18.30872427	13.35930687	14	13
12	28.75324808	20.61452130	21	21
13	15.24923636	11.21882124	12	11
14	8.29244733	6.30444399	7	7
15	18.23434992	13.30737758	14	13
16	11.42304180	8.52637651	9	9
17	14.88521702	10.96349537	11	11
18	22.05000732	15.96593261	16	16
19	10.51277430	8.00000000	8	8
20	15.05415151	11.08200627	12	11
The unit cost $\overline{C}$		0.00318866	0.003280013	0.003208016
The average task response time T		1.01	0.876112985	1.013917482

edge servers in the algorithm, that is, we reduced the number of processors on the edge servers with higher performance to save cost.

We can observe from Tables 8–10 that when the performance constraint is $\tilde{T} = 1.01$, Processor I is the best choice, which makes the system have the lowest cost, that is, the unit cost of the system is 0.003102872.

Observe the numerical data in Sections 5.2 and 5.3, we also get two significant observations as follows:

- Comparing the data in Tables 6 and 7, we find that the average response time of the system has not been reduced by choosing a faster Processor III. In Table 10, we find that the average task response time of the system significantly deviates from the performance constraint when the faster and more expensive Processor III are selected. This is because increasing or decreasing the number of processors by only one can have a significant impact on overall performance in some areas with higher computation demands. By analyzing the data in Tables 8–10, we find that the system unit cost has increased due to the selection of Processor II and Processor III which have a higher price and basic power than Processor I. In some cases, a slower, cheaper processor will yield better configuration because its impact on overall performance and cost is limited. This also means that the speed of the processor is not the higher the better.
- In this article, we assume that the sites of edge servers has been determined, which means that base stations with low task arrival rate also need to deploy at least one edge server. It can be observed that it is inappropriate to deploy edge servers in base stations with low task arrival rate. How to select the reasonable sites for MEC servers based on computation demand statistics of different geographies is also an important issue, and we will further solve this problem in future research.

TABLE 10 Numerical data for minimizing unit cost with average response time constraint (configure with Processor III)

i	λ_i	m_i (Algorithm 7)	m_i (round up $m_1, m_2, \ldots, m_n$)	m_i (Algorithm 8)
1	12.49945292	7.00000000	7	7
2	14.66239904	8.10973137	9	8
3	2.18282343	2.00000000	2	2
4	24.84162485	13.50819099	14	14
5	4.23938649	3.00000000	3	3
6	9.94951581	6.00000000	6	6
7	18.01728850	10.00000000	10	10
8	7.52827148	4.28362732	5	4
9	8.65228669	5.00000000	5	5
10	9.54595431	5.37173589	6	6
11	18.30872427	10.04908203	11	10
12	28.75324808	15.57253237	16	16
13	15.24923636	8.42238927	9	9
14	8.29244733	5.00000000	5	5
15	18.23434992	10.00959800	11	10
16	11.42304180	6.37914190	7	7
17	14.88521702	8.22847249	9	8
18	22.05000732	12.03205956	13	12
19	10.51277430	6.00000000	6	6
20	15.05415151	8.31847647	9	8
The unit cost $\overline{C}$		0.003971932	0.004091328	0.003995427
The average task response time T		1.01	0.758340406	1.072924688

6 | CONCLUSION

In this article, we have mentioned the significance of server configuration optimization in MEC. We have reviewed the existing research and summarized the unique features of our research. From the perspective of cost-performance tradeoff, we investigated the problem of server configuration optimization based on a given computation demand statistics. Two optimization problems, including *cost constrained performance optimization* and *performance constrained cost optimization*, are formulated by our models and solved by a series of fast numerical algorithms (since our algorithms only need to be used once before system deployment, and can be reused when the environment changes significantly, the cost/energy of our algorithms are not discussed in this article, and we think it can be ignored to certain extent). By solving the *cost constrained performance optimization* problem, MEC service providers can obtain a processor configuration of edge servers that can deliver the highest quality of service with a given cost constraint. Service providers of MEC can obtain a processor configuration of edge servers that can minimize the investment cost (including hardware cost and energy consumption cost) with a service-quality guarantee by solving the *performance constrained cost optimization* problem. Finally, we analyzed and verified our approach based on numerical experiments. Our research results in this article provide theoretical and practical insights into the configuration of computing resources on edge servers in MEC. In future work we will focus on the problem on how to select the reasonable sites for MEC servers based on computation demand statistics.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to the anonymous reviewers whose constructive comments are very important to improve this manuscript. This work was supported in part by the National Natural Science Foundation of China under grant no. 61876061, and the Yunnan Applied Basic Research Projects under grant no. 202001BB050034.

ORCID

Zhenli He https://orcid.org/0000-0002-7986-2222
Keqin Li https://orcid.org/0000-0001-5224-4048

REFERENCES

1. Taleb T, Dutta S, Ksentini A, Iqbal M, Flinck H. Mobile edge computing potential in making cities smarter. *IEEE Commun Mag*. 2017;55(3):38-43.
2. Xu Z, Liang W, Xu W, Jia M, Guo S. Efficient algorithms for capacitated cloudlet placements. *IEEE Trans Parall Distrib Syst*. 2015;27(10):2866-2880.
3. Jia M, Cao J, Liang W. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. *IEEE Trans Cloud Comput*. 2015;5(4):725-737.
4. Alelaiwi A. An efficient method of computation offloading in an edge cloud platform. *J Parall Distrib Comput*. 2019;127:58-64.
5. Tao Z, Xia Q, Hao Z, et al. A survey of virtual machine management in edge computing. *Proc IEEE*. 2019;107(8):1482-1499.
6. Lin L, Liao X, Jin H, Li P. Computation offloading toward edge computing. *Proc IEEE*. 2019;107(8):1584-1607.
7. Cisco Annual Internet Report White Paper; 2020. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectiv%es/annual-internet-report/white-paper-c11-741490.html/Updated. Accessed March 9, 2020.
8. Ma X, Wang S, Zhang S, Yang P, Lin C, Shen XS. Cost-Efficient Resource Provisioning for Dynamic Requests in Cloud Assisted Mobile Edge Computing. *IEEE Transactions on Cloud Computing*. 2019;1(1):1–1. http://dx.doi.org/10.1109/tcc.2019.2903240.
9. Li H, Shou G, Hu Y, Guo Z. Mobile edge computing: progress and challenges. Paper presented at: Proceedings of the 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), Oxford, UK: IEEE; 2016:83-84. https://doi.org/10.1109/MobileCloud.2016.16.
10. Dolui K, Datta SK. Comparison of edge computing implementations: fog computing, cloudlet and mobile edge computing. Paper presented at:2017 Global Internet of Things Summit (GIoTS) 2017. Geneva, Switzerland: IEEE; 2017:1-6. https://doi.org/10.1109/GIOTS.2017.8016213
11. Mach P, Becvar Z. Mobile edge computing: a survey on architecture and computation offloading. *IEEE Commun Surv Tutor*. 2017;19(3):1628-1656.
12. Abbas N, Zhang Y, Taherkordi A, Skeie T. Mobile edge computing: a survey. *IEEE IoT J*. 2017;5(1):450-465.
13. Mao Y, You C, Zhang J, Huang K, Letaief KB. A survey on mobile edge computing: the communication perspective. *IEEE Commun Surv Tutor*. 2017;19(4):2322-2358.
14. Ahmed E, Rehmani MH. Mobile Edge Computing: Opportunities, solutions, and challenges. *Future Generation Computer Systems*. 2017;70:59–63. http://dx.doi.org/10.1016/j.future.2016.09.015.
15. Dayarathna M, Wen Y, Fan R. Data center energy consumption modeling: a survey. *IEEE Commun Surv Tutor*. 2015;18(1):732-794.
16. Li K. How to stabilize a competitive mobile edge computing environment: a game theoretic approach. *IEEE Access*. 2019;7:69960-69985.
17. Shi W, Pallis G, Xu Z. Edge computing [Scanning the issue]. *Proc IEEE*. 2019;107(8):1474-1481.
18. Rahimi MR, Ren J, Liu CH, Vasilakos AV, Venkatasubramanian N. Mobile cloud computing: a survey, state of art and future directions. *Mob Netw Appl*. 2014;19(2):133-143.
19. Lyu X, Tian H, Jiang L, et al. Selective offloading in mobile edge computing for the green internet of things. *IEEE Netw*. 2018;32(1):54-60.
20. Tham CK, Chattopadhyay R. A load balancing scheme for sensing and analytics on a mobile edge computing network. Paper presented at: Proceedings of the 2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), Macau, China: IEEE; 2017:1-9. https://doi.org/10.1109/WoWMoM.2017.7974307.
21. Tao X, Ota K, Dong M, Qi H, Li K. Performance guaranteed computation offloading for mobile-edge cloud computing. *IEEE Wirel Commun Lett*. 2017;6(6):774-777.
22. Chen X, Jiao L, Li W, Fu X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans Netw*. 2015;24(5):2795-2808.
23. Huang D, Wang P, Niyato D. A dynamic offloading algorithm for mobile computing. *IEEE Trans Wirel Commun*. 2012;11(6):1991-1995.
24. Liu J, Mao Y, Zhang J, Letaief KB. Delay-optimal computation task scheduling for mobile-edge computing systems. Paper presented at: Proceedings of the 2016 IEEE International Symposium on Information Theory (ISIT), Barcelona, Spain: IEEE; 2016:1451-1455. https://doi.org/10.1109/ISIT.2016.7541539.
25. Xu X, Cao H, Geng Q, Liu X, Dai F, Wang C. Dynamic resource provisioning for workflow scheduling under uncertainty in edge computing environment. *Concurrency and Computation: Practice and Experience*. 2020;e5674. http://dx.doi.org/10.1002/cpe.5674.
26. Li K. Computation Offloading Strategy Optimization with Multiple Heterogeneous Servers in Mobile Edge Computing. *IEEE Transactions on Sustainable Computing*. 2019;1–1. http://dx.doi.org/10.1109/tsusc.2019.2904680.
27. Li K. A Game Theoretic Approach to Computation Offloading Strategy Optimization for Non-cooperative Users in Mobile Edge Computing. *IEEE Transactions on Sustainable Computing*. 2018;1–1. http://dx.doi.org/10.1109/tsusc.2018.2868655.
28. Mao Y, Zhang J, Letaief KB. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE J Select Areas Commun*. 2016;34(12):3590-3605.
29. Dinh TQ, Tang J, La QD, Quek TQS. Offloading in mobile edge computing: task allocation and computational frequency scaling. *IEEE Trans Commun*. 2017;65(8):3571-3584.

30. Zhang G, Zhang W, Cao Y, Li D, Wang L. Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices. *IEEE Trans Ind Inform*. 2018;14(10):4642-4655.
31. Zhang J, Hu X, Ning Z, et al. Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks. *IEEE IoT J*. 2017;5(4):2633-2645.
32. Xu J, Chen L, Zhou P. Joint service caching and task offloading for mobile edge computing in dense networks. Paper presented at: Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications, Honolulu, HI: IEEE; 2018:207-215. https://doi.org/10.1109/INFOCOM.2018.8485977.
33. Tran TX, Pompili D. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Trans Veh Technol*. 2018;68(1):856-868.
34. Ma X, Zhang S, Yang P, Zhang N, Lin C, Shen X. Cost-efficient resource provisioning in cloud assisted mobile edge computing. Paper presented at: Proceedings of the GLOBECOM 2017-2017 IEEE Global Communications Conference 2017, Singapore, Singapore: IEEE; 2017:1-6. https://doi.org/10.1109/GLOCOM.2017.8254704.
35. Mao Y, Zhang J, Song SH, Letaief KB. Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. *IEEE Trans Wirel Commun*. 2017;16(9):5994-6009.
36. Zhou Z, Abawajy J, Chowdhury M, et al. Minimizing SLA violation and power consumption in cloud data centers using adaptive energy-aware algorithms. *Futur Gener Comput Syst*. 2018;86:836-850.
37. Paščinski U, Trnkoczy J, Stankovski V, Cigale M, Gec S. QoS-aware orchestration of network intensive software utilities within software defined data centres. *J Grid Comput*. 2018;16(1):85-112.
38. Kleinrock Leonard. *Queueing Systems. Volume I: Theory*. New York: wiley; 1976.
39. Chandrakasan AP, Sheng S, Brodersen RW. Low-power CMOS digital design. *IEICE Trans Electron*. 1992;75(4):371-382.
40. Zhai B, Blaauw D, Sylvester D, Flautner K. Theoretical and practical limits of dynamic voltage scaling. Paper presented at: Proceedings of the DAC '04Association for Computing Machinery; 2004:868-873; New York, NY.
41. Cao J, Li K, Stojmenovic I. Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers. *IEEE Trans Comput*. 2013;63(1):45-58.
42. Li K. Optimal partitioning of a multicore server processor. *J Supercomput*. 2015;71(10):3744-3769.
43. Li K. Optimal configuration of a multicore server processor for managing the power and performance tradeoff. *J Supercomput*. 2012;61(1):189-214.
44. Romik D. Stirling's approximation for n!: the ultimate short proof? *Am Math Mon*. 2000;107(6):556-557.

How to cite this article: He Z, Li K, Li K, Zhou W. Server configuration optimization in mobile edge computing: A cost-performance tradeoff perspective. *Softw Pract Exper*. 2021;51:1868–1895. https://doi.org/10.1002/spe.2951

APPENDIX. DETAILED DERIVATION PROCESS

Let us rewrite Equation (6) as

$$T_i = \overline{x}\left(1 + \frac{P_{q,i}}{m_i(1-\rho_i)}\right) = \frac{\overline{r}}{s}\left(1 + p_{i,0}\frac{{m_i}^{m_i-1}{\rho_i}^{m_i}}{m_i!(1-\rho_i)^2}\right) = \frac{\overline{r}}{s}\left(1 + \frac{p_{i,m_i}}{m_i(1-\rho_i)^2}\right). \tag{A1}$$

Notice that Equation (A1) contains the factorial of m_i and thus its partial derivative cannot be obtained directly. In order to take the partial derivative $\partial T_i/\partial m_i$, we use the Stirling's approximation of $m_i!$,[44] that is,

$$m_i! \approx \sqrt{2\pi m_i}\left(\frac{m_i}{e}\right)^{m_i}, \tag{A2}$$

and the following closed-form approximation

$$\sum_{k=0}^{m_i-1}\frac{(m_i\rho_i)^k}{k!} \approx e^{m_i\rho_i}, \tag{A3}$$

to obtain a closed-form approximation of $p_{i,0}$ as

$$p_{i,0} \approx \left(e^{m_i \rho_i} + \frac{(e\rho_i)^{m_i}}{\sqrt{2\pi m_i}} \cdot \frac{1}{1-\rho_i} \right)^{-1}, \tag{A4}$$

and a closed-form approximation of p_{i,m_i} as

$$p_{i,m_i} \approx \frac{\frac{(e\rho_i)^{m_i}}{\sqrt{2\pi m_i}}}{e^{m_i \rho_i} + \frac{(e\rho_i)^{m_i}}{\sqrt{2\pi m_i}} \cdot \frac{1}{1-\rho_i}}, \tag{A5}$$

namely,

$$p_{i,m_i} \approx \frac{1-\rho_i}{\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i} + 1}. \tag{A6}$$

Hence, the average task response time of S_i can be approximated as

$$T_i = \frac{\overline{r}}{s}\left(1 + \frac{1}{m_i(1-\rho_i)\left(\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i} + 1\right)}\right). \tag{A7}$$

Let us rewrite T_i as

$$T_i = \frac{\overline{r}}{s}(1 + F_i), \tag{A8}$$

where

$$F_i = \frac{1}{m_i(1-\rho_i)\left(\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i} + 1\right)}. \tag{A9}$$

It is clear that

$$\frac{\partial T_i}{\partial m_i} = \frac{\overline{r}}{s} \cdot \frac{\partial F_i}{\partial m_i}. \tag{A10}$$

In order to take the partial derivative $\partial F_i / \partial m_i$, let us rewrite F_i as

$$F_i = \frac{1}{m_i(1-\rho_i)\left(\sqrt{2\pi m_i}(1-\rho_i)H_i + 1\right)}, \tag{A11}$$

where

$$H_i = (e^{\rho_i}/e\rho_i)^{m_i}, \tag{A12}$$

that is,

$$F_i = \frac{1}{\sqrt{2\pi} m_i^{3/2}(1-\rho_i)^2 H_i + m_i(1-\rho_i)}. \tag{A13}$$

Since

$$\ln H_i = m_i \ln(e^{\rho_i}/e\rho_i) = m_i(\rho_i - \ln \rho_i - 1),$$

we get

$$\frac{1}{H_i}\cdot\frac{\partial H_i}{\partial m_i}=(\rho_i-\ln\rho_i-1)+m_i\left(1-\frac{1}{\rho_i}\right)\frac{\partial\rho_i}{\partial m_i}. \tag{A14}$$

Notice that

$$\frac{\partial\rho_i}{\partial m_i}=-\frac{\lambda_i\bar{x}}{{m_i}^2}=-\frac{\rho_i}{m_i}, \tag{A15}$$

then, we have

$$\frac{\partial H_i}{\partial m_i}=-H_i\ln\rho_i. \tag{A16}$$

Therefore, we get

$$\begin{aligned}\frac{\partial F_i}{\partial m_i}&=-{F_i}^2\left(\frac{\partial\left(\sqrt{2\pi}{m_i}^{\frac{3}{2}}(1-\rho_i)^2H_i+m_i(1-\rho_i)\right)}{\partial m_i}\right)\\&=-{F_i}^2\left(\sqrt{2\pi m_i}(1-\rho_i)H_i\left(\frac{\rho_i+3}{2}-m_i(1-\rho_i)\ln\rho_i\right)+1\right).\end{aligned} \tag{A17}$$

Summarizing the above discussion, based on Equations (A10) and (A17), we get

$$\frac{\partial T_i}{\partial m_i}=-\frac{\bar{r}}{s}{F_i}^2\left(\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{\rho_i+3}{2}-m_i(1-\rho_i)\ln\rho_i\right)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i}+1\right). \tag{A18}$$