Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.037483

ARTICLE

A New Partial Task Offloading Method in a Cooperation Mode under Multi-Constraints for Multi-UE

Shengyao Sun[1,2], Ying Du[3], Jiajun Chen[4], Xuan Zhang[5], Jiwei Zhang[6,*] and Yiyi Xu[7]

[1]School of Information Science and Technology, Zhengzhou Normal University, Zhengzhou, 450044, China

[2]Henan Key Laboratory of Big Data Analysis and Processing, Henan University, Kaifeng, 475004, China

[3]School of Geography and Tourism, Zhengzhou Normal University, Zhengzhou, 450044, China

[4]Science and Engineering College, South China University of Technology, Guangzhou, 510641, China

[5]Department of Electrical and Electronic Engineering, Luohe Vocational Technology College, Luohe, 462002, China

[6]School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, 100876, China

[7]Cardiff School of Engineering, Cardiff University, Cardiff, CF10 3XQ 15, UK

*Corresponding Author: Jiwei Zhang. Email: jwzhang666@bupt.edu.cn

Received: 05 November 2022 Accepted: 07 April 2023 Published: 08 October 2023

ABSTRACT

In Multi-access Edge Computing (MEC), to deal with multiple user equipment (UE)'s task offloading problem of parallel relationships under the multi-constraints, this paper proposes a cooperation partial task offloading method (named CPMM), aiming to reduce UE's energy and computation consumption, while meeting the task completion delay as much as possible. CPMM first studies the task offloading of single-UE and then considers the task offloading of multi-UE based on single-UE task offloading. CPMM uses the critical path algorithm to divide the modules into key and non-key modules. According to some constraints of UE-self when offloading tasks, it gives priority to non-key modules for offloading and uses the evaluation decision method to select some appropriate key modules for offloading. Based on fully considering the competition between multiple UEs for communication resources and MEC service resources, CPMM uses the weighted queuing method to alleviate the competition for communication resources and uses the branch decision algorithm to determine the location of module offloading by BS according to the MEC servers' resources. It achieves its goal by selecting reasonable modules to offload and using the cooperation of UE, MEC, and Cloud Center to determine the execution location of the modules. Extensive experiments demonstrate that CPMM obtains superior performances in task computation consumption reducing around 6% on average, task completion delay reducing around 5% on average, and better task execution success rate than other similar methods.

KEYWORDS

MEC; partial task offloading; parallel dependencies; completion delay

1 Introduction

In recent years, mobile user equipment (UE), represented by smartphones and tablets, has gradually become an essential part of people's daily lives [1–4]. UE, with its built-in camera, microphone, stereo, and a large variety of sensors, can provide users with social, business, information entertainment, games, and other services, and plays a vital role in people's learning, entertainment, social, travel and other aspects of the increasingly [1–4]. The rapid growth in the number of UEs also makes it possible to expand data-based applications. At present, the number of UE applications is increasing. According to the latest data released by Sensor Tower, in the first quarter of 2022, the global App Store downloads reached 8.6 billion times. The total number of app downloads between the App Store and Google Play Store was 36.9 billion [5].

Although these applications have brought various conveniences to people's lives, due to the natural limitations of UE, such as its computing power, battery power, storage, etc., they cannot handle computation-intensive, energy-intensive, and other applications that require high capability of equipment (such as virtual reality, augmented reality, face recognition, etc.) and seriously affect the service experience of users [1–4]. MEC can effectively address this problem. It considers that computing is no longer confined to the cloud (cloud side) and the client (device side), but can occur on any device during data transmission and is closer to the data [1–4]. When UE's task is running, the task uses the pre-set task offloading method to offload another execution location to reduce UE's computational complexity and energy consumption.

Nowadays, numerous task-offloading strategies have been proposed [6–25]. Much research shows that task offloading needs are offloading time and the location of task processing. Offloading location refers to selecting the appropriate task terminal. The offloading time refers to triggering a pre-set condition when UE starts offloading all (or part) of its tasks to the correct location for processing. Among them, partial offloading thinks the task can be split into several modules. According to the dependence relationship among modules and the consumption of UE resources by modules, some modules are selected to be offloaded to the appropriate MEC server for processing [1,3,4].

Because of the simplicity of describing the modules that make up a task using a serial relationship, most current partial task offloading methods assume serial dependencies among modules when studying task offloading [10–16]. However, in practical applications, modules can present both serial and parallel dependencies, such as face-detection tasks [3]. It is not reasonable to deal with parallel tasks using serially. Meanwhile, traditional methods typically offload tasks to MEC servers, with less involvement of other devices in the network, such as the servers of Mobile Cloud Center (MCC). Compared to MCC's servers, MEC has far fewer resources [1,2,4,11]. MCC taking part in offloading can effectively reduce the service pressure of MEC and can balance the load of MEC servers. However, most methods usually consider that MCC is far from UE, and MCC is less considered to participate in task offloading. In addition, when task offloading, most methods assume that there are sufficient resources to perform the task, and less consideration is given to the impact of resource constraints when offloading. This may lead to the problem that although many tasks are offloaded, the processing latency is increasing, and may also lead to the failure of task offloading. For example, during a unit of time, the communication channel of the Base station (BS) is fixed. Multi-UE offload tasks may lead to many requirements for the communication channel or even much more than the amount of traffic that BS can handle. Then, some modules will be waiting for the channel. This will increase the tasks' processing delay. If a task fails to get the channel all the time, it will lead to the failure of task offloading.

To deal with the task offloading problem caused by insufficient resources of multi-UEs, and the modules that make up the task present parallel dependencies, this paper proposed a new Cooperation Partial task offloading method under Multi-constraints for Multi-UE (named CPMM). CPMM adopts a cooperation manner to offload the task (Rather than simply offloading tasks to a specific server, multiple servers cooperate to offload some tasks from one server to other servers to improve the system's resource utilization. This offloading mode is called cooperation mode), and considers many constraints, aiming to reduce multi-UE's computation and energy consumption while meeting the completion delay as much as possible.

In summary, the main contributions of this paper are summarized as follows:

- This paper studies the parallel modules offloading of multiple UEs under multiple constraints and proposes a partial offloading method in a cooperative mode.
- To better study the task offloading of multi-UE under multi-constraints, CPMM first studied the task offloading of single-UE. Then, it studies the multi-UE task offloading under multi-constraints according to the method of single-UE task offloading.
- CPMM uses the critical path algorithm to divide the modules into key and non-key modules. It proposes the non-key module offloading method and key module offloading method according to the multi-constraints. It uses different offloading methods to offload the modules according to their type.
- To ensure the task's completion time, CPMM prioritizes non-key modules for offloading according to multi-constraints and prioritizes key modules to obtain offloading resources when competing for offloading resources.

The rest of the paper is structured as follows: Section 2 presents the related works. Section 3 elaborates on the overview of CPMM. Section 4 presents single-UE task offloading with theoretical and analysis. Section 5 discusses multi-UE task offloading with theoretical analysis under the multi-constraints. In Section 6, extensive experiments show that CPMM obtains better performances than similar approaches within a variety of metrics, and analyses of various factors are conducted.

2 Related Work

2.1 The Task Offloading under MEC

Task offloading has been successfully applied in many fields, such as MEC, vehicle networks, the Internet of Things (IoT), Artificial Intelligence (AI), Geographic Information Systems (GIS), etc. [1–4]. Many task-offloading methods for mobile edge computing have been proposed [6–25]. These methods can be classified from the following three aspects.

Based on task offloading scale: According to the scale of task offloading, the task-offloading methods can be divided into executing locally, completely offloading, and partially offloading. Liu et al. proposed a complete task offloading method based on a one-dimensional search algorithm [6]. This method finds the optimal offloading scheme according to the queue state of the application buffer, the available computing power at the UE and MEC servers, and the channel characteristics between the UE and MEC servers. LODCO is a dynamic complete task offloading method, aiming to optimize application execution delay [7]. It assumes that UE uses energy collection technology to minimize energy consumption during local execution, and uses battery power control method to optimize energy consumption for data transmission. Paper [10] used partial data to audit the integrity of edge server cache data. It analyzes the threat model and the audit objectives, then proposes a lightweight sampling-based probabilistic approach, namely EDI-V, to help app vendors audit the integrity of their data cached on a large scale of edge servers. Paper [11] proposed a MEC service pricing scheme to coordinate with the service caching decisions and control wireless devices' task offloading behavior in a cellular network. It proposes a two-stage dynamic game of incomplete information to model and analyzes the two-stage interaction between the BS and multiple associated wireless devices.

Based on the dependency relationship of modules: During the task offloading, if a task can be split into many modules, the relationships presented by the modules that make up the task can be serial or parallel [1,3]. In a serial dependency relationship, the execution of the latter task (or module) must wait for the result of the previous task (or module). A parallel relationship is one in which the tasks (or modules) offloaded to a remote are offloaded and processed concurrently. The relationship between the tasks (or modules) can be expressed in a task (or module) dependency graph [1]. It needs to analyze them according to the specific characteristics of the APP/program when analyzing the dependencies between tasks (or modules). Dependencies relationships between tasks are not absolute or fixed and can be changed according to different criteria [3].

Since using the serial relationship to describe the dependencies between tasks (or modules) is relatively simple, many partial tasks offloading method deal with the relationship between modules according to the serial relationship. Paper [16] considered the cooperation of cloud computing and MEC in IoT. It starts with the single-user computation offloading problem. Then it considers the multiuser computation offloading problem formulated as a mixed integer linear programming problem by considering resource competition among mobile users. It designs an iterative heuristic MEC resource allocation algorithm to make the offloading decision dynamically. Paper [17] considered a practical application consisting of a set of tasks and models it as a generic graph topology. Then, the energy-efficient task offloading problem is mathematically formulated as a constrained 0–1 programming.

Based on the manner of offloading execution: Currently, most offloading strictly follows the definition of MEC. The offloading task is only allowed in the MEC server cluster relying on BS, and less consideration is given to other service resources to participate in the task offloading, such as remote MCC servers and other adjacent BS's MEC servers [1–4]. This manner can effectively ensure that the offloading task has a small data transmission delay. However, it is easy to cause an overload of MEC service in the local area when the communication range of BS has a great offload demand, which leads to the imbalance of MEC.

In cooperative methods, they usually think that the resources to execute the offloading task are limited, and multi-constraints limit the task offloading. The offloading location is no longer limited to the MEC servers, and the task can also be offloaded to the MCC server or other MEC servers relying on other BS.

Reducing the service pressure of MEC servers through multi-terminal cooperation. Paper [14] extended the task offloading scenario to multiple cloud servers, aiming to obtain the optimal computation distribution among cloud servers in closed form for the energy consumption of minimization and latency of application execution minimization problems. Paper [16] considered multi-constraints when task offloading, designs an iterative heuristic MEC resource allocation algorithm to make the offloading decision dynamically, and accomplishes the task offloading through the cooperation of MEC resources and MCC resources.

2.2 MEC Architecture

European Telecommunications Standards Institute (ETSI) has proposed some generic reference architecture [1,20,21]. The framework can be divided into the MEC system, MEC server management, and network layer. Although ETSI has proposed some reference architecture, no concrete standardized architectural framework exists for MEC. Therefore, many researchers first need to define the system model of task offloading when they study task offloading [4,10–16]. Different researchers have different definitions of the MEC system model, which can be summarized into the following three system model structures: (1) Single MEC server in the single Base station (BS): This system model includes only one MEC server and only one BS. MEC servers are attached to BS and provide services to UEs within the communication range of BS. (2) Multiple MEC sever with single BS: This system model exists with multiple MEC servers and one BS. These multi-MEC servers are attached to BS. (3) Distributed MEC servers with multiple BS: In this system model, it includes multiple BS. Each BS is attached to one or multiple MEC servers. UE offloads its task to the nearest BS. BS can distribute assignments to its attached MEC servers or route them to other MEC servers running to other BSs.

3 The Overview of CPMM

CPMM is a partial task offloading method for multi-UE under multi-constraints MEC. It focuses on the system model as multi-MEC servers with single-BS and the parallel dependencies of the modules that make up the task. It aims to reduce energy and computation consumption while meeting the task completion delay as much as possible. It is divided into offloading under the single UE and offloading under multi-UE. The first part mainly describes how single-UE offloads tasks under multi-constraints. The last part mainly describes how multi-UE offloads tasks under constraints according to the single UE offloading. CPMM begins to select some modules of UE to offload according to the multi-constraints concerned when the UE starts to execute the task. Fig. 1 shows its process and constraint conditions.

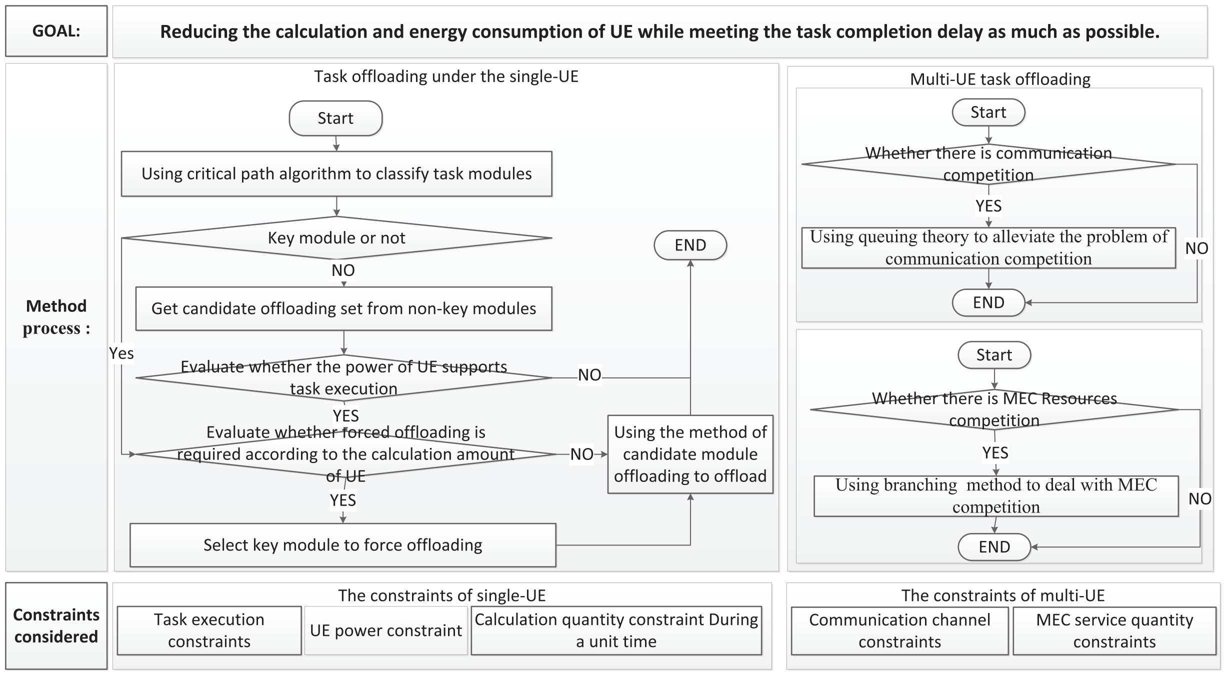

GOAL:
Reducing the calculation and energy consumption of UE while meeting the task completion delay as much as possible.
Method process :
Task offloading under the single-UE
Start
Using critical path algorithm to classify task modules
Key module or not
NO
Get candidate offloading set from non-key modules
Yes
Evaluate whether the power of UE supports task execution
NO
YES
Evaluate whether forced offloading is required according to the calculation amount of UE
NO
YES
Select key module to force offloading
Using the method of candidate module offloading to offload
END
Multi-UE task offloading
Start
Whether there is communication competition
YES
Using queuing theory to alleviate the problem of communication competition
NO
END
Start
Whether there is MEC Resources competition
YES
Using branching method to deal with MEC competition
NO
END
Constraints considered
The constraints of single-UE
Task execution constraints
UE power constraint
Calculation quantity constraint During a unit time
The constraints of multi-UE
Communication channel constraints
MEC service quantity constraints

Figure 1: The process and the constraint conditions of CPMM

When the single UE offloading, CPMM uses the critical path algorithm to divide the modules into key and non-key modules. And then, the offloading module is preferentially selected from the non-key module set as the candidate offloading module set. After getting the set, CPMM evaluates the battery power of UE to determine whether the current battery power can support the normal execution of tasks. If the battery power is sufficient, it evaluates the current amount of computation to determine whether a task must be offloaded due to resource constraints. The offloading that must be offloaded due to insufficient resources is called forced offloading. The offloading under the condition of sufficient resources is called active offloading. When the multi-constraint conditions considered by CPMM can meet the normal execution of the task, it uses active offloading to offload some non-key modules. Suppose it cannot be satisfied (e.g., even though some non-key modules have been offloaded, the current UE computing resources still cannot meet the normal execution of the task), CPMM will consider whether it is necessary to offload some key modules according to the multi-constraints. Then, it selects the candidate module set from the key modules in a successive verification manner, waiting for offloading.

Since multi-UE task offloading can be regarded as multiple single-UE tasks offloading without external resource competition, CPMM proposes the offloading method of key and non-key modules on the premise of assuming there are enough communication and MEC server resources. According to the offloading method of single-UE key and non-key modules, CPMM discusses the task offloading method of multi-UE with external resource competition. It adopts the weighted queuing method to alleviate the problem of multi-UE competing for BS's communication resources and uses the branching method to divide the task offloading of multi-UE into many different cases according to the constraints of BS's communication and MEC servers. Finally, it discusses the location of task module offloading and execution according to the candidate offloading module and the case of the resources at a time interval.

4 The Task Offloading of Single-UE under Multi-Constraints

4.1 The Constrain of Task Offloading for Single-UE

Task offloading is a complex process that is affected and constrained by many factors, such as user preferences, network link quality, mobile device performance, BS performance, etc. [1]. CPMM mainly focuses on the following aspects:

- Constraint 1 (UE's battery power constraint): The battery power of UE must be able to support the standard task processing.
- Constraint 2 (Task execution constraint): All modules must be executed to ensure the normal execution of the task.
- Constraint 3 (Calculation constraint during a unit of time): The CPU computing capacity of UE needs to meet the computing required for task execution during a unit of time.
- Constraint 4 (Communication channel constraint): When multi-UE offload candidate modules concurrently, it is constrained by the number of available communication channels per unit of time.
- Constraint 5 (MEC server resource constraint): When multi-UE concurrently selects MEC servers to perform the service, it is constrained by the number of services acceptable to the MEC servers.

Constraint 1 is a prerequisite for task execution. Constraint 2 ensures that user-submitted tasks execute smoothly and with correct results. Constraints 1 and 3 determine why tasks are offloaded, i.e., whether the UE actively offloads modules to reduce energy and computational consumption or is forced to offload due to insufficient UE resources. Constraints 4 and 5 determine that communication competition and MEC resource competition should be paid attention to when the task is offloading.

4.2 The Method of Task Module Classification

This section mainly describes which modules under the parallel relationship belong to modules with lower impact. Using $UE_i = \{cap_{i,m}, pow_i, PC_{i,com}, PI_{i,send}\}$ to represent UE. $cap_{i,m}$ indicates the calculation frequency of the CPU during a unit of time (e.g., 1 s). The current battery power is pow_i. CPU calculation power is$PC_{i,com}$. The sent power of the network card is $PI_{i,send}$. Using $task_i'$ to denote UE_i's a pre-performed task. Assuming that it can be split into $m + 2$ modules. These modules are a parallel dependency. The initial module is $Ta_{i,0}$, which means the module is initiated by the user locally. The last output module is $Ta_{i,m+1}$, which indicates that the final task execution results are assembled locally after execution at different execution terminals. Using $Ta_{i,k}$ represents the kth module, and its execution position is either local, MEC servers, or MCC servers according to CPMM focuses system model. Its execution position can be expressed as:

$$\alpha = \begin{cases} 1, local \\ 0, other \end{cases} \tag{1}$$

The task processing process can be split into several time intervals, using $\{T_0, T_1, T_2, \ldots, T_{i-1}, T_i, \ldots\}$ to denote and form a sequential relationship. Any two-timing intervals are expressed as $\Delta T_j (\Delta T_j = T_i - T_{i-1})$. These time intervals are the same. So, the models of $task_i'$ can be shown the Fig. 2 according to the possible processing position and the dependencies among modules.

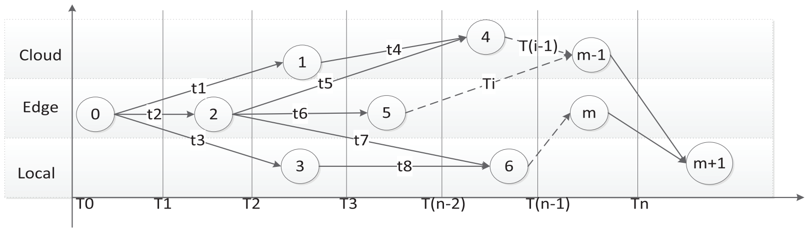


Figure 2: The time series model of single-UE task offloading

Based on the relevant theory of critical path in graph theory [26], the critical path determines the latest project completion time and can obtain the latest project completion time by this algorithm. CPMM uses the directed weighted graph to describe the relationship between modules too. So, it can also use the critical path algorithm to get the latest completion time of the task. According to the critical path algorithm, in the directed weighted graph, if the delay of a module on the critical path increases, the task completion time also increases. That is, the module completion delay on the critical path has a greater impact on the task completion delay. In other words, CPMM can use the critical path algorithm to get which modules have less impact on delay.

Setting $Ta_{i,k} = \left\{Data_{i,k}, BI_{i,k}, IsKey_{i,k}\right\}$. $Data_{i,k}$ is the size of the data to be processed. $BI_{i,k}$ is the module attribute. If $BI_{i,k} = 1$, it means that $Ta_{i,k}$ must be processed locally. $IsKey_{i,k}$ indicates whether a module is a key module. If a module is on a critical path, it is called the key module by CPMM, and set $IsKey_{i,k} = 1$. Else, it is called the non-key module and set $IsKey_{i,k} = 0$. Using CTS_i to denote the key module set and using NCT_i to denote the non-key module set. According to the definition of critical path, CPMM can get $Ta_{i,k}$'s earliest start time ($ET_{i,k}$), latest start time ($LT_{i,k}$), and maneuvering time $MT_{i,k}$.

$$\mathrm{MT}_{i,k} = LT_{i,k} - ET_{i,k} MT_{i,k} > 0 \tag{2}$$

Maneuver time is the time difference between the time the module can complete (i.e., earliest start time) and the time the module must complete (i.e., latest start time). Based on the definition of the critical path, the maneuvering time of the non-key module must meet $MT_{i,k} \geq 0$. It means that even the delay of the late $MT_{i,k}$ start of non-key modules will not influence the overall task processing delay. In other words, selecting module offloading from CTS_i has a lower impact on the processing delay of $task_i'$. Consequently, whether active or forced offloading, the non-key module set can be chosen as the offloading module to reduce the UE's energy and computational consumption.

4.3 The Selection Method of Candidate Offloading Set

This section mainly describes the method for UE to select a candidate offloading module from non-key modules. The steps are as follows:

- Step 1: Obtain the initial candidate offloading set according to whether the module needs to be processed locally.

Since $BI_{i,k} = 1$ means the module must process locally, CPMM selects modules with $BI_{i,k} = 0$, forming the initial candidate offloading set, and denoted by NCT_i^{C1}.

- Step 2: Perform secondary candidate offloading set NCT_i^{C1} according to the required calculation amount of each module.

Since the module with high computational can consume UE's larger computational, CPMM selects the module with high computational from NCT_i^{C1} to form a new candidate set.

To describe which module belongs to computationally intensive, CPMM defines a calculate threshold, demoted δ ($\delta \geq 2$). When the calculation of $Ta_{i,k}$ satisfies the inequality (3) during a unit of time, it means that $Ta_{i,k}$ with high calculation.

$$UC_{i,k} \geq \delta \times AVca_i \tag{3}$$

In which, $AVca_i$ represents the average amount of computation during a unit of time for all tasks performed by UE_i, $AVca_i$ can be obtained from the formula (4).

$$AVca_i = \sum_{i=1}^{count} \left(\frac{Tca_{i,all}}{(m+2)} \right) \Big/ count = \sum_{i=1}^{count} \left(\sum_{j=0}^{m+1} \frac{C_{i,j}}{Data_{i,j}} \Big/ (m+2) \right) \Big/ count \tag{4}$$

The *count* represents the number of tasks processed by UE_i during the unit of time. According to formula (4), if a module in NCT_i^{C1} satisfy inequalities (3), it is a module with high computational and is divided into set NCT_i^{C2}.

- Step 3: Getting the final candidate offloading set according to maneuvering time.

Due to the need for data migration when offloading, according to the definition of maneuver time, if the execution delay of offloading to another location is still less than the maneuver time, the offloading of the module will not affect the task processing delay and vice versa. Therefore, NCT_i^{C2} is filtered again based on the relationship between maneuver time and the execution time of offloading to the MEC service, getting the final candidate offloading set.

The module first migrates to BS when offloading. Assuming the communication bandwidth between UE_i and BS is B, the channel adopts a Code Division Multiple Access (CDMA) cellular model with h channels, the channel transmission rate conforms to Shannon's theorem, the signal-to-noise ratio is fixed, expressed as η. Then the data transmission rate (Tr_i) between UE and BS can be expressed as:

$$Tri = \eta \times (B/h) \tag{5}$$

If $Ta_{i,k}$ needs to offload, the delay transmission ($Tt_{i,k}$) from UE to BS can be expressed as:

$$Tt_{i,k} = Data_{i,k}/Tr_i = (Data_{i,k} \times h)/(B \times \eta) \tag{6}$$

Assuming the distance between BS and MEC servers is one hop, the bandwidth is $B_{B\rightarrow E}$, $Ta_{i,k}$'s execution delay on MEC severs is $\zeta_{i,k}$, then the delay of $Ta_{i,k}$ offloading to the MEC server ($TECD_{i,k}^{MEC}$) is expressed as:

$$TECD_{i,k}^{MEC} = Tt_{i,k} + Data_{i,k}/B_{B\rightarrow E} + \xi_{i,k} \tag{7}$$

If $Ta_{i,k}$'s $MT_{i,k}$ satisfies the inequality (8), then offloading the module would increase the task processing delay. So, it cannot be offloading. Otherwise, consider offloading.

$$MTi, k > TECD_{i,k}^{MEC} \tag{8}$$

According to the inequality (8), NCT_i^{C2} has filtered again, getting the final candidate offloading set.

4.4 The Energy Evaluation Method

This section mainly describes whether the submitted tasks can be successfully executed.

After getting the candidate offloading set from non-key modules, the remaining modules are to be processed locally. The data sent to the offloading module still needs to consume the local battery energy. Battery energy is what keeps the task running. Therefore, CPMM needs to evaluate whether the UE's battery energy can support the offloading and running of these modules to determine whether the task can be executed normally.

If $Ta_{i,k}$ is processed locally, then the power waste of CPU ($EC_{i,k}$) can be expressed as:

$$EC_{i,k} = PC_{i,com} \times Tp_{i,k} = PC_{i,com} \times Data_{i,k}/C_{i,k} \tag{9}$$

If $Ta_{i,k}$ is offloaded to other locations, then transmit power consumption ($ES_{i,k}$) is expressed as:

$$ES_{i,k} = PI_{i,send} \times Tt_{i,k} = PI_{i,send} \times (Data_{i,k} \times m)/(B \times \eta) \tag{10}$$

Therefore, the local energy consumption of all modules for $task_i'$ ($EC_{i,all}$) can be expressed as:

$$EC_{i,all} = \sum_{k=0}^{m+1} [\alpha \times EC_{i,k} + (1 - \alpha) \times ES_{i,k}] \tag{11}$$

CPMM assumes when the power of UE_i satisfies inequality (16), it indicates that the battery power does not support the current task. At this time, task execution failed.

$$(pow_i + EC_{i,all})/pow_{i,ALL} \times 100\% \geq pow_{\min} \tag{12}$$

In which, $pow_{i,All}$ is the maximum battery power that UE can provide. pow_{min} is a pre-defined threshold, which is called the power threshold.

4.5 The Trigger Method of Forced Offloading

Even if some non-key modules are offloaded, the total computation amount of the remaining modules may need to be bigger to meet the needs of the remaining modules. At this point, forced offloading is required to meet Constraint 2. This section describes the reasons for forced offloading.

If $Ta_{i,k}$ is processed locally, then the completion delay $Tp_{i,k}$ is:

$$Tp_{i,k} = Data_{i,k}/cap_{i,m} \tag{13}$$

Then, during a unit of time, the calculation ($UC_{i,k}$) required by $Ta_{i,k}$ can be expressed as:

$$UC_{i,k} = 1/Tp_{i,m} = cap_{i,m}/Data_{i,k} \tag{14}$$

According to offloading the non-key candidate module set, CPMM uses formula (15) to get computational of the remaining modules during a unit of time, denoted by (Tca_i).

$$Tca_i = \sum_{k=0}^{m+1} \left(\alpha \times UC_{i,k}\right) = \sum_{k=0}^{m+1} \left(\alpha \times \frac{1}{Tp_{i,k}}\right) = \sum_{k=0}^{m+1} \left(\alpha \times \frac{cap_{i,m}}{Data_{i,k}}\right) \tag{15}$$

According to Tca_i and Constraint 3, using the following inequality (16) to judge whether the current calculation is sufficient:

$$\left(Tca_{i,all} + Tca_{i,now}\right)/cap_{i,m} \times 100\% \geq q_{i,\max} \tag{16}$$

Which, $cap_{i,now}$ is the current calculation of UE_i. $q_{i,max}$ is a pre-defined threshold called the max-tolerance calculation threshold, which represents the maximum amount of calculation UE_i can bear per unit of time. For example, $q_{i,max} = 75\%$ indicates that if the submitting task and the current computation exceeds the total computation reached 75%, the task cannot meet the calculation requirement of the current task, i.e., it means that after the non-key modules are offloaded, the calculation is still poor. The key modules must be selected to offload.

4.6 The Method of Non-Key Module Offloading

After each UE selects the candidate set, multi-UE competes for resources to complete task offloading. To provide a better reference for multi-UE under multi-constraints, CPMM discusses the single-UE task offloading under the following Sections 4.6 and 4.7. This section mainly addresses the method of key module offloading under the single UE.

The step of single-UE task offloading is as follows:

- Step 1: Using the selection method of candidate offloading set to get non-key candidate module set.
- Step 2: Sorting the set in ascending order according to the sequential relationship. The sorted candidate sets are migrated to BS.
- Step 3: BS determines the offloading position of the module according to the relationship between module maneuver time and module execution delay in a centralized manner.

When a module is offloaded from BS to MCC, it must go through different network devices. These network devices have various capacities in terms of bandwidth, memory storage, processing speed, etc.

CPMM assumes that different capacities can be represented by one metric, such as bandwidth, and assumes that there is a l hop distance from BS to MCC. The bandwidth of each hop denoted by $\{B_{B\to 1}, B_{1\to 2}, \ldots, B_{l-2\to l-1}, B_{l-1\to l}\}$.

If $Ta_{i,k}$ is processed on MCC servers, then the processing delay (denoted by $TECD_k^{MCC}$) can be expressed as:

$$TECD_{i,k}^{MCC} = Tt_{i,k} + \sum_{i=0}^{l-1}\left(\frac{Data_{i,k}}{B_{i\to \mathrm{i+1}}}\right) + \rho_{i,k} \tag{17}$$

Generally, the delay for a module to process on the MEC server is less than on the MCC server [1–4]. Given a waiting-for-offloading module $Ta_{i,k}$, if its maneuver time satisfies inequality (18), it means that $Ta_{i,k}$ is offloaded to MEC and has no impact on task execution latency. Therefore, it can be offloaded on MEC servers.

$$TECD_{i,k}^{MEC} \le MT_{i,k} < TECD_{i,k}^{MCC} \tag{18}$$

If $Ta_{i,k}$'s maneuver time satisfies inequality (19), it is offloaded to MCC and has no impact on task execution latency. Therefore, it can be offloaded to MCC.

$$TECD_{i,k}^{MCC} \le MT_{i,k} \tag{19}$$

4.7 The Method of Key Module Offloading

This section mainly discusses the method of key module offloading under the single UE.

CPMM uses the verification method to offload the key modules. The steps of key module offloading are as follows:

- Step 1: Getting the key module candidate offloading set.

Taking CTS_i as the input, CPMM uses the selection method of candidate offloading set to obtain the candidate key module offloading set CTS_i^{C3}.

- Step 2: Sorting CTS_i^{C3} in ascending according to the sequential relationship among modules.
- Step 3: Take modules from CTS_i^{C3} in turn, getting the calculation number of modules.

Taking a module (denoted by $Ta_{i,k}^{CT}$) from CTS_i^{C3} and getting its calculation amount according to formula (15), denoted by $UC_{i,k}^{CT}$. Then, according to formulas (17) and (20), obtaining the offloading modules calculation amount, denoted by Tca_i'.

$$T\mathrm{ca}_i' = Tca_i + UC_{i,k}^{CT} \tag{20}$$

- Step 4: Verify again whether inequality (16) is true.

Verify again whether inequality (16) is true. If true, it means that UE_i still cannot meet the computation amount required by the task. Then, getting the next key module from the sorted CTS_i^{C3} again for offloading. Else, stop.

5 The Multi-UE Task Offloading under Multi-Constraint

Since multi-UE can be regarded as task offloading of multiple single-UE tasks offloading without external resource competition, single-UE task offloading lays a foundation for studying multi-UE task offloading. Based on single-UE offloading, Section 5 focuses on multi-UE task offloading.

5.1 *Using Weighted Queuing Method to Deal with Communication Competition*

CPMM first considers the problem of communication channel competition. Assuming the number of modules to be offloaded exceeds the number of channels during ΔT_j, and starts competing for channels.

The modules waiting to be offloaded during ΔT_j come from different offloading sets submitted by the different UEs (including key and non-key modules). Since the key module directly impacts the task processing delay, the priority key module obtains the channel when competing for the channel. According to queuing theory [27], the modules with short execution delays can occupy the channel first, effectively reducing the overall queueing time when there is a queueing competition for resources. So, CPMM prioritizes modules with short delays to obtain communication channels. However, there is a high probability that some modules with long delays will be waiting if key modules and modules with short delays are always given priority. In that case, affecting the processing delay of the offloading task. CPMM adopts the weighted linear queuing method to address the resource competition problem caused by Constraint 4.

Assuming $Count_j$ modules waiting for migration to BS during ΔT_j. CPMM uses the following formula (21) to get every module's priority.

$$T\mathrm{ca}'_i = \omega 1 \times IsKey_{j,k} + \omega 2 \times \left(1/Tt_{i,k} + round \times \phi\right) \tag{21}$$

In which, ϕ is a constant, *round* is the number of times the module waits for the offload interval. Initial $round = 0$. ω_1 and ω_2 are weighted, $\omega_1 + \omega_2 = 1$. When competing for communication channels, $\omega_1 < \omega_2$ means that a higher weight is assigned to the number of competing rounds, in this case, the modules that did not obtain the communication channel in the last round have a higher probability of getting the communication resources in this round. $\omega_1 > \omega_2$ means the key module is given greater weight. Key modules are still prioritized in the next round of communication channel acquisition. These two weight values can be dynamically adjusted according to the actual situation so that specific types of modules can obtain communication channels.

Assuming the communication channel is in contention. The modules that not only the module from the waiting for offloading modules during ΔT_j but also the remaining modules that do not obtain communication channels during ΔT_{j-1} (denoted by ROm_{j-1}) will participate in the competition. These modules use formula (21) to get their priority.

Setting $round = 0$ as the module from the waiting for offloading, and $round = round + 1$ as the module from ROm_{j-1}. When getting every module's priority, CPMM arranges the competing modules in descending order according to the priority and selects the former h modules to obtain the channel. The remaining modules cannot obtain the communication channel. They participate in the next competition of time intervals (i.e., ΔT_{j+1}).

5.2 *Using the Branching Method to Deal with MEC Competition*

Next, consider the MEC resource competition (i.e., Constraint 5).

If MEC resources are sufficient, the key and non-key modules shall be offloaded by the key (or non-key) offloading method. CPMM adopts the branch decision method to address the MEC server competition problem.

Since the key module offloading directly impacts the task completion latency, key modules are prioritized to MEC servers. According to the relationship between the number of concurrent modules to be offloaded and the number of services that MEC can be provided in each time interval, MEC resource competition presents the following relationship when MEC resources are insufficient:

[1] The number of MEC services is greater than the number of key modules to be offloaded

In this case, MEC can meet the requirements of key module offloading, but the non-key module requirements may still need to be met. To reduce the task completion delay, CPMM prioritizes offloading key modules, and the competition of non-key modules obtains the remaining MEC service resources. In other words, under the multi-UE, the key module still adopts the method of the key module to offload, but the method of the non-key module needs to be revised. The improvement ideas are as follows.

According to the timing relationship of task completion, CPMM first selects the non-key module from the waiting-for-offloading module during ΔT_j and then verifies the relationship between the remaining MEC services and the number of non-key modules. If the number of non-key modules is less than the remaining number of services, the remaining services can still meet the non-key offloading. Then, CPMM uses the method of the non-key module to offload. If the number of non-key modules is larger than the remaining number of services, the remaining number of services cannot meet the non-key offloading. In this case, CPMM first arranges the non-key modules in ascending order according to the maneuver time and then takes modules from the sorted set. If the selected non-key module's maneuver time satisfies inequality (18), this module will get MEC servers until all the MEC service resources have been completed. The remaining non-key module will be offloaded to MCC.

[2] The number of MEC services less than the number of key modules to be offloaded

In this case, MEC servers cannot meet the requirements of the key module and non-key module offloading. To reduce the task completion delay, CPMM prioritizes offloading the key modules. The remaining key modules are offloaded to MCC servers. The non-key modules offloaded to MCC servers too. So, the method of the non-key module and the method of the key module need to be revised.

For the method of the non-key module, key modules obtain all MEC resources. MCC servers process all non-key modules. For the method of the key module, CPMM allows some key modules to obtain MEC services while others are processed on MCC servers. CPMM prioritized modules with short delays to get MEC servers' resources according to the queuing theory. So, CPMM arranges these modules in ascending order according to the processing delay and then takes modules from the sorted set until all the MEC service resources are occupied. The remaining key module will be offloaded to MCC servers.

6 Experimental Simulation and Analysis

6.1 Experimental Description

This paper designs and implements a simulator with VC++ language for evaluating the CPMM and compares the CMPP's performance with other similar offloading methods in multi-conditional: MEC offloading method (named MEC), MCC offloading method (named MCC), and without offloading method (named LOCAL). MEC and MCC select the offloading module in the same manner as CPMM, but it is different when selecting the offloading location. MEC only selects MEC servers, and MCC only selects MCC servers. LOCAL does not offload any modules, and all modules are processed locally.

This paper designs the BS, UE, MEC servers, and the MCC in the simulator. UE, BS, MEC servers, and MCC center are organized by the layered heterogeneous networks, which can reflect the real world. BS can perform services and deploy a MEC server with lightweight resources to provide services for UE. MEC servers belonging to BS, which has a fixed number of VMS, can provide execution resources for offloaded modules. MCC center has enough resources to support task offloading. The UE's task in a random manner, and assuming that tasks can be processed concurrently on UE at the same time.

To verify the superior performances of CMPP, this paper compares the processing delay, the failure rate of the task execution, the energy consumption, and the calculated consumption. The compared performance is closely related to the calculated threshold, the min power threshold, the amount of UE, the min-calculation threshold, and the max-tolerance calculation threshold. Consequently, in each group of the performance comparison, this paper sets the max-tolerance threshold as 70%, the min power threshold from 5% to 50%, the calculated threshold from 2 to 5, and the amount of UE from 100 to 400. Meanwhile, this paper also compares the delay and the failure rate under different max-tolerance calculation thresholds. In addition, according to the operation mode of CPMM, when offloading a task, CPMM first needs to classify the modules that make up the task and then use different offloading methods according to the module type. Since running CPMM and partitioning the task into key and non-key modules can consume local resources, the simulation experiment also takes the cost of CPMM operation as an important factor to objectively demonstrate the superior performance of CPMM.

6.2 Comparison of the Task Execution Failure Rate

This experiment shows that CPMM can ensure the success rate of task execution. Fig. 3 demonstrates the failure rate of task execution in different methods under different parameters. It shows that the failure rate increases with the amount of UE increasing, the calculated threshold increasing, and the power threshold increasing. Fig. 3 also shows that the failure rate of CPMM is lower than LOCAL and MEC. MCC has the least failure rate. MEC and LOCAL have the worst failure rate. Compared to LOCAL, CPMM can reduce the failure rate by around 3.9% on average under the power threshold is 15%, can reduce the failure rate by about 4.4% on average under the calculated threshold is 4, and can be reduced by about 4% on average under the amount of UE is 300.

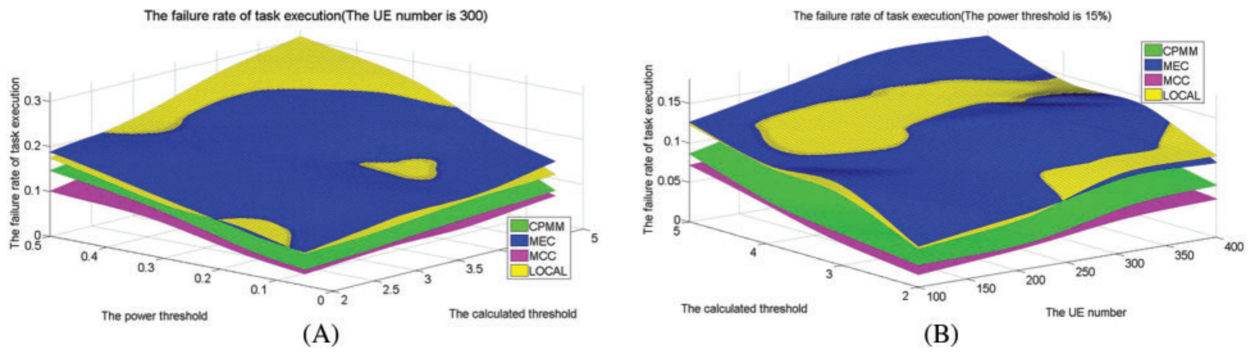
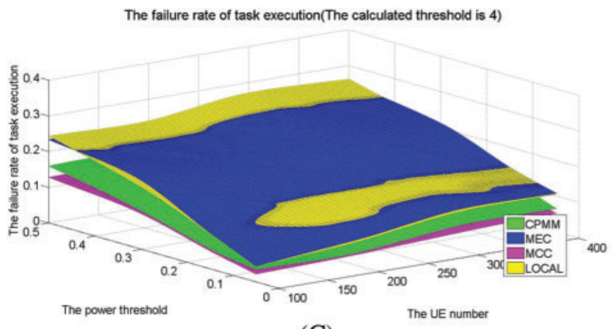
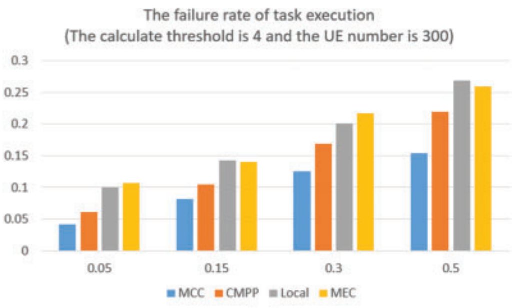


Figure 3: (Continued)



(C)



(D)

Figure 3: The failure rate of task execution

Through above simulation shows that different methods of offloading tasks all have a certain probability of causing the task to fail. This is because UE battery power is Poisson distributed in the experiment. Some UEs have lower battery power in the initial state. Many UEs cannot meet the task requirement, leading to task failure. This phenomenon also exists in the real world. In this experiment, MEC and LOCAL have the max failure rate. The max failure rate even exceeds 30% of the total tasks. This is because LOCAL does not adopt the task-offloading manner to process tasks, but CPMM and MCC process tasks in a task-offloading manner. Therefore, many tasks cannot be processed normally due to energy consumption, resulting in a higher failure rate. Although MEC has also adopted the task-offloading manner, many modules cannot get MEC resources due to the limited MEC service resources, resulting in many task-offload failures. MCC has a sufficient resource supply and does not need to offload tasks to compete for execution resources, so its execution success rate is the highest. Although CPMM uses the remote cloud to perform the offloaded task, it first competes for MEC service resources and then cooperates with MCC to offload. Therefore, CPMM performs better than MCC.

The task failure rate is closed related to the power threshold, the calculation threshold, and the UE amount. When the amount of UE increases, more tasks compete for limited resources, so the task failure rate will increase with the UE number increasing. According to the definition of power threshold, the higher the power threshold, the lower power the UE can use to process tasks. So, the lower power with a higher task failure rate. Recall the definition of calculation threshold, and it is mainly used to select which modules to offload. The higher threshold, the fewer modules that match offloading, resulting in more modules that need to be executed locally. This will undoubtedly exacerbate local energy consumption, resulting in many modules being unable to process due to insufficient power. Therefore, the task failure rate will increase with the calculation threshold increasing.

6.3 Comparison of the Task Completion Delay

This experiment demonstrates that CPMM can effectively reduce task completion delay. This experiment uses the ratio of task completion delay, denoted by$CD_i = Md_{i,j}/Loc_i$, to compare other methods. Loc_i indicate the delay of the execution module when UE_i adopts LOCAL. $Md_{i,j}$ represents the delay of the execution module when CPMM, MEC, and MCC are adopted, respectively. When statistical the processing delay, if a module fails to process due to resource competition, recording 0.

Fig. 4 shows the delay ratio in different methods under different parameters. The delay ratio increases with the number of UE increasing and the power threshold increasing, reducing with the calculated threshold increasing. Fig. 4 also shows that MEC has a minor delay ratio. The delay ratio of CPMM is better than MCC and less than MEC. LOCAL has the worst delay ratio. Compared with MEC, CPMM can increase by around 4.6% on average under the power threshold is 15%, grow by about 3.6% on average under the calculation threshold is 4, and increase by about 4.6% on average under the amount of UE is 300. Compared with MCC, CPMM can reduce around 5.6% on average under the power threshold is 15%, reduces about 5.2% on average under the calculation threshold is 4, and reduces about 4.7% on average under the amount of UE is 300.

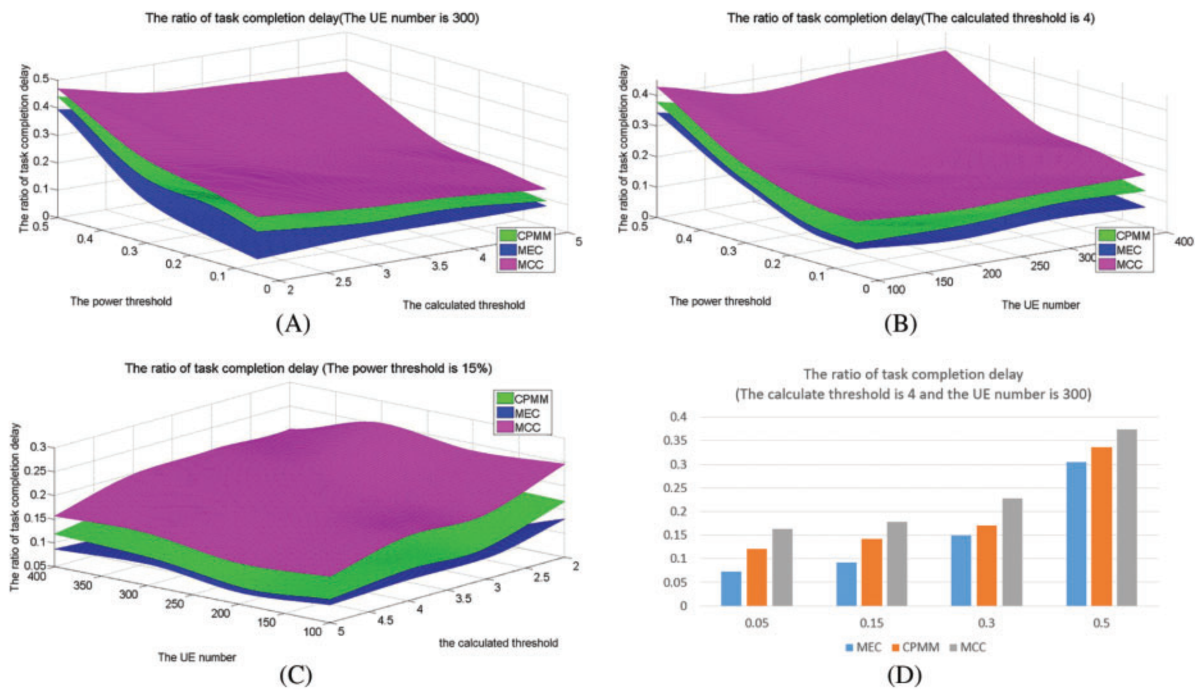


Figure 4: The ratio of task completion delay

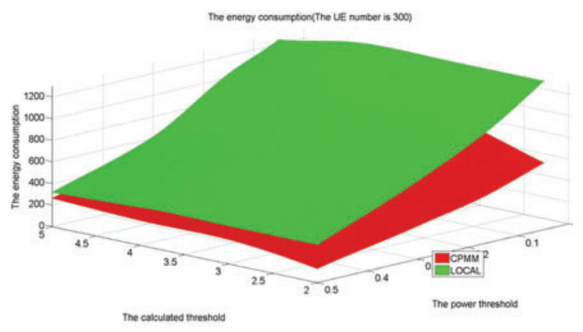
This is because of the following reasons: (1) since MEC, MCC, and CPMM all adopt task offloading, their delay is less than LOCAL. (2) Since the delay of module offloading to the remote cloud is greater than that of MEC, the delay of MEC is less than CPMM and MCC. According to the experiment on failure rate, Fig. 4 shows that MEC has a tall task failure rate. This experiment sets the execution delay as 0 for modules that are not executed. So, MEC has the lowest delay. But its lower delay is obtained by a high failure rate. (3) CPMM uses cooperation mode to offload. Compared to MCC, not all modules are offloaded to MCC. Therefore, its execution delays less than MCC.

The delay is closely related to the UE number, the calculation threshold, and the power threshold. The execution delay is increasing with the number of UE increases. The reasons have been analyzed above. Since the calculation threshold is directly related to the number of modules to be offloaded, it increases with a lower calculation threshold. The power threshold directly impacts whether the module can process or not. The higher the power threshold, the lower the battery power available to UE, resulting in many modules that cannot be executed due to insufficient battery power. So, delay decreases with the power threshold increasing. This can be verified by Fig. 4D. However, from

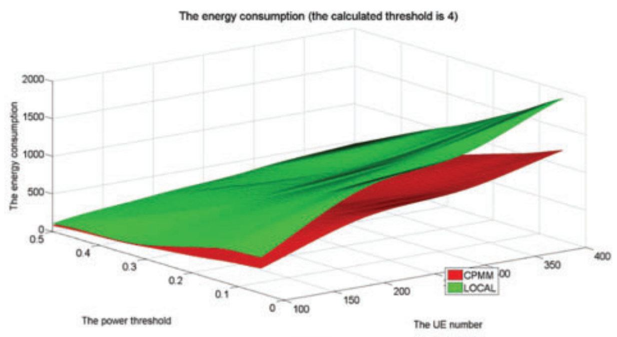
Figs. 4B and 4C, we can see that the delay ratio increases with the power threshold increasing. This is because when the power threshold increases, the modules of CPMM and MCC are not completely dependent on the local. LOCAL is implemented locally and is very sensitive to the change in power threshold. When the power threshold becomes lower, many modules cannot be processed due to insufficient power of UE, so the overall module processing delay changes significantly. According to the definition of the delay ratio, the ratio shows an upward trend.

6.4 Comparison of the Energy Consumption

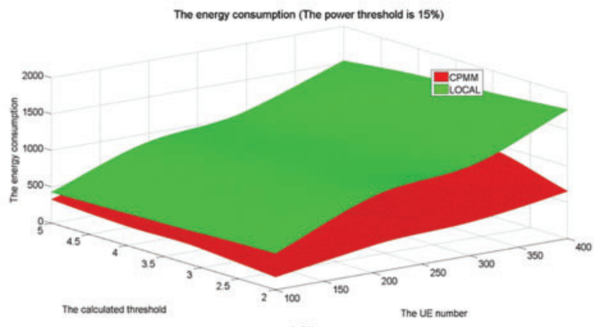
This experiment demonstrates that CPMM can reduce energy consumption. Since MEC and MCC adopt the same offloading mode as CPMM, this experiment only compares CPMM and LOCAL. Fig. 5 shows the energy consumption in different methods under different parameters and shows that the energy consumption increases with the amount of UE increasing, with the calculated threshold increasing, and reduces with the power threshold increasing. CPMM is less than LOCAL. Compared to LOCAL, CPMM's average energy consumption is about 59% of LOCAL under the power threshold is 15%, about 65.4% of LOCAL under the calculated threshold is 4, is about 63% of LOCAL under the amount of UE is 300.



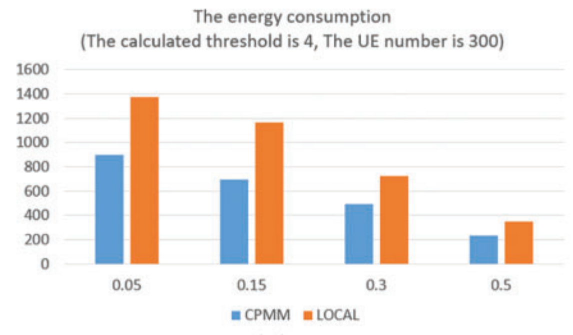
(A)



(B)



(C)



(D)

Figure 5: The energy consumption

According to the above experiment, the energy consumption of CPMM is better than LOCAL. This is because CPMM offloads some non-key modules according to the relationship between modules under the same conditions, reducing the energy consumption of UE. LOCAL does not offload tasks, so the energy consumed by local is higher than CPMM. Meanwhile, Fig. 5 also shows that the increase in the number of UE will increase the number of tasks. Therefore, the total energy consumption will increase. Recalling the definition of power threshold again, when the power threshold increases, it indicates that the UE has less power to process tasks. This will cause many modules to fail. Therefore, the energy consumed decreases with the power threshold increasing. According to the definition of calculating threshold, as the calculation threshold increases, fewer modules will be offloaded, and more modules will be executed locally. So, the energy consumed locally will also increase.

6.5 *Comparison of the Computation Consumption*

Fig. 6 shows the computation consumption in different methods under different parameters and shows that the computation consumption increases with the calculation threshold increasing, and the amount of UE increases and reduces with the power threshold increasing. Compared to LOCAL, CPMM is about 55% of LOCAL when the power threshold is 15%, about 68.6% of LOCAL when the calculation is 4, and about 59% of LOCAL when the amount of UE 300.

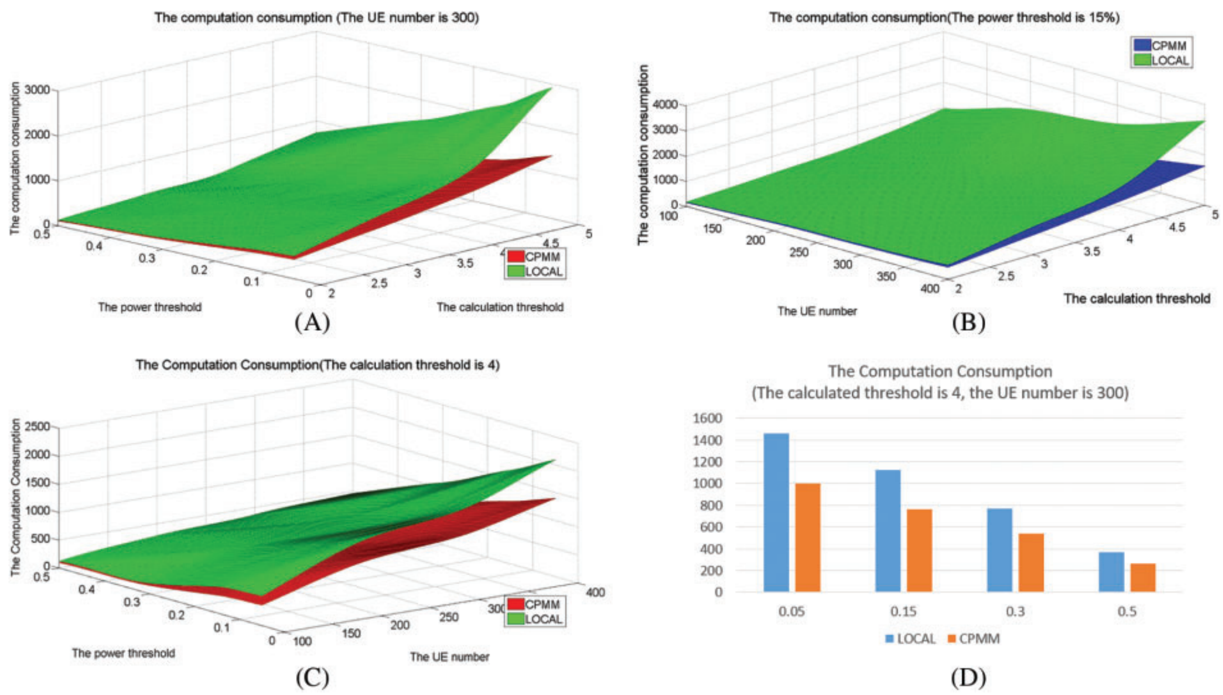


Figure 6: The computation consumption

Rather than select some modules to offload randomly, CPMM selects key (non-key) modules with a larger calculation. In LOCAL, all modules need to be processed locally. Therefore, the LOCAL has the largest computation consumption. In addition, the number of tasks increases with the number of UE increases. Since the calculation threshold is used to determine the scale of offloading modules, a higher threshold means that a larger number of modules need to be processed locally to increase the local calculation. According to the definition of power threshold, a higher threshold indicates lower available power of UE. The lower available power means a larger number of modules cannot be executed. So, the overall calculation consumption will decrease.

6.6 *Comparison under Different Max-Calculation Threshold*

In the above experiments, the max-tolerance calculation threshold is 70%. According to the definition of the max-tolerance threshold, the performance to be compared by simulation is also closely related to this threshold. This experiment sets $\delta = 2, pow_{min} = 15\%$, and the UE number as 300, mainly used to verify that CPMM also has better performance under the different max-tolerance thresholds.

Fig. 7A shows the ratio task completion delay under the different max-tolerance thresholds and shows that the delay ratio increases with the max-tolerance threshold reduction. The UE's computing resources will be increased relatively with the threshold increasing according to the definition of the max-tolerance threshold. Increased computing power means the probability of UE triggering the forced offloading is also decreasing. In other words, when the max-tolerance threshold increases, it tends to use active offloading to form the candidate offloading set. Active offloading mainly selects non-key modules and has a lower probability of selecting key modules. Since key modules directly impact task processing delay, fewer key modules are selected for offloading, and the overall delay is reduced too. Under the parameters this experiment sets, MEC has the minimum delay ratio. CPMM following, MCC larger than CPMM. LOCAL has the Maximum delay ratio. MEC achieves lower task execution delay through a higher task execution failure rate, while MCC increases task execution latency due to longer data migration delay. CPMM completes the task offload cooperatively, so it is between MCC and MEC. Compared to MEC, CPMM can increase the delayed radio by around 5% on average. CPMM can reduce the delay ratio by around 6% on average compared to MCC.

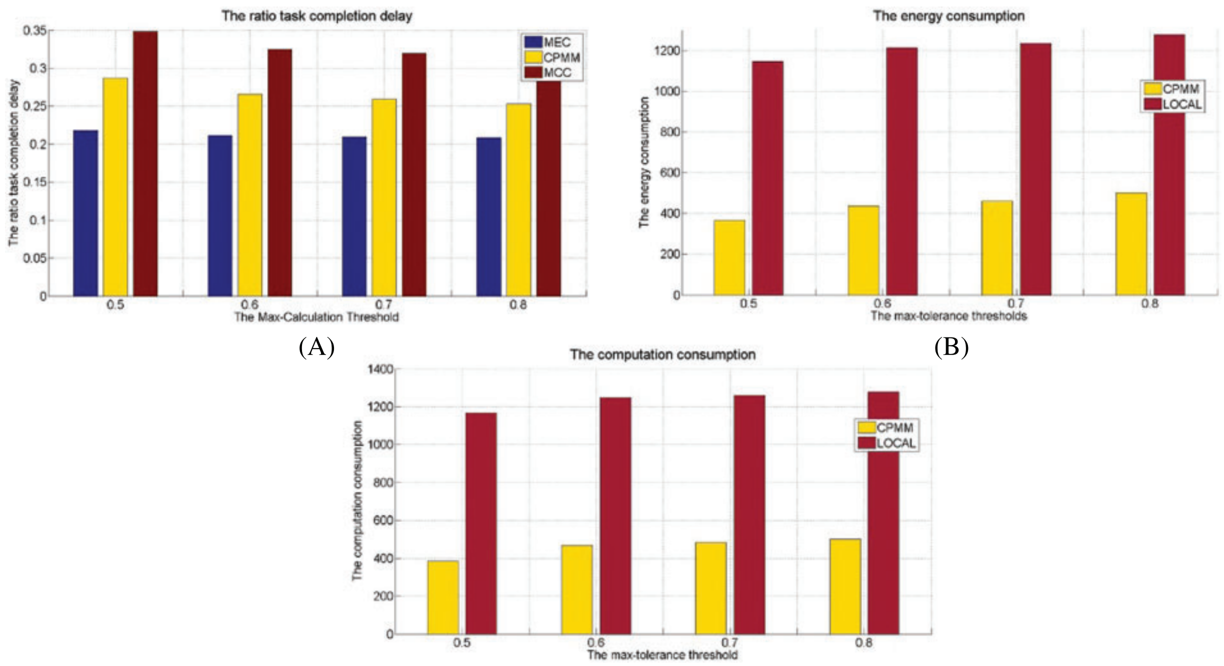


Figure 7: The performance under different max-calculation threshold

Fig. 7B demonstrates the energy consumption under the different max-tolerance thresholds. According to Fig. 7B, energy consumption increases with the max-tolerance threshold increasing. This is because the computing power of the UE increases with the max-tolerance threshold increasing, which means more modules are executing locally. Therefore, more energy-consuming locally. Under the experimental parameters we set, CPMM is better than LOCAL. Since CPMM is a partial task offloading strategy in a cooperative mode, task offloading can reduce local energy consumption. Compared to the LOCAL, the average energy consumption of CPMM is about 40% of that of LOCAL.

Fig. 7C shows that the computation consumption increases with the max-tolerance threshold increasing. This is because more modules will be processed locally with the max-tolerance threshold increasing, so the total local computation is also increasing. Since CPMM adopts the offloading method to reduce local computing consumption, LOCAL does not offload. Therefore, CPMM is better than LOCAL. Under the experimental parameters this experiment sets, the calculation consumption of CPMM only accounts for about 37% of the calculation consumption of LOCAL.

7 Conclusion

This paper proposes a cooperation mode partial task offloading method (CPMM) to deal with the problem of task offloading for multi-UE under multi-constraints. CPMM focuses on the modules that make up a task are parallel dependencies, aiming to reduce the energy and computation consumption while meeting the task completion delay as much as possible. It focuses on many constraint conditions, including UE battery power, task execution, computing power, communication channel, MEC service resources, etc. CPMM first discusses the method for single-UE to select candidate offloading module sets under multi-constraints. Then, the task offloading method of multi-UE under multi-constraints is discussed. When discussing single-UE offloading, it uses a critical path algorithm to divide modules into key and non-key modules. It proposes the selection method of the non-key module and key module. Meanwhile, the method of how MEC and MCC cooperate in offloading is formulated. According to the constraints of the multi-UE and single-UE task offloading method, the weighted queuing method and branch processing method is used to offload multi-UE tasks in multi-constraints. Extensive experiments show that CPMM has better performance than other similar methods.

However, CPMM also has many limitations. For example, CPMM focuses on computing-type task offloading and less on data resource access-type task offloading when considering task offloading. In the future, we plan to study the data resource access-type task offloading and the edge caching strategy to reduce the completion delay of data resource access-type tasks through caching.

Acknowledgement: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Shengyao Sun and Jiwei Zhang; data collection: Ying Du, Jiwei Zhang; analysis and interpretation of results: Shengyao Sun, Jiajun Chen, Xuan Zhang; draft manuscript preparation: Yiyi Xu. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: According to the edge tasks offloading in the real world, we used C++ language to simulate and validate our proposed method; the data source mainly comes from laboratory simulations, rather than real-life datasets, so we feel that there is no need to present these simulated data.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] K. Y. Zhang, X. L. Gui, D. W. Ren, J. Li, J. Wu *et al.,* "Survey on computation offloading and content caching in mobile edge networks," *Journal of Software*, vol. 30, no. 8, pp. 2491–2516, 2019.

[2] Y. Z. Zhou and D. Zhang, "Near-end cloud computing: Opportunities and challenges in the post-cloud computing era," *Chinese Journal of Computers*, vol. 42, no. 4, pp. 677–700, 2019.

[3] Z. Y. Li, Q. Wang, Y. F. Chen and R. F. Li, "A survey on task offloading research in vehicular edge computing," *Chinese Journal of Computers*, vol. 44, no. 5, pp. 963–982, 2021.

[4] I. Akhirul, D. Arindam, G. Manojit and C. Suchetana, "A survey on task offloading in multi-access edge computing," *Journal of Systems Architecture*, vol. 118, no. 1, pp. 1–16, 2021.

[5] Global App Store downloads reach 8.6 billion in the first quarter of 2022. [Online]. Available: https://baijiahao.baidu.com/s?id=1731224939845177595&wfr=spider&for=pc

[6] J. Liu, Y. Mao, J. Zhang and K. B. Letaie, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *2016 IEEE Int. Symp. on Information Theory (ISIT)*, Barcelona, Spain, pp. 1451–1455, 2016.

[7] Y. Y. Mao, J. Zhang and B. L. Khaled, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[8] S. Ulukus, A. Yener, E. Erkip, O. Simeone and M. Zorzi, "Energy harvesting wireless communications: A review of recent advances," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 3, pp. 360–381, 2015.

[9] M. Kamoun, W. Labidi and M. Sarkiss, "Joint resource allocation and offloading strategies in cloud enabled cellular networks," in *2015 IEEE Int. Conf. on Communications (ICC)*, London, UK, pp. 5529–5534, 2015.

[10] B. Li, Q. He, F. F. Chen, H. Jin, Y. Xiang *et al.,* "Auditing cache data integrity in the edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1210–1223, 2020.

[11] J. Yan, S. Z. Bi, L. J. Duan, Y. Jun and A. Zhang, "Pricing-driven service caching and task offloading in mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 20, no. 7, pp. 4495–4512, 2021.

[12] J. Wang, J. Hu, G. Min, A. Y. Zomaya and N. Georgalas, "Fast adaptive task offloading in edge computing based on Meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2020.

[13] Z. L. Ning, X. J. Wang, X. J. Kong and W. G. Hou, "A social-aware group formation framework for information diffusion in narrowband Internet of Things," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1527–1538, 2017.

[14] Y. T. Wang, M. Sheng, X. J. Wang, L. Wang and J. D. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[15] W. Liu, Y. C. Huang, W. Du and W. Wang, "Resource-constrained serial task offload strategy in mobile edge computing," *Journal of Software*, vol. 31, no. 6, pp. 1889–1908, 2020.

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

[17] M. Deng, T. Hui and M. Bo, "Fine-granularity based application offloading policy in small cell cloud-enhanced networks," in *2016 IEEE Int. Conf. on Communications Workshops (ICC)*, Kuala Lumpur, pp. 638–643, 2016.

[18] M. H. Chen, B. Liang and D. Min, "A semidefinite relaxation approach to mobile cloud offloading with computing access point," in *2015 IEEE 16th Int. Workshop on Signal Processing Advances in Wireless Communications (SPAWC) IEEE*, Stockholm, Sweden, pp. 186–190, 2015.

[19] O. Munoz, A. P. Iserte, J. Vidal and M. Molina, “Energy-latency trade-off for multiuser wireless computation offloading,” in *Wireless Communications & Networking Conf. Workshops IEEE*, Istanbul, Turkey, pp. 29–33, 2014.

[20] N. A. Sulieman, C. L. Ricciardi, W. Li, Z. Albert and V. Massimo, “Edge-oriented computing: A survey on research and use cases,” *Energies*, vol. 15, no. 2, pp. 1–28, 2022.

[21] R. P. Lin, T. Z. Xie, S. Luo, X. N. Zhang, Y. Xiao *et al.,* “Energy-efficient computation offloading in collaborative edge computing,” *IEEE Internet of Things Journal*, vol. 9, no. 21, pp. 21305–21322, 2022.

[22] T. Khikmatullo and D. H. Kim, “Blockchain-enabled approach for big data processing in edge computing,” *IEEE Internet of Things Journal*, vol. 9, no. 19, pp. 18473–18486, 2022.

[23] J. Yan, S. Z. Bi, L. J. Duan, Y. Jun and A. Zhang, “Pricing-driven service caching and task offloading in mobile edge computing,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 4, pp. 4495–4512, 2021.

[24] K. L. Xiao, Z. P. Gao, W. S. Shi, X. S. Qiu, Y. Yang *et al.,* “EdgeABC: An architecture for task offloading and resource allocation in the Internet of Things,” *Future Generation Computer Systems*, vol. 107, no. 1, pp. 498–508, 2020.

[25] J. W. Zhang, M. Z. A. Bhuiyan, X. Yang, T. Wang, X. Xu *et al.,* “AntiConcealer: Reliable detection of adversary concealed behaviors in EdgeAI assisted IoT,” *IEEE Internet of Things Journal*, vol. 9, no. 22, pp. 22184–22193, 2021.

[26] C. A. Shaffer, “Graphs,” in *A Practical Introduction to Data Structures and Algorithm Analysis*, 3rd ed., Virginia, USA: Publishing House of Electronics Industry, pp. 381–411, 2011.

[27] S. M. Ross, “Queueing Theory,” in *The Introduction to Probability Models*, 11th ed., Los Angeles, California, USA: Academic Press, pp. 481–538, 2014.