A Cloud–MEC Collaborative Task Offloading Scheme With Service Orchestration

Mingfeng Huang, Wei Liu, Tian Wang, Anfeng Liu, and Shigeng Zhang, *Member, IEEE*

***Abstract*—Billions of devices are connected to the Internet of Things (IoT). These devices generate a large volume of data, which poses an enormous burden on conventional networking infrastructures. As an effective computing model, edge computing is collaborative with cloud computing by moving part intensive computation and storage resources to edge devices, thus optimizing the network latency and energy consumption. Meanwhile, the software-defined networks (SDNs) technology is promising in improving the quality of service (QoS) for complex IoT-driven applications. However, building SDN-based computing platform faces great challenges, making it difficult for the current computing models to meet the low-latency, high-complexity, and high-reliability requirements of emerging applications. Therefore, a cloud-mobile edge computing (MEC) collaborative task offloading scheme with service orchestration (CTOSO) is proposed in this article. First, the CTOSO scheme models the computational consumption, communication consumption, and latency of task offloading and implements differentiated offloading decisions for tasks with different resource demand and delay sensitivity. What is more, the CTOSO scheme introduces orchestrating data as services (ODaS) mechanism based on the SDN technology. The collected metadata are orchestrated as high-quality services by MEC servers, which greatly reduces the network load caused by uploading resources to the cloud on the one hand, and on the other hand, the data processing is completed at the edge layer as much as possible, which achieves the load balancing and also reduces the risk of data leakage. The experimental results demonstrate that compared to the random decision-based task offloading scheme and the maximum cache-based task offloading scheme, the CTOSO scheme reduces delay by approximately 73.82%–74.34% and energy consumption by 10.71%–13.73%.**

***Index Terms*—Delay, energy consumption, Internet of Things (IoT), service orchestration, task offloading decision.**

I. Introduction

RECENTLY, with the rapid development of wireless communication technology and Internet of Things (IoT) [1]–[3], billions of devices have been connected to the IoT infrastructure [4]–[7]. However, the computing and storage capacity of these mobile devices is limited, so they need to be combined with devices with more powerful computing and storage capabilities to perform tasks in a collaborative manner. Parallel to this trend, the emergence of low-latency applications, such as face/fingerprint/iris recognition, augmented/virtual reality, natural language processing, interactive gaming, and privacy-preserving data in these applications [8], has brought great challenges to the existing computing models [9]–[12].

Then, cloud computing is proposed to address limitations (e.g., storage and computing resources) in the existing IoT networks [13]. In this computing model, a cloud server with powerful computing and storage capability is deployed on the core network layer (e.g., the cloud), and the workload of mobile devices can be reduced by offloading the data starvation or computation intensive tasks to the cloud server. However, since the cloud server is located on the core network layer, it is far away from mobile terminals and users, so when offloading tasks to the cloud server, the data resources must flow through the entire access network and core network, and through multiple base stations and other key equipment or transport equipment [9]. Similarly, the task computational results can only be returned to mobile devices after going through long-distance routing. Therefore, it causes a serious delay and additional transmission consumption, and easily leads to data leakage and other problems. So, this cloud-centric IoT solution is not suitable for applications with low-latency, high-reliability, and high-security requirements.

The European Telecommunications Standardization Institute (ETSI) proposed mobile edge computing (MEC) in 2014 to address the problems that exist in the cloud computing model [9]. In the MEC model, the computation-intensive or large data amount tasks are offloaded to MEC servers. Since MEC servers are located at the network edge, so compared with offloading tasks to the cloud, transferring task resources to MEC servers does not need to go through the core network, thus satisfying the low-latency requirement of applications. In addition, returning task results directly from the edge layer can protect and isolate sensitive data and reduce the risk of privacy leakage. Therefore, this MEC computing model avoids the long delay, serious energy consumption, data leakage, etc., caused by tasks migration in the cloud computing model and has developed into an important computing paradigm which attracts numerous researchers.

Manuscript received August 29, 2019; revised October 16, 2019; accepted November 1, 2019. Date of publication November 11, 2019; date of current version July 10, 2020. This work was supported by the National Natural Science Foundation of China under Grant 61772554 and Grant 61772559. *(Corresponding author: Shigeng Zhang.)*

M. Huang and A. Liu are with the School of Computer Science and Engineering, Central South University, Changsha 410083, China (e-mail: mingfenghuang@csu.edu.cn; afengliu@mail.csu.edu.cn).

W. Liu is with the School of Informatics, Hunan University of Chinese Medicine, Changsha 410208, China (e-mail: weiliu@csu.edu.cn).

T. Wang is with the College of Computer Science and Technology, Huaqiao University, Xiamen 361021, China (e-mail: wangtian@hqu.edu.cn).

S. Zhang is with the School of Computer Science and Engineering, Central South University, Changsha 410083, China, and also with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China (e-mail: sgzhang@csu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2019.2952767

2327-4662 © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

Obviously, combining the powerful computing and storage capability of the cloud server with the close-range advantage of MEC servers, and performing the task offloading based on the cloud–MEC computing model, can better satisfy the diverse requirements of applications for latency, energy consumption, and security, and make the workload of network devices more balanced. Although there are some studies in this field, there are still some challenging issues.

1) The cloud–MEC computing model is more complex than the previous models, so its research is more challenging. In the pure cloud computing (or MEC computing) model, just consider which cloud server (or which MEC server) the offloading task is to be given to. In the cloud–MEC computing model, it is first decided whether to offload the task to the cloud server or MEC servers (i.e., making the task offloading decision), and then it is considered which cloud server or MEC server the offloading task is to be given to (i.e., selection of specific offloading terminal), and thus the offloading in this hybrid model is more complicated. In addition, the resource migration overhead involved in task offloading is also a problem to be considered [14] because frequent migrations between edge and cloud will impose communication delays, as well as resulting in latency, bandwidth, and inefficient energy consumption [14], [15].
2) In previous task offloading researches, tasks often refer to the abstraction of operations that consume certain computing and storage resources. In such researches, each time mobile devices submit an offloading task request, the system will choose an appropriate MEC server or cloud server to perform the task. Therefore, the previous task offloading is based on a single task, is independent and separated, and does not consider the data correlation and operation relevance between tasks. Moreover, the previous task offloading only focuses on task execution, and does not achieve load balancing in the whole life cycle from data gathering, data transmission, and task execution to results returning. In other words, these researches do not consider system-level applications, resulting in network performance and external application services have limitations. In the actual network, especially, with the development of big data network, many applications perform tasks based on massive data [16], [17], [20]. It is estimated that the global data traffic is expected to reach 49EB by 2021 [9]. These massive data construct the basis of current Internet applications. Many Internet technology companies, such as Google, Microsoft, Amazon, Baidu, Alibaba, etc., are based on big data. Many emerging applications include VTrack and Waze [18], which generate omnipresent traffic information; Weather Lah, which generates fine-grained conditions on the ground, and NoiseTube [19] also rely heavily on the large amount of data perceived by the monitoring area. Obviously, the tasks under these big data-based applications are different from those in the previous task offloading researches. They have the following characteristics.
   a) The amount of data generated continuously and transmitted upstream in these applications is very huge, reaching the EB level. The transmission and processing of these huge amounts of data results in severe network load, and the processing offload of these data will produce completely different network performance for different MEC servers or cloud server.
   b) In such big data-based applications, data access is the most frequent, and related data access or duplicate data access is likely to exist, which requires the establishment of a global systematic task offloading to achieve maximum resources effectiveness. However, the current research does not consider it.
3) A shortcoming in the current offloading research is that the potential of network softwarization paradigms is not fully utilized. The offloading strategies proposed at the present increases the additional communication load caused by resources migration. However, we find that the introduction of network softwarization paradigms can break this limitation. For example, using the software-defined network (SDN) technology can reduce network load. We propose an orchestrating data as service networking (ODSN) framework in [21], which is similar to software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Recently, we have found that this orchestrating data as services (ODaS) mechanism can also be used in task offloading to effectively improve the network performance. The main idea is that the large amount of metadata can be processed by MEC servers during the upstream transmission process, and then orchestrating them into services by using the SDN technology, which is equivalent to offloading the data processing tasks and request-response tasks to MEC servers. This task offloading not only can achieve load balance, but also significantly reduce the network transmission data amount. According to previous researches [21], [22], the communication load can be reduced by an order of magnitude by ODaS.

Based on the above analysis, a cloud–MEC collaborative task offloading scheme with service orchestration (CTOSO) is proposed in this article. Compared with previous researches, the contributions of this article are as follows.

1) A cloud–MEC collaborative task offloading scheme (CTOSO) is proposed in this article, which solves the key issues of task offloading decision making and specific offloading terminal selection. First, the CTOSO scheme establishes the computational model, communication model, and delay model of task offloading and derives a target optimization function based on the models. Then, according to the task computation and latency requirements, the CTOSO adopts a differentiated offloading decision mechanism. For resource-demanding or delay-tolerant tasks, they are offloaded to the cloud server, while delay-sensitive tasks are offloaded to MEC servers. Finally, based on

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

the preestimation of the delay and energy of tasks on MEC servers, the MEC server that minimizes the weighted sum of energy consumption and delay is selected as the offloading terminal.

2) Unlike other solutions that only focus on independent task offloading, the CTOSO scheme introduces two important changes to effectively improve the network performance. First, the CTOSO scheme not just focuses on the single-task offloading, but integrates the entire process from data collection, data transmission, and service orchestration to service access into the offloading scheme of this article, and these cover almost the entire load of the network, enabling CTOSO scheme to reduce network costs at the system level. In addition, the CTOSO scheme innovatively introduces the ODaS technology, which for the first time breaks through the shortcomings of the previous offloading strategies that balance the network load but increase the additional communication load. It is the first offloading scheme that can achieve both load balancing and communication load optimization. At the same time, MEC servers cache the forwarded data and the generated services, which can greatly reduce the transport and secondary computing overhead.
3) The experimental results demonstrate that the CTOSO scheme greatly reduces latency while maintaining efficient energy consumption. Compared to the random decision-based noncache nonservitization offloading scheme NCNS-R and the maximum cache-based nonservitization offloading scheme CNS-C, the CTOSO scheme reduces delay by approximately 73.82%–74.34% and energy consumption by 10.71%–13.73%.

The rest of this article is organized as follows. In Section II, the related works are reviewed. The system model and problem statement are presented in Section III. In Section IV, the CTOSO scheme is proposed. Then, Section V provides the experimental results. The conclusions are given in Section VI.

II. Related Work

Task offloading schemes can be roughly divided into four categories: 1) the task offloading scheme based on binary offloading; 2) the task offloading scheme based on partial offloading; 3) the task offloading scheme based on random task model; and 4) the cloud–MEC collaborative task offloading scheme.

In the binary offloading scheme, each task is treated as a whole and cannot be further partitioned. It can only be executed either locally on the mobile device or remotely offloaded to MEC servers [23]–[25]. The binary task offloading is usually a typical mixed-integer nonlinear programming problem and is generally an NP-hard problem.

Game theory is widely used to solve binary computational offloading issues with restrictive objectives and multiuser participation. Liu *et al.* [23] studied two data offloading mechanisms among multiple mobile users based on game theory, and modeled the decentralized data offloading decision-making problem as multi-item auction (MIA)-based data offloading and congestion game (COG), where MIA can maximize the revenue of mobile users, COG can minimize the payment of mobile subscribers (MSs), and implement Nash equilibriums of the game. Messous *et al.* [24] proposed a strategy based on game theory, which can be applied to the MEC paradigm to solve the problem of offloading heavy computing tasks of unmanned aerial vehicles (UAVs).

Bi and Zhang [25] proposed a binary computation offloading scheme in a multiuser MEC network powered by the WPT, and a simple bi-section search algorithm and a joint optimization method based on the alternating direction method of multipliers (ADMMs) are given in the scheme, and solve the combinatorial nature of the multiuser computing mode selection and its strong coupling with the transmission time allocation.

In recent years, with the development of code decomposition and parallel computing, partial computational offloading in the MEC network has attracted wide attention. In the partial computational offloading model, a task is decomposed into different parts, some are performed locally on mobile devices, and some are offloaded to MEC servers, so resource allocation is the key problem of this kind of scheme. Kuang *et al.* [9] considered the partial offloading scheduling and resource allocation for MEC systems with multiple independent tasks, and model it as a nonconvex mixed-integer optimization problem by proposing a two-level alternation method framework based on the Lagrangian dual decomposition, and implement near-optimal delay performance.

Ning *et al.* [26] started with a single-user computation offloading problem, using the branch-and-bound algorithm to solve partial computation offloading in delay-constrained networks, and expressed the multiuser computation offloading problem as a mixed-integer linear programming problem by designing an iterative heuristic MEC resource allocation algorithm to make the offloading decision dynamically. Ahn *et al.* [27] studied the partial task offloading from the perspective of a noncooperative game and designed an energy-oriented task scheduling scheme. Under this scheme, authors give a joint consideration of the destination and client sides.

Compared with the deterministic task model, the average energy consumption and execution latency of the random task model system are more correlated. Therefore, the design of the MEC system with random task arrival is a less explored field. Assuming that the computational requests of mobile users obey the Poisson distribution, Liu *et al.* [28] studied the computational offloading of the MEC system based on the queuing theory, and find the best offloading probability and optimal transmission power of the mobile user by scaling scheme and interior point method.

An optimization framework of mobile devices with fixed CPU frequency and elastic CPU frequency is proposed in [14]. For the fixed CPU frequency case, authors give a linear relaxation-based approach and a semi-definite relaxation (SDR)-based approach and propose an exhaustive search-based approach and an SDR-based approach for the elastic CPU frequency case.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

Because MEC is close to mobile users, mobile users can obtain the required computing resources only by one-hop wireless transmission. Therefore, many researchers propose that task computing can be accomplished through the cooperation of mobile devices and MEC servers. Ridhawi *et al.* [29] proposed an MEC and mobile devices collaborative framework to handle delay sensitive and context-aware applications, the proposed solution decomposes cloud data into a set of files and services, and frequently requested files and services are cached onto user mobile devices, thus providing fast delivery of the requested cloud composite services to end users.

However, the computing and storage capacity of the MEC servers are limited. With the explosive growth of mobile users and emerging applications, the MEC alone cannot fully and efficiently satisfy users' computing offload requests. Therefore, researchers propose a cloud–MEC collaborative task offloading scheme that utilizes the powerful computing power of the cloud server to handle highly complex but nondelay-sensitive tasks, while MEC servers perform delay-sensitive tasks to meet the diverse needs of users.

Guo and Liu [30] proposed a cloud–MEC collaborative computation offloading architecture by adopting the FiWi access networks. Under this framework, an approximation collaborative computation offloading scheme and a game-theoretic collaborative computation offloading scheme are proposed to achieve better offloading performance. Du *et al.* [31] proposed a low-complexity suboptimization algorithm that takes into account user fairness and minimizes the user with the largest delay and energy weighting of all users. The optimization algorithm obtains the optimal offloading decision by semi-determined relaxation and randomization, and obtains the optimal resource allocation through the fractional programming theory and the Lagrangian dual algorithm.

III. System Model and Problem Statement

A. Network Architecture

The network architecture in this article is an integration framework of cloud, MEC, and IoT. The network scenario consists of many IoT devices, MEC servers, and a cloud server. Its hierarchical structure is shown in Fig. 1. The cloud server is also called the data center, and is composed of core network devices with powerful computing ability and huge storage capacity. Based on the service requests of users, the cloud server performs in-depth data analysis and processing. MEC servers are located at the edge layer and implement data calculation localization by deploying some hardware devices at the edge of the network to support data refinement and processing. MEC servers are served by various heterogeneous network devices, which have certain computing and storage capabilities. Numerous IoT devices, such as smartphones, sensing devices, monitoring probes, and radio frequency identification devices [20], [21], are deployed on the bottom of the network. These devices are distributed in different scenarios for environmental monitoring, industrial control, and public or civilian data collection [22]. Some data collection devices, such as smart-phones and computers, can also be used as application terminals to send requests to the cloud server.

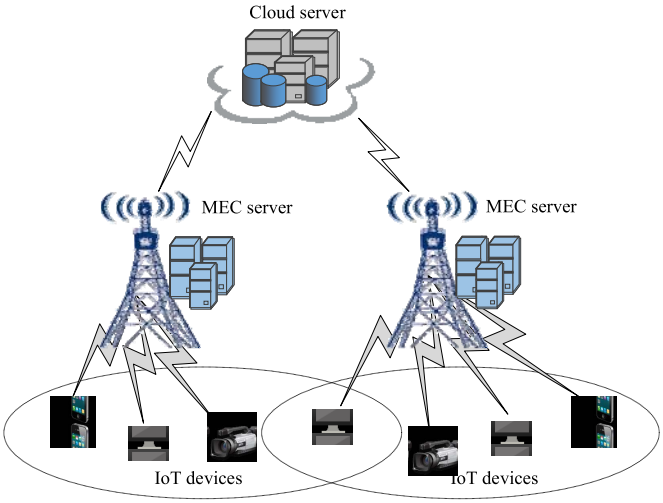


Fig. 1. Cloud–MEC collaborative network framework.

The normalized network model of this article is defined as follows: assume that the set of MEC servers is $\mathbb{M}$, $\mathbb{M} = \{\text{MEC}_i | i = 1, \ldots, N\}$, $|\mathbb{M}| = N$. For each MEC server MEC_i, $\text{MEC}_i = \{\mathcal{Q}^i_{\text{MAX}}, \mathcal{R}^i_{\text{MAX}}\}$, where $\mathcal{Q}^i_{\text{MAX}}$ is the maximum available resource of the server. For each task, the task can be offloaded to MEC_i only when its required calculation resource is less than or equal to $\mathcal{Q}^i_{\text{MAX}}$, and $\mathcal{R}^i_{\text{MAX}}$ is the maximum capacity of the local storage space of MEC_i.

B. Task Offloading Model

In this article, a binary cloud–MEC collaborative task offloading model is adopted. In this model, mobile devices act as big data collectors and can also serve as terminals for requesting data. Therefore, all tasks generated in the network are assigned to MEC servers or the cloud server for execution. Suppose that the set of all tasks to be executed in the network is $\mathbb{T}$, $\mathbb{T} = \{T_i | i = 1, \ldots, M\}$, $|\mathbb{T}| = M$. For each task T_i, the system first makes task offloading decision, and defines $A_i \in \{0, 1\}$ as the offloading decision of T_i, 0 means the task is offloaded to the cloud server, and 1 means the task is offloaded to MEC servers. Therefore, the set of task decisions can be obtained, which is $\mathbb{A} = \{A_1, A_2, A_3, \ldots\}$. The offloading operation can be described as $T_i = \{A_i, \mathcal{Q}^i_c, \mathcal{X}^i_{\text{tot}}, \mathcal{T}^i_{\text{MAX}}\}$, where A_i is the offloading decision of T_i, $A_i \in \{0, 1\}$, $\mathcal{Q}^i_c$ is the resource needed for the execution of T_i, $\mathcal{X}^i_{\text{tot}}$ is the data amount involved in the execution of T_i, $\mathcal{T}^i_{\text{MAX}}$ is the maximum tolerable delay of T_i, that is, the deadline for the execution of T_i.

C. Problem Statement

The goal of this article is to design an efficient cloud–MEC collaborative task offloading scheme to ensure that the overall energy consumption and latency are reduced while satisfying the service requirements. For the task T_i, whether the task is executed by the cloud server or MEC servers, we uniformly set the total consumption as E^i_{tot}, E^i_{tot} includes communication energy consumption, transmission energy consumption, task execution energy consumption, and set the total delay as $\mathcal{D}^i_{\text{tot}}$, $\mathcal{D}^i_{\text{tot}}$ includes communication delay, transmission delay, and

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

task execution delay. Then, the joint optimization objective function of the task T_i on offloading decision A_i is as follows:

$$\begin{aligned} \text{Min} \quad & S = \min_{\forall T_i, \forall A_i} \left[\lambda E^i_{\text{tot}} + (1-\lambda)\mathcal{D}^i_{\text{tot}}\right] \quad \forall i \in M \\ \text{s.t.} \quad & \text{C1: } \mathcal{D}^i_{\text{tot}} < \mathcal{T}^i_{\text{MAX}} \\ & \text{C2: } \mathcal{Q}^i_c \le \mathcal{Q}_{\text{MAX}} \\ & \text{C3: } A_i \in \{0, 1\}. \end{aligned} \tag{1}$$

λ in (1) is a weight parameter used to achieve a dynamic tradeoff between delay and energy consumption. C1–C3 are constraints on (1), where C1 indicates that the execution time of task T_i must be within its tolerant delay $\mathcal{T}^i_{\text{MAX}}$, C2 indicates that the resources required for task execution cannot be greater than the available resources of the executing server, and C3 indicates that the task is inseparable and can only be performed completely by the cloud server or an MEC server.

IV. Design of the CTOSO Scheme

A. Overview of the CTOSO Scheme

A cloud–MEC CTOSO is proposed in this article. First, the terminology involved in the CTOSO scheme is explained.

Data flow, refers to the transmission stream of data resources in the network. The data resources, in this article, include the data perceived and collected by various IoT devices, which are metadata containing redundant and invalid information.

Service flow, refers to the transmission stream of service resources in the network. In this article, the service refers to the resource obtained by processing the data with the ODaS technology by MEC servers or the cloud server. There are several differences between data and services. First, compared with data, the service has a smaller data volume and a higher data quality. Second, the service is usually the result of aggregating multiple data packets, so they are different in packet format. Each packet can be divided into the packet header and the data part, where the header is used to identify the source address, destination address, data length, checksum, transmission protocol, and other information for transmission. When multiple data packets are orchestrating as a service, the data part of packets is compressed and only one packet header is reserved. This is why the service has a smaller data volume than the data. Finally, the service is the valid information that can be directly invoked by users, but the data is impurity metadata containing invalid information and cannot be used directly by users. The service resources in this article include: 1) the resource obtained after the metadata are further processed by MEC servers; 2) task execution results of the cloud server; and 3) Task execution results of the MEC servers.

Service request refers to the network request initiated by mobile terminals for a specific service resource.

Service orchestration, which is a concept proposed in the SDNs, refers to the process of orchestrating the data into services. Its process in the content deliver network (CDN) is as follows. First, the service providers distribute programming codes to edge devices, then edge devices use the software programming codes and hardware resources, combined with the network coding technology and the data processing technology, to convert the data into services that can be directly used by users. The integration of the processes and technologies involved in the service orchestration is called the ODaS mechanism.

The process under the CTOSO scheme is as follows.

1) In the data gathering phrase, the IoT devices located at the bottom of the network continuously collect data and encapsulate the data into data packets, and transmit them to MEC servers.
2) MEC servers receive the data and process the data locally, first clearing the invalid, erroneous, and redundant information in the data, and then arranging the remaining valid data as services by using ODaS technology and programing codes distributed by the system. Cache services in local storage. If the storage space capacity reaches the maximum, the least recently used (LRU) replacement algorithm is used to update the content. Finally, MEC servers forward these services to the cloud server.
3) The cloud server receives these services and stores them in the cloud storage space. Note that although MEC servers cache part service locally, all service resources of the entire network are still stored in the storage space of the cloud server.
4) During the data access phase, the mobile user sends a service request for requiring a specific service. Then, the system first parses the content of the request, and queries the service storage table to determine whether the request has been processed before. If the request is processed, the resource is returned directly from the corresponding storage node (the cloud server or a specific MEC server). Otherwise, a new task is established for the request, and an offloading decision is made based on task attributes, such as the data amount, computing resources, and tolerate latency.
5) When making task offloading decision, a task is considered as an independent object and can only be executed completely by the cloud server or an MEC server, and cannot be divided into multiple parts. For each task T_i, its execution operation can be described as $T_i = \{A_i, \mathcal{Q}^i_c, \mathcal{X}^i_{\text{tot}}, \mathcal{T}^i_{\text{MAX}}\}$, and the system makes the offloading decision based on the $\mathcal{Q}^i_c$ and $\mathcal{T}^i_{\text{MAX}}$, the specific decision algorithm is given in the next section. For the decision result, $A_i = 0$ means the task is offloaded to the cloud server, $A_i = 1$ means the task is offloaded to MEC servers for execution, and all tasks adopt the first come first serve (FCFS) scheduling principle.
6) If the task is offloaded to the cloud server, since the network scenario in this article only contains one cloud server, and the computing power and storage capacity of the cloud server are very powerful, the task T_i is directly added to the task cache queue of the cloud server, and the task-related resources are offloaded to the cloud server. After the task is executed, the cloud server arranges the execution result into a service and returns it to the user.
7) If the task is offloaded to MEC servers for execution, it is necessary to determine an MEC server that minimizes

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

the overall cost as the offloading terminal. First, the computing resources and data resources required for the task execution are determined according to the resource content requested by T_i. Then, retrieve the global caching data to determine the cache status of T_i and the remaining computing resources of each MEC server. Finally, based on the communication model, the calculation model and the delay model (the specific model will be given in the next section), the energy consumption and delay of offloading task for each MEC server are pre-estimated, and according to the joint objective optimization function (1), the target MEC server is selected and then resources are offloaded to it.

8) The MEC server performs the task and synthesizes the results into a service, then returns the service directly to the user, while caching the service in the local storage space.

B. Energy Consumption and Delay Modeling

In this section, the communication energy consumption, computational energy consumption, and task delay involved in task offloading under the CTOSO scheme are modeled.

1) Communication Model: Communication energy consumption refers to the energy consumption generated by migrating resources from the mobile user to the server (the cloud server or the MEC server) during task offloading and the execution results back from the server to the mobile user after the task is completed, which is related not only to the amount of data transferred but also to the bandwidth and actual transmission rate of the wireless network.

Assuming that H_k is the channel gain between the sender and the receiver, and during the resource migration, both the resource sender and receiver do not move, then H_k is constant, P_s is the transmission power of the sender, σ^2 is the noise power consumption, b_w is the wireless channel bandwidth between the server and mobile user, then the transmission rate between the server and mobile user is

$$\varphi_C^k = b_w \log_2\left(1 + \frac{P_s H_k}{\sigma^2}\right). \tag{2}$$

Assuming that the total amount of data involved in the task execution is $\mathcal{X}_{\text{tot}}$, where the server has cached $\mathcal{X}_{ch}^k$, and the amount of data obtained after processing is β ($0 \le \beta < 1$) times the original data, then the downlink transmission time and uplink backhaul time of resources are

$$t_{\text{com}}^{\text{down}} = \frac{\left(\mathcal{X}_{\text{tot}} - \mathcal{X}_{ch}^k\right)}{\varphi_C^k} \tag{3}$$

$$t_{\text{com}}^{\text{up}} = \frac{\beta \mathcal{X}_{\text{tot}}}{\varphi_C^k}. \tag{4}$$

Assuming that the communication power consumption per unit time is $\mathfrak{R}_{\text{com}}$, then the total communication energy consumption of offloading tasks from the mobile user to the server is as follows:

$$E_{\text{com}} = \mathfrak{R}_{\text{com}}\left[\frac{\left(\mathcal{X}_{\text{tot}} - \mathcal{X}_{ch}^k\right)}{\varphi_C^k} + \frac{\beta \mathcal{X}_{\text{tot}}}{\varphi_C^k}\right]. \tag{5}$$

2) Computation Model: For each task T_i, $T_i = \{A_i, \mathcal{Q}_c^i, \mathcal{X}_{\text{tot}}^i, \mathcal{T}_{\text{MAX}}^i\}$, where $\mathcal{Q}_c^i$ is the CPU resource required for the execution of T_i, and $\mathcal{X}_{\text{tot}}^i$ is the data volume involved in the execution of T_i. Based on the offloading decision A_i, the computational energy consumption of T_i executed by the cloud server or the MEC server can be obtained.

When $A_i = 0$, the task is offloaded to the cloud server. The calculation energy consumption includes data query energy consumption and task execution energy consumption, let f_C^c represent the computing power of the cloud server, f_C^s represent the data retrieval capability of the cloud server, $\mathcal{X}_{\text{tot}}^C$ represent the total amount of data cached by the cloud server, and $\mathfrak{R}_C^s$ is the data query power consumption per unit time of the cloud server. According to [32], the energy consumption of a single CPU cycle is expressed as $E = \kappa (f_C^c)^2$, the calculation energy consumption of the task T_i when it is executed by the cloud server is as follows:

$$E_{\text{cal}}^C = \mathfrak{R}_C^s\left(\frac{\mathcal{X}_{\text{tot}}^C}{f_C^s}\right) + \kappa\left(f_C^c\right)^2 \mathcal{Q}_c^i. \tag{6}$$

Among them, κ is an energy factor, and its size depends on the CPU chip technology. In this article, $\kappa = 10^{-26}$ [32], the former part of (6) represents the data retrieval consumption. In fact, this part of the consumption is very small and can be omitted, and the total consumption mainly comes from the task execution consumption.

When $A_i = 1$, the task is offloaded to the MEC server for execution. Assuming that the total data volume of the task T_i is $\mathcal{X}_{\text{tot}}$, where the MEC server has cached $\mathcal{X}_{ch}^k$ locally, and f_M^c is the computing capability of MEC_k, and f_M^s represents the data retrieval capability of MEC_k, then the calculation energy consumption of the task T_i when it is executed by MEC_k is as follows:

$$E_{\text{cal}}^M = \mathfrak{R}_M^{\text{s}}\left(\frac{\mathcal{X}_{ch}^k}{f_M^s}\right) + \mathfrak{R}_M^c\left(\frac{\mathcal{Q}_c^i}{f_M^c}\right) \tag{7}$$

where $\mathfrak{R}_M^{\text{s}}$ represents the data retrieval power consumption of MEC_k in a unit time, and $\mathfrak{R}_M^c$ is the calculation power consumption of MEC_k in a unit time.

3) Delay Model: Like the computing model, when the task is executed by the cloud server, its delay is as follows:

$$\mathcal{D}_{\text{tot}}^C = \left[\frac{\left(\mathcal{X}_{\text{tot}} - \mathcal{X}_{ch}\right) + \beta \mathcal{X}_{\text{tot}}}{\varphi_C}\right] + \frac{\mathcal{X}_{\text{tot}}^C}{f_C^s} + \frac{\mathcal{Q}_c^i}{f_C^c}. \tag{8}$$

When the task is executed by the MEC server, its delay can be calculated as

$$\mathcal{D}_{\text{tot}}^M = \left[\frac{\left(\mathcal{X}_{\text{tot}} - \mathcal{X}_{ch}^k\right) + \beta \mathcal{X}_{\text{tot}}}{\varphi_C^k}\right] + \frac{\mathcal{X}_{ch}^k}{f_M^s} + \frac{\mathcal{Q}_c^i}{f_M^c}. \tag{9}$$

C. Task Offloading Algorithm Under CTOSO Scheme

Assuming that M tasks are generated in one cycle, and an offloading decision is required for each task, let $\mathbb{A}$ represent the set of offloading decisions of these M tasks, $\mathbb{A} = \{A_i | i = 1, \ldots, M\}$. Denote the set of MEC servers as $\mathbb{M}$, $\mathbb{M} = \{\text{MEC}_k | k = 1, \ldots, N\}$. Before making the first round decisions, the network initial settings are as follows: the available resources of each MEC server is the maximum value,

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

$\mathcal{Q}_{\text{res}} = (\mathcal{Q}^1_{\text{res}}, \ldots, \mathcal{Q}^j_{\text{res}}, \ldots, \mathcal{Q}^N_{\text{res}}) = \mathcal{Q}_{\text{MAX}}$, the total energy consumption and delay is zero, $E = (E_1, \ldots, E_j, \ldots, E_N) = 0$, $\mathcal{D} = (\mathcal{D}_1, \ldots, \mathcal{D}_j, \ldots, \mathcal{D}_N) = 0$.

For each task T_i, the process of decision making and task offloading is as follows.

1) It is determined whether the task is a delay-sensitive task or a delay-tolerant task according to the maximum tolerance delay $\mathcal{T}^i_{\text{MAX}}$ and the delay threshold $\mathcal{T}_{bd}$. If $\mathcal{T}^i_{\text{MAX}} \geq \mathcal{T}_{bd}$, the task is a delay-tolerant task, the time $\mathcal{T}_\Theta$ of executing the task by the cloud server is estimated based on the computing load of the cloud server, $\mathcal{T}_\Theta = [(\sum_{i=1}^{N} \mathcal{Q}^i_c)/(f^c_C)] + t_{ex}$, where $[(\sum_{i=1}^{N} \mathcal{Q}^i_c)/(f^c_C)]$ is the time required to execute other tasks already in the task queue, t_{ex} is the other delay that may exist, such as data retrieval delay, etc. If the $\mathcal{T}_\Theta$ is less than $\mathcal{T}^i_{\text{MAX}}$, then $A_i = 0$, indicating that the task T_i is offloaded to the cloud server.
2) If $\mathcal{T}^i_{\text{MAX}} < \mathcal{T}_{bd}$, the task is a delay-sensitive task, then the computational resource $\mathcal{Q}^i_c$ required for the execution of T_i is further determined. If at least one MEC_k can be found in the set $\mathbb{M}$ to satisfy the requirement that its available computational resource $\mathcal{Q}^k_{\text{res}}$ is larger than $\mathcal{Q}^i_c$, the offloading decision of T_i is $A_i = 1$; otherwise $A_i = 0$.
3) Next, the specific MEC server is selected. First, the remaining available resource $\mathcal{Q}^k_{\text{res}}$ of each MEC server in the set $\mathbb{M}$ is determined, and the MEC server that does not satisfy the condition $\mathcal{Q}^k_{\text{res}} > \mathcal{Q}^i_c$ is deleted from $\mathbb{M}$ to get a new set $\mathbb{M}_{\text{cad}}$.
4) According to (1)–(9), the energy consumption and delay of each MEC server in $\mathbb{M}_{\text{cad}}$ are estimated. The sum of weights of energy consumption and delay of each MEC server performing T_i is obtained
$$S_{ik} = \lambda E^{i,k}_{\text{tot}} + (1-\lambda)\mathcal{D}^{i,k}_{\text{tot}}. \tag{10}$$
5) MEC servers in $\mathbb{M}_{\text{cad}}$ are arranged in ascending order of S_{ik}, and get the set $\overline{\mathbb{M}}$. According to the constraint condition of (1), each MEC server in the set $\overline{\mathbb{M}}$ is traversed in order from the arrival to the end until an MEC server is found to satisfy all the constraints and is used as the offloading terminal of the task T_i, then the available resources of the MEC server are updated for the next round of decision making, $\mathcal{Q}^k_{\text{res}}(i+1) = \mathcal{Q}^k_{\text{res}}(i) - \mathcal{Q}^i_c$.
6) Resources of T_i is offloaded to the cloud server or the target MEC server. If there are still tasks to be determined in the system, then enter the next round of decision, otherwise the algorithm ends.

To more clearly demonstrate the task offloading under the CTOSO scheme, we present a complete description of the task offloading and decision-making process in Algorithms 1 and 2. Assuming that the set of all tasks to be offloaded to MEC servers is $\mathbb{T}_{\text{off}}$ and the number of MEC servers is N, the task offloading is as shown in Algorithm 1.

The algorithm mainly includes the following steps: 1) traversing the available computing resources of the N MEC servers to filter the servers that can be used as the offloading terminal, and obtain the set $\mathbb{M}_{\text{cad}}$, the complexity of this part is $O(N)$; 2) traversing all MEC servers in the set $\mathbb{M}_{\text{cad}}$ ($|\mathbb{M}_{\text{cad}}| \leq N$), and calculate the S_{ik} of each MEC server, which has a complexity of up to $O(N)$; 3) sorting the set $\mathbb{M}_{\text{cad}}$, assuming we adopt the fast sorting algorithm, the complexity is $O(N \log N)$; and 4) traversing the sorted $\mathbb{M}_{\text{cad}}$, only need to find an MEC server that satisfies the condition, so the complexity of this part is $O(1)$. Overall, the complexity of Algorithm 1 is $O(N \log N)$.

Algorithm 1 Task Offloading Algorithm

Initial: $S_{ik} = 0$, j=0, k=0, t=0
Input: Required computing resources $\mathcal{Q}^i_c$ and maximum allowable delay $\mathcal{T}^i_{\text{MAX}}$ of task T_i

For each $T_i \in \mathbb{T}_{\text{off}}$ **Do**
  While j<N **Do**
    If $\mathcal{Q}^j_{res} > \mathcal{Q}^i_c$ **Do**
      Let $MEC_j \in \mathbb{M}_{cad}$
    End if
    j=j+1
  End while
  For each $MEC_k \in \mathbb{M}_{cad}$ **Do**
    Compute S_{ik} of MEC_k based on Eq. (10)
  End for
  Sort the $\mathbb{M}_{cad}$ in ascending order of S_{ik}
  While t< $|\mathbb{M}_{cad}|$ **Do**
    If $\mathcal{D}^t_{tot} < \mathcal{T}^i_{MAX}$ **Do**
      index=t
      t= $|\mathbb{M}_{cad}| - 1$
    End if
    t=t+1
  End while
  Offload resource of T_i to MEC_{index}
  Let $\mathcal{Q}^{index}_{res}(i+1) = \mathcal{Q}^{index}_{res}(i) - \mathcal{Q}^i_c$
End for

Output: the index of MEC server that offload task T_i

Algorithm 2 Task Offloading Decision Making Algorithm

Initial: $A_i = 0$, j=0
Input: Required computing resources $\mathcal{Q}^i_c$ and maximum allowable delay $\mathcal{T}^i_{MAX}$ of task T_i

For each $T_i \in \mathbb{T}_{dec}$ **Do**
  Compute the estimated execution time $\mathcal{T}_\Theta$ of T_i
  If $\mathcal{T}^i_{MAX} \leq \mathcal{T}_{bd}$ or $\mathcal{T}_\Theta > \mathcal{T}^i_{MAX}$ **Do**
    While j<N **Do**
      If $\mathcal{Q}^j_{res} > \mathcal{Q}^i_c$ **Do**
        $A_i = 1$
        break
      End if
      $j = j + 1$
    End while
  End if
  If $A_i = 0$ **Do**
    Offload T_i to the cloud server
  End if
  If $A_i = 1$ **Do**
    Goto Algorithm 1
  End if
End for

Output: task offload decision A_i of task T_i

Assuming that the set of all tasks to be made decision is $\mathbb{T}_{\text{dec}}$, the boundary of delay-sensitive and delay-tolerant tasks is $\mathcal{T}_{bd}$, and the number of MEC servers is N, then for each task T_i, its offloading decision-making process can be illustrated by Algorithm 2.

Algorithm 2 mainly includes the following steps.

1) Estimating the execution time $\mathcal{T}_\Theta$ of T_i and determining the task sensitivity, where the complexity is $O(1)$.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

2) Traversing the available computing resources of the N MEC servers. Note that this traversal is ended as long as an MEC server is found to have more resources available than the resources required by the task T_i, so the complexity of this part can be recorded as a constant level $O(1)$ in general cases, and $O(N)$ in the worst-case.
3) Invoking different execution operations for different decisions, the complexity is $O(1)$. In summary, the time complexity of Algorithm 2 is $O(1)$ in the best case and $O(N)$ in the worst-case.

D. Service Orchestration and Caching

Different from previous researches, in this article, the metadata collected by the IoT devices will be orchestrated as services in the uplink transmission to the cloud via the MEC servers. According to [21], after the data are converted to service, the network transmission load will be reduced by an order of magnitude. Before the service orchestration, the cloud server distributes programming codes to various MEC servers. These programing codes reside locally on MEC servers, and coordinate with local hardware resources of MEC servers to arrange the calculation results into services.

Task computing is like refining, correlating, and combining multiple metadata to meet the information needs of particular applications. When the calculation results are encapsulated as a service, the data amount is further compressed and reduced. Suppose that the set of data packets involved in the computation of task T_i is $\mathbb{P}$, $\mathbb{P} = \{\text{Pac}_1, \text{Pac}_2, \ldots, \text{Pac}_K\}$, the data volume of these data packets is Π, then the data result of the task calculation is as follows:

$$(T_i) = \sum_{i=1}^{K} \sum_{j=1, j\neq i}^{K} \left[\text{Max}\left(\Pi_i, \Pi_j\right) + \left(1 - \lambda_{i,j}\right)\text{Min}\left(\Pi_i, \Pi_j\right)\right] \quad (11)$$

where $\lambda_{i,j}$ is the correlation coefficient of Pac_i and Pac_j, and $\lambda_{i,j}$ is calculated as

$$\lambda_{i,j} = \frac{1}{e^{\left(d_{i,j}^2/\gamma\right)}}. \quad (12)$$

$d_{i,j}$ is the distance between the data collection devices that generates Pac_i and Pac_j, and γ is the constant of the correlation coefficient. When γ keeps approaching 0, $\lambda_{i,j}$ also approaches 0, indicating that the data has no correlation; conversely, the larger the γ, indicates the higher the service correlation.

Meanwhile, if orchestrating task results into services with a compression ratio of δ, the data servitization result is

$$\Gamma_s(T_i) = \delta[(T_i)]. \quad (13)$$

In addition, the CTOSO scheme, in this article, introduces the in-network caching mechanism. Each MEC server in the network has a unique identifier M_ID, and each service also has a unique identity S_ID. Services are stored in chunks, and cached in the local storage space of MEC servers. The services are divided into equal size content chunks, the size of each content chunk is 10 KB, and since each MEC server stores multiple services, each MEC server needs to maintain a service caching table to map with specific cached content.

TABLE I
SERVICE CACHING TABLE OF MEC0014

M_ID	S_ID	C_chunks	Size
MEC0014	SERa00E0	C[1]-C[M1]	M1*10KB
MEC0014	SERa00E1	C[1]-C[M]	M*10KB
.....	.....	.....	.....
MEC0014	SERa00N6	C[1]-C[M2]	M2*10KB

Algorithm 3 Service Caching Algorithm

Initial: Max_time=0, Del_index=NULL
For each New_S of MEC_i **Do**
  If Res_S>New_S.Size **then**
    Loop: Divide New_S into M content chunks
    Cache M content chunks locally
    Insert a statement "$MEC_i.ID$;New_S.ID; C[1], C[2],...,C[M]; New_S.Size" into Service Caching Table
  End if
  Else
    For each service S_k caching in MEC_i **Do**
      If Max_time<S_k.N_time and S_k.Size>New_S.Size **then**
        Max_time=S_k.N_time
        Del_index=S_k.ID
      End if
    End for
    Remove S[Del_index] from the storage of MEC_i
    Goto Loop
  End else
End for

Table I shows the service cache table of the MEC server numbered MEC0014, which accurately records the block storage of the services SERa00E0 to SERa00N6. There are four fields in the table, namely, M_ID, S_ID, C_chunks, and Size, each service corresponds to a record, S_ID and M_ID correspond to each other, they are a many-to-one relationship, and C_chunks and Size are used to mark the size of the service. Since the storage capacity of each MEC server is limited, when the storage space of the MEC server is full, the system will clear the most recently unused service.

Suppose the set of MEC servers is $\mathbb{M}$, $\mathbb{M} = \{\text{MEC}_1, \text{MEC}_2, \ldots, \text{MEC}_N\}$, $|\mathbb{C}| = N$. For each server MEC_i, assume that the new service to be cached is N_Ser, and N_Ser is divided into M contents. The chunks are cached, and Res_s is used to represent the remaining service storage space of MEC_i. The N_time of each service indicates its most recently unused time. For each N_Ser, its caching process can be described by Algorithm 3.

The time complexity of Algorithm 3 is O(M) in the worst case, and M is the total number of cached services in the service storage space of MEC_i. At this time, since the service space has reached the maximum capacity, it is necessary to traverse the storage space to find the most recently unused service.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

The experimental scenario is set as follows.

1) The entire network consists of 1 cloud server, 10 MEC servers, and 500 IoT devices. The MEC servers are randomly deployed within 200–1000 m from the cloud

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

TABLE II
PARAMETERS FOR EXPERIMENT

Symbol	Value	Description
$\Re_{com}$	100 mW	Communication power consumption of cloud
$\Re_C^s$	20 mW	Data retrieval power consumption of cloud
f_C^c	10 GHz	Computing capacity of cloud
f_C^s	200 KB/ms	Resource retrieval rate of cloud
$\Re_M^c$	80 mW	Computing power consumption of MEC
f_M^s	20 mW	Data retrieval power consumption of MEC
f_M^c	2-4 GHz	Computing capacity of MEC
f_M^s	10 KB/ms	Data retrieval rate of MEC
$\mathcal{R}_{MAX}$	10.45 MB	Maximum storage capacity of MEC
$\mathcal{Q}_c$	1-5 GHz	Computational resources required for tasks

server, and the IoT devices are distributed within 100–300 m of the MEC server.

2) Each IoT device generates 1–3 packets in one time period, and data size of each packet is 10–30 KB. Each mobile user generates 0–3 tasks in a time period, the data volume of each task is 10–90 KB, and the tolerance delay of the task is 1–4 s.
3) The network bandwidth is 50 MHz, $H_k = 127 + 30 \times \log_d$, d represents the distance between the server and the mobile user. The sending power is 1 W, the noise power consumption is 2×10^{-13}, other parameters are shown in Table II.

To evaluate the performance of the CTOSO scheme in terms of energy consumption and delay, we select two comparison schemes, the first is a noncached nonservitization offloading scheme based on random decision, named NCNS-R. The NCNS-R scheme adopts the same offloading decision-making principle as the CTOSO scheme, when selecting a specific offloading terminal among MEC servers, the NCNS-R scheme randomly selects one server from all candidate servers as the offloading terminal as long as its computing resources and capabilities can meet the requirements. When the MEC server completes the task, it does not cache the relevant data, but directly returns the result to users. The second scheme is a nonservitization offloading scheme based on maximum cache, named the CNS-C scheme. The CNS-C scheme adopts the caching mechanism and selects an MEC server with the largest amount of cache data related to the task, and with sufficient remaining computing resources as the offloading terminal. After the task is executed, its execution result is cached.

B. Experimental Results of CTOSO Scheme

Figs. 2–7 are the delay, energy consumption, and task distribution under the CTOSO scheme. Fig. 2 illustrates the average transmission delay, communication delay, and task calculation delay when the task is offloaded to MEC servers. It can be seen that as the number of execution tasks increases, the transmission delay does not change much because the transmission delay is closely related to the transmission distance, and MEC servers are deployed in a geographical area closer to users, so the average transmission delay presents an approximately

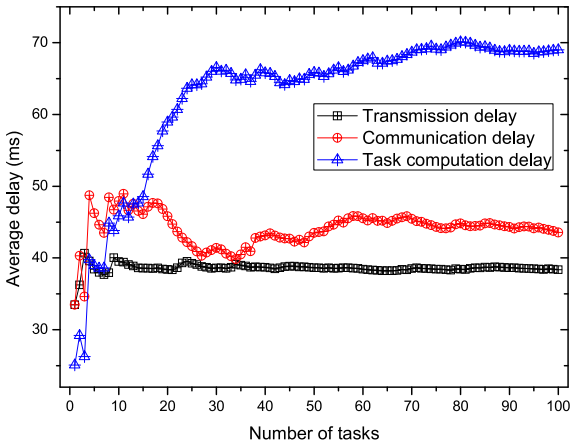


Fig. 2. Average delay of tasks executed by MEC servers.

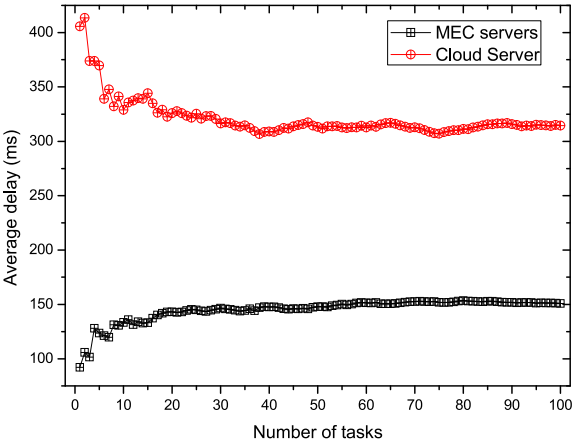


Fig. 3. Average delay of MEC servers and the cloud server.

steady curve. The communication delay has some fluctuations because the communication delay depends on the amount of data offloaded to MEC servers. For different tasks, the values of the two parameters are different, so the communication latency changes more significantly relative to different tasks, but the overall trend is still flat. The average task calculation delay shows an increasing trend as the number of tasks increases. This is because the computing ability of the MEC server is limited. Therefore, as the caching data of MEC servers and the number of tasks to be processed increases, the MEC server takes longer time to perform data retrieval and actual calculation.

Fig. 3 is the average delay of MEC servers and the cloud server under the CTOSO scheme, in addition to the performance fluctuations in the initial of the network, it can be clearly seen that the delay performed by MEC servers is much lower than that of cloud servers, and is reduced by nearly 50%.

Fig. 4 is the average energy consumption of MEC servers. The total energy consumption is composed of transmission energy consumption, communication energy consumption, and task computation energy consumption. Since MEC servers are close to users, the transmission energy consumption is not large. At the same time, MEC servers adopt the

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

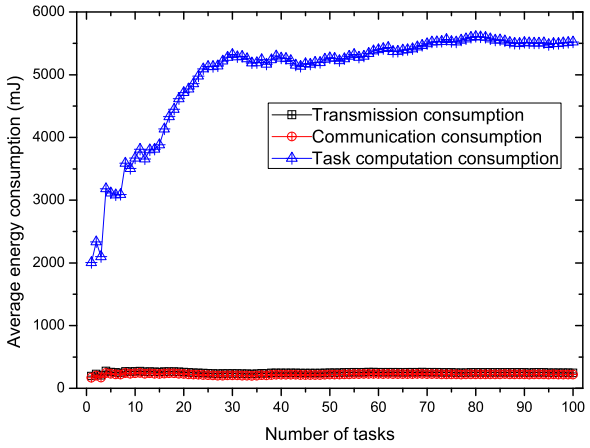


Fig. 4. Average energy consumption of tasks executed by MEC servers.

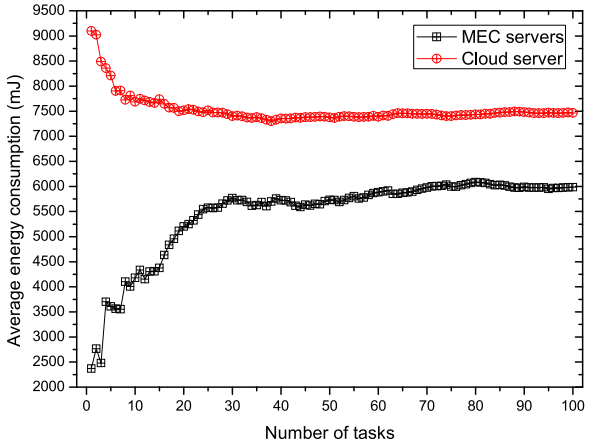


Fig. 5. Average energy consumption of MEC servers and the cloud server.

intranetwork caching mechanism, so the amount of offloading data is also relatively small, and the communication consumption of a single task is within 1000 mJ. The most serious energy consumption is the computational consumption, which increases slightly with the increase of the number of tasks.

Fig. 5 is the average energy consumption of MEC servers and the cloud server. As can be seen from the figure, the average energy consumption of MEC servers is 3500–6000 mJ, and the average energy consumption of the cloud server is 7500–8500 mJ. The total energy consumption of MEC servers is 20% smaller than the cloud server.

Fig. 6 is the number of tasks performed by MEC servers and the cloud server. In five iterations, the total number of tasks processed by MEC servers increased significantly compared with the cloud server. Observing the changes in a single iteration, the number of tasks of MEC servers is lower than that of the cloud server in the first cycle. As the number of iterations increases, the number of tasks processed by MEC servers exceeds the cloud server. Fig. 7 is the average number of tasks for each MEC server in five iterations. It can be seen that the number of tasks offloaded to each MEC server is relatively balanced, which shows that the task offloading of this article considers the load balancing of MEC servers.

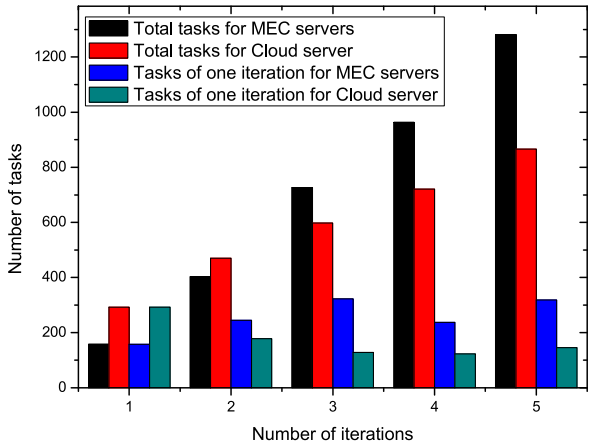


Fig. 6. Number of tasks executed by MEC servers and the cloud server.

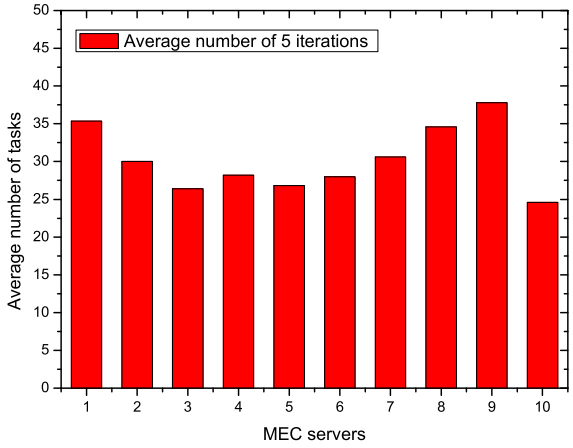


Fig. 7. Average number of tasks executed by each MEC server.

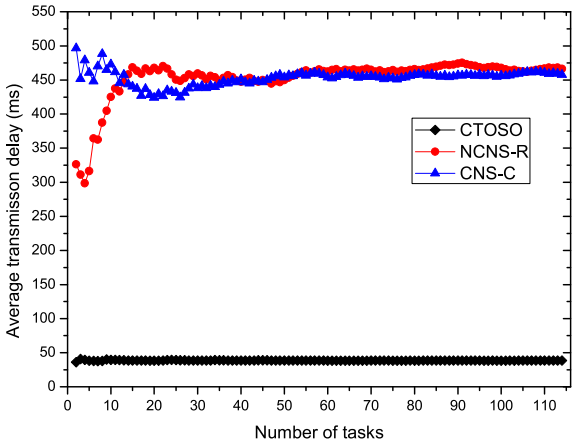


Fig. 8. Comparison of average transmission delay.

C. Comparison of Experimental Results in Delay

The delays of MEC servers of CTOSO, NCNS-R, and CNS-C schemes are compared in this section. It can be seen from Fig. 8 that the CTOSO scheme has a low transmission delay compared to NCNS-R and CNS-C schemes. This is because under the CTOSO scheme, data are processed into services by the MEC server, greatly reducing the amount of data transferred, thereby reducing the transmission delay,

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

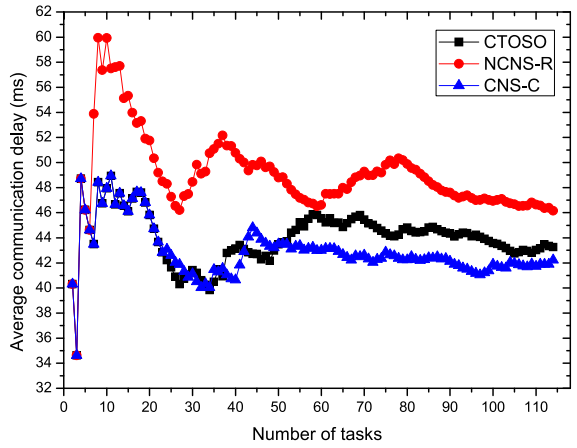


Fig. 9. Comparison of average communication delay.

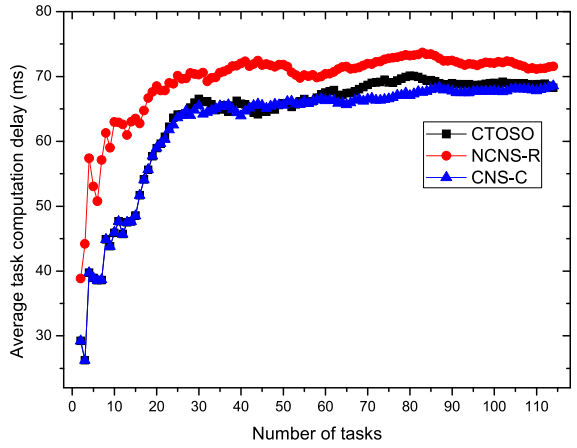


Fig. 10. Comparison of average task computation delay.

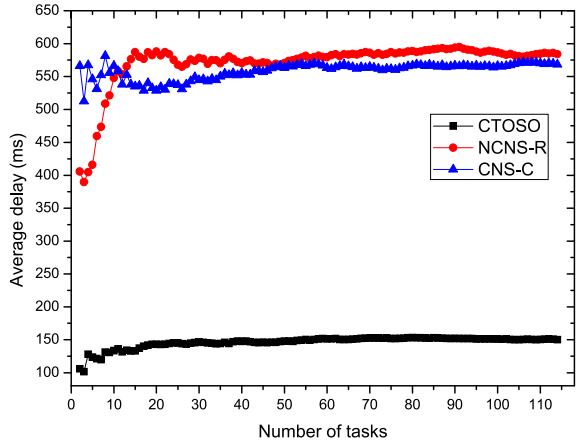


Fig. 11. Average delay of CTOSO, NCNS-R, and CNS-C when tasks are executed by MEC servers.

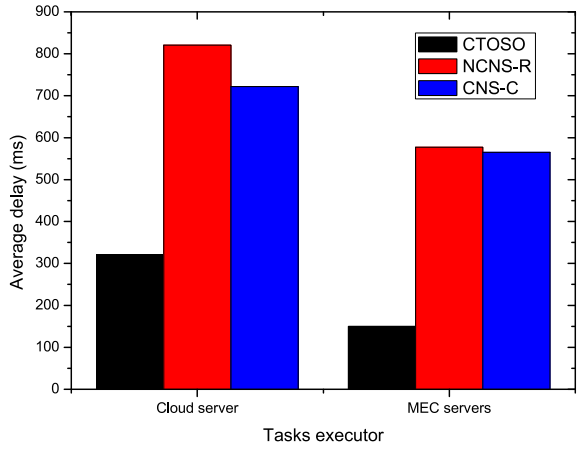


Fig. 12. Comparison of average delay of CTOSO, NCNS-R, and CNS-C scheme.

and its transmission delay is only 10%–15% of NCNS-R and CNS-C schemes.

Fig. 9 is the average communication delay under CTOSO, NCNS-R, and CNS-C schemes. The communication delay is mainly related to the amount of resources that need to be offloaded. Because NCNS-R does not adopt caching mechanism, it is necessary to offload all task-related data resources to MEC servers. While under CTOSO and CNS-C schemes, because MEC servers caches some task data locally, only part resources need to offloaded, so the communication delay of these two schemes is significantly lower than that of NCNS-R scheme. In addition, since the CNS-C scheme selects the offloaded MEC server terminal based on the maximum cache, it has the smallest communication delay, followed by the CTOSO scheme.

Fig. 10 is the average task computation delay under the CTOSO, NCNS-R, and CNS-C schemes. The computation delay is mainly related to the task data amount and the computing power of the MEC server. Because the NCNS-R adopts a random decision-making mechanism, the terminal computing power of its selected MEC server may not be optimized, so its latency is slightly larger than that of CTOSO and CNS-C scheme, but overall, the three schemes have little difference in computation delay.

Figs. 11 and 12 give the comparison of delay under CTOSO, NCNS-R, and CNS-C schemes. Regardless of whether tasks are offloaded to the cloud server or MEC servers, the CTOSO scheme has a significantly lower latency than the NCNS-R and CNS-C schemes. This is mainly because the CTOSO scheme introduces the service orchestration mechanism that reduces the transmission load by an order of magnitude. At the same time, compared with the NCNS-R and CNS-C schemes, the CTOSO scheme selects the MEC server with the lowest weighted sum of energy consumption and delay, so the performance is better in terms of delay. When tasks are executed by MEC servers, the delay of the CTOSO scheme is 74.34% lower than that of NCNS-R scheme and 73.82% lower than that of the CNS-C scheme.

D. Comparison of Experimental Results in Energy Consumption

Figs. 13–17 show the transmission consumption, communication consumption, and task computation consumption of the CTOSO, NCNS-R, and CNS-C schemes when tasks are executed by MEC servers. As can be seen from Fig. 13, the transmission energy consumption of NCNS-R is the most serious, followed by the CNS-C scheme, and the energy

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

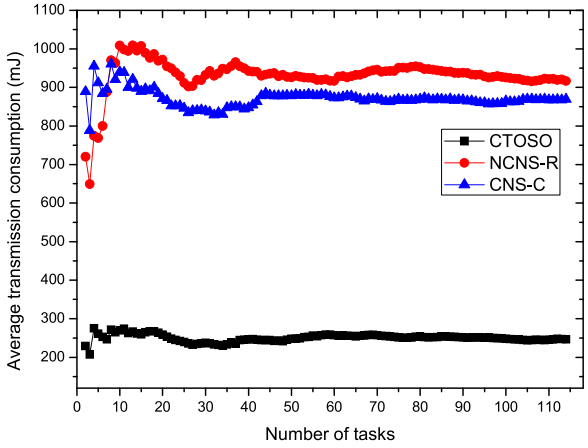


Fig. 13. Comparison of average transmission consumption.

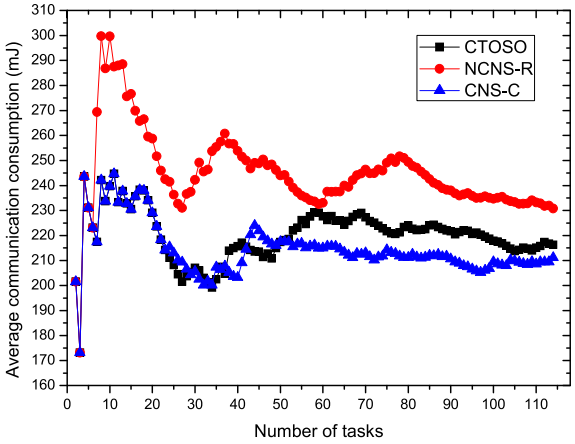


Fig. 14. Comparison of average communication consumption.

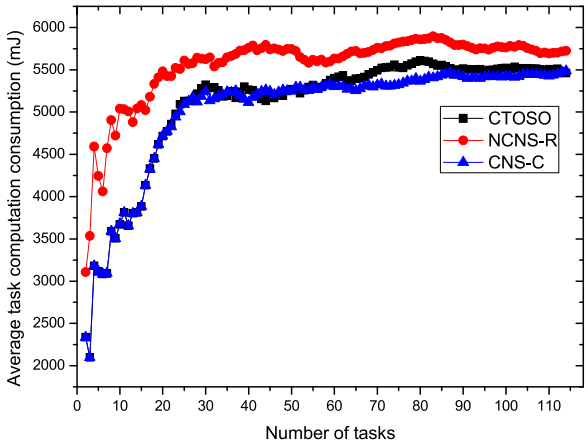


Fig. 15. Comparison of average task computation consumption.

consumption under the CTOSO scheme is only one-third of them. As can be seen from Fig. 14, considering data caching in the task offloading can effectively reduce the communication energy consumption, CNS-C has the lowest communication consumption, followed by the CTOSO scheme, and the consumption of NCNS-R scheme is higher than the other two schemes. Fig. 15 is the task computation energy consumption, and there is little difference between the three schemes, the

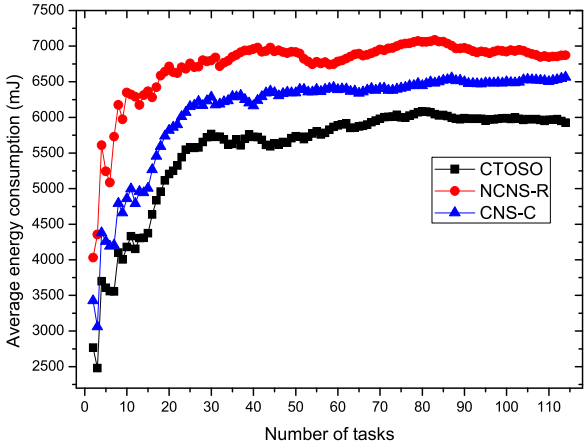


Fig. 16. Average energy consumption of CTOSO, NCNS-R, and CNS-C when tasks are executed by MEC servers.

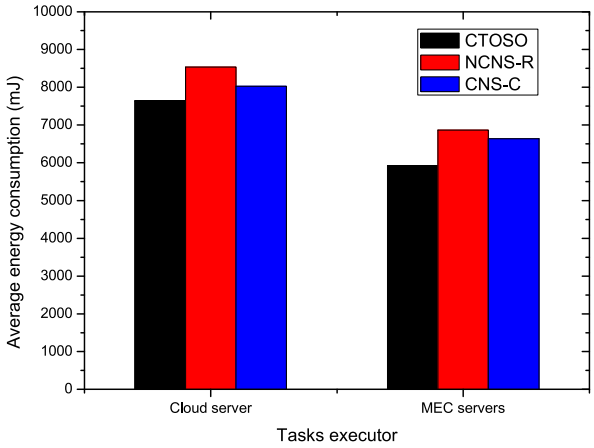


Fig. 17. Comparison of average energy consumption of CTOSO, NCNS-R, and CNS-C scheme.

NCNS-R scheme is slightly higher than the other two schemes. Comparing these three delays, it can be found that with the increase of the number of execution tasks, the transmission energy consumption shows a relatively stable trend, and the communication energy consumption is affected by the uncertainty of the offloaded data volume, the fluctuation is obvious. Finally, the computation energy consumption begins to show a growing trend with the increase of the number of tasks, and gradually stabilized.

Figs. 16 and 17 compare the energy consumption of the CTOSO, NCNS-R, and CNS-C schemes when tasks are executed by MEC servers. First, the average energy consumption increases slowly with the increase of the number of tasks. The energy consumption difference is not serious, but the consumption of the CTOSO scheme is smaller than the other two schemes. When tasks are offloaded to MEC servers, compared with the NCNS-R scheme, the CTOSO scheme reduces the energy consumption by 13.72%. Compared with the CNS-C scheme, the CTOSO scheme reduces the energy consumption by 10.71%.

VI. Conclusion

In this article, we constructed a cloud–MEC collaborative computing platform, and proposed a CTOSO scheme under

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

this platform to meet the low-latency, high-complexity, and high-reliability requirements of emerging applications. First, we innovatively introduce a service orchestration mechanism to achieving orchestrating data as high-quality services at the edge layer with software-defined networking technologies, thereby greatly reducing the network transmission load. At the same time, the CTOSO scheme established a target optimization function based on communication energy consumption, computation energy consumption, and task delay models, and pre-estimated the task offloading cost based on the optimization function, and adopted a differentiated offloading decision based on the estimation result and the task attributes. The experimental results demonstrated that the CTOSO scheme proposed in this article plays an important role in improving the delay and energy performance for big data-based applications.

REFERENCES

[1] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, J. J. Rodrigues, and M. Guizani, "Edge computing in the industrial Internet of Things environment: Software-defined-networks-based edge-cloud interplay," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 44–51, Feb. 2018.

[2] X. Liu, J. Yin, S. Zhang, B. Ding, S. Guo, and K. Wang, "Range-based localization for sparse 3-D sensor networks," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 753–764, Feb. 2019.

[3] Y. Yuan, W. Liu, T. Wang, Q. Deng, A. Liu, and H. Song, "Compressive sensing-based clustering joint annular routing data gathering scheme for wireless sensor networks," *IEEE Access*, vol. 7, pp. 114639–114658, 2019.

[4] G. Zhang, T. Wang, G. Wang, A. Liu, and W. Jia, "Detection of hidden data attacks combined fog computing and trust evaluation method in sensor-cloud system," *Concurrency Comput. Pract. Exp.*, vol. 2018, pp. 1–13, Dec. 2018, doi: 10.1002/cpe.5109.

[5] T. Wang, H. Luo, W. Jia, A. Liu, and M. Xie, "MTES: An intelligent trust evaluation scheme in sensor-cloud enabled industrial Internet of Things," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2019.2930286.

[6] T. Wang, D. Zhao, S. Cai, W. Jia, and A. Liu, "Bidirectional prediction based underwater data collection protocol for end-edge-cloud orchestrated system," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2019.2940745.

[7] Q. Li, A. Liu, T. Wang, M. Xie, and N. N. Xiong, "Pipeline slot based fast rerouting scheme for delay optimization in duty cycle based M2M communications," *Peer-to-Peer Netw. Appl.*, vol. 12, no. 6, pp. 1673–1704, 2019, doi: 10.1007/s12083-019-00753-z.

[8] A. Alsharif, M. Nabil, A. Sherif, M. Mahmoud, and M. Song, "MDMS: Efficient and privacy-preserving multi-dimension and multi-subset data collection for AMI networks," *IEEE Internet Things J.*, to be published, doi: 10.1109/JIOT.2019.2938776.

[9] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6774–6785, Aug. 2019.

[10] L. Zhang, T. Han, and N. Ansari, "Energy-aware virtual machine management in inter-datacenter networks over elastic optical infrastructure," *IEEE Trans. Green Commun. Netw.*, vol. 2, no. 1, pp. 305–315, Mar. 2017.

[11] T. Wang, H. Ke, X. Zheng, K. Wang, A. Sangaiah, and A. Liu, "Big data cleaning based on mobile edge computing in industrial sensor-cloud," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2019.2938861.

[12] X. Liu *et al.*, "Adaptive data and verified message disjoint security routing for gathering big data in energy harvesting networks," *J. Parallel Distrib. Comput.*, vol. 135, pp. 140–155, Jan. 2020, doi: 10.1016/j.jpdc.2019.08.012.

[13] O. Osanaiye, S. Chen, Z. Yan, R. Lu, K.-K. R. Choo, and M. Dlodlo, "From cloud to fog computing: A review and a conceptual live VM migration framework," *IEEE Access*, vol. 5, pp. 8284–8300, 2017.

[14] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[15] X. Liu, Q. Yang, J. Luo, B. Ding, and S. Zhang, "An energy-aware offloading framework for edge-augmented mobile RFID systems," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 3994–4004, Jun. 2019.

[16] X. Liu, M. Zhao, A. Liu, and K. K. L. Wong, "Adjusting forwarder nodes and duty cycle using packet aggregation routing for body sensor networks," *Inf. Fusion*, vol. 53, pp. 183–195, Jan. 2020.

[17] M. Huang *et al.*, "A game-based economic model for price decision making in cyber-physical-social systems," *IEEE Access*, vol. 7, pp. 111559–111576, 2019.

[18] L. Hu, A. Liu, M. Xie, and T. Wang, "UAVs joint vehicles as data mules for fast codes dissemination for edge networking in smart city," *Peer-to-Peer Netw. Appl.*, vol. 12, no. 6, pp. 1550–1574, 2019, doi: 10.1007/s12083-019-00752-0.

[19] Y. Liu, A. Liu, X. Liu, and M. Ma, "A trust-based active detection for cyber-physical security in industrial environments," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2019.2931394.

[20] J. Wu, S. Guo, J. Li, and D. Zeng, "Big data meet green challenges: Big data toward green applications," *IEEE Syst. J.*, vol. 10, no. 3, pp. 888–900, Sep. 2016.

[21] X. Liu, Y. Liu, H. Song, and A. Liu, "Big data orchestration as a service network," *IEEE Commun. Mag.*, vol. 55, no. 9, pp. 94–101, Sep. 2017.

[22] M. Huang *et al.*, "A services routing based caching scheme for cloud assisted CRNs," *IEEE Access*, vol. 6, pp. 15787–15805, 2019.

[23] D. Liu, L. Khoukhi, and A. Hafid, "Decentralized data offloading for mobile cloud computing based on game theory," in *Proc. Int. Conf. Fog Mobile Edge Comput. (FMEC)*, 2017, pp. 20–24.

[24] M.-A. Messous, S.-M. Senouci, H. Sedjelmaci, and S. Cherkaoui, "A game theory based efficient computation offloading in an UAV network," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4964–4974, May 2019.

[25] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.

[26] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4804–4814, Jun. 2019.

[27] S. Ahn, J. Lee, S. Park, S. H. S. Newaz, and J. K.Choi, "Competitive partial computation offloading for maximizing energy efficiency in mobile cloud computing," *IEEE Access*, vol. 6, pp. 899–912, 2017.

[28] L. Liu, Z. Chang, X. Guo, and T. Ristaniemi, "Multi-objective optimization for computation offloading in mobile-edge computing," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, 2017, pp. 832–837.

[29] I. A. Ridhawi, M. Aloqaily, Y. Kotb, Y. A. Ridhawi, and Y. Jararweh, "A collaborative mobile edge computing and user solution for service composition in 5G systems," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 11, 2018, Art. no. e3446.

[30] H. Guo and J. Liu, "Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, May 2018.

[31] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee," *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1594–1608, Apr. 2018.

[32] J. Zhang *et al.*, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2633–2645, Aug. 2018.

Mingfeng Huang is currently pursuing the Doctoral degree with the School of Computer Science and Engineering, Central South University, Changsha, China.

Her research interests include edge computing, services-based networks, crowd sensing networks, and wireless sensor networks.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.

Wei Liu received the Ph.D. degree in computer application technology from Central South University, Changsha, China, in 2014.

He is an Associate Professor and a Senior Engineer with the School of Informatics, Hunan University of Chinese Medicine, Changsha. His research interests include software engineering, data mining and medical informatics. He has published more than 20 papers in related fields.

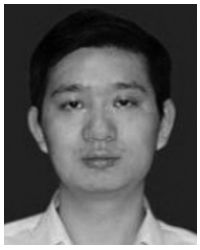
Tian Wang received the B.Sc. and M.Sc. degrees in computer science from the Central South University, Changsha, China, in 2004 and 2007, respectively, and the Ph.D. degree from the City University of Hong Kong, Hong Kong, in 2011.

He is currently an Associate Professor with the National Huaqiao University of China, Xiamen, China. His research interests include wireless sensor networks, social networks, and mobile computing.

Anfeng Liu received the M.Sc. and Ph.D. degrees in computer science from Central South University, Changsha, China, 2002 and 2005, respectively.

He is a Professor with the School of Information Science and Engineering, Central South University, China. His major research interest is wireless sensor network.

Prof. Liu is also a member (E200012141 M) of the China Computer Federation.



Shigeng Zhang (M'12) received the B.Sc., M.Sc., and D.Eng. degrees in computer science from Nanjing University, Nanjing, China, in 2004, 2007, and 2010, respectively.

He is currently an Associate Professor with the School of Computer Science and Engineering, Central South University, Changsha, China. He has authored or coauthored over 70 technique papers in top international journals and conferences, including Infocom, ICNP, the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the *ACM Transactions on Sensor Networks*, and the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS. His current research interests include Internet of Things, mobile computing, RFID systems, and IoT security.

Dr. Zhang is on the Editorial Board of the *International Journal of Distributed Sensor Networks*. He was a Program Committee Member of many international conferences, including ICC, ICPADS, MASS, UIC, and ISPA. He is a member of ACM.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on March 21,2022 at 12:54:57 UTC from IEEE Xplore. Restrictions apply.