information

MDPI

Article

Computation Offloading Strategy in Mobile Edge Computing

Jinfang Sheng, Jie Hu, Xiaoyu Teng, Bin Wang * and Xiaoxia Pan

School of Computer Science and Engineering, Central South University, Changsha 410073, China; jfsheng@csu.edu.cn (J.S.); 174712017@csu.edu.cn (J.H.); 164611109@csu.edu.cn (X.T.); 1402140114@csu.edu.cn (X.P.)

* Correspondence: wb_csut@csu.edu.cn

Received: 25 April 2019; Accepted: 29 May 2019; Published: 2 June 2019

Abstract: Mobile phone applications have been rapidly growing and emerging with the Internet of Things (IoT) applications in augmented reality, virtual reality, and ultra-clear video due to the development of mobile Internet services in the last three decades. These applications demand intensive computing to support data analysis, real-time video processing, and decision-making for optimizing the user experience. Mobile smart devices play a significant role in our daily life, and such an upward trend is continuous. Nevertheless, these devices suffer from limited resources such as CPU, memory, and energy. Computation offloading is a promising technique that can promote the lifetime and performance of smart devices by offloading local computation tasks to edge servers. In light of this situation, the strategy of computation offloading has been adopted to solve this problem. In this paper, we propose a computation offloading strategy under a scenario of multi-user and multi-mobile edge servers that considers the performance of intelligent devices and server resources. The strategy contains three main stages. In the offloading decision-making stage, the basis of offloading decision-making is put forward by considering the factors of computing task size, computing requirement, computing capacity of server, and network bandwidth. In the server selection stage, the candidate servers are evaluated comprehensively by multi-objective decision-making, and the appropriate servers are selected for the computation offloading. In the task scheduling stage, a task scheduling model based on the improved auction algorithm has been proposed by considering the time requirement of the computing tasks and the computing performance of the mobile edge computing server. Extensive simulations have demonstrated that the proposed computation offloading strategy could effectively reduce service delay and the energy consumption of intelligent devices, and improve user experience.

Keywords: mobile edge computation; computation offloading; analytic hierarchy process; auction algorithm

1. Introduction

With the rapid development of mobile Internet services such as augmented reality, virtual reality, online games, and ultra-clear video, as well as IoT applications such as group intelligence, intelligent perception, and environmental monitoring, the service level of the mobile devices are becoming increasingly demanding. These applications require a wealth of computing resources with high energy consumption. However, mobile smart devices are subjected to computing power and battery life due to the limitations of their size. A processing delay due to these limitations could ruin the user experience and lead to negative feedback if it is more than tens of milliseconds. A promising technique to solve this problem is to offload computationally intensive tasks to nearby servers with more abundant resources, which is called computation offloading [1] or nomadic services [2]. The method includes six steps:

migration environment awareness (server discovery), task segmentation, migration decision, task upload, task remote execution, and result return. The two most important steps are task segmentation and migration decisions. Task segmentation focuses on fine-grained partitionings such as MAUI in [3], ThinkAir in [4], and Phone2Cloud in [5], which is based on application code. It divides applications according to methods, functions, and other criteria. Migration decision is the core issue of computation offloading technology, which focuses on whether to perform computation offloading, time overhead, and energy consumption. The authors in [6,7] considered the offloading decision from the perspective of energy, while Deng et al. [8] considered it on server resources and bandwidth.

Computation offloading was first applied to Mobile Cloud Computing (MCC) [9–11], which offloads the computing tasks of mobile terminals to traditional cloud data centers where centralized computing and storage are the main features. The main obstacle to the traditional cloud computing model for computation offloading is that the delay experienced by reaching a (remote) cloud server over a Wide Area Network (WAN) may offset the time that is saved by computation offloading. In many real-time mobile applications such as online gaming, speech recognition, and Facetime, the quality of the user experience can be severely influenced [12–14]. Focusing on this shortcoming, scholars put forward the idea that a cloud server could be made closer to the users. Therefore, the concept of the cloudlet [15–17] was proposed. The fixed server was connected via Wi-Fi to perform the computation and offloading [18–20]. However, the cloudlet server's location is fixed and there is still a limited number of them. The lack of available fixed servers may limit the suitability of the cloudlet. Aiming to solve the drawbacks of both the MCC and cloudlet in computation offloading applications, the European organization (TROPIC) has proposed the ability to provide cloud computing at base stations. Subsequently, the European Telecommunications Standards Institute (ETSI) proposed the concept of Mobile Edge Computing (MEC), which combines MEC with the deployment of small base stations that can achieve a physical proximity between the mobile device and the cloud server providing the computing service, thereby, reducing the delay in application access [21,22].

As the key technology for future 5G communication, the most important application of the MEC is computation offloading [23–25]. Many scholars have made important contributions to this research [26–28]. The multi-user computation offloading problem in a multi-channel wireless interference environment was studied in [29], and the game theory method was used to implement effective channel allocation in a distributed manner. Orthogonal Frequency Division Multiplexing (OFDMA) was used to access multi-user and multi-MEC server systems in [30,31]. The two-layer optimization method was used to decouple the original NP-hard problem into a low-level problem to seek power and sub-carrier allocation and upper-layer task offloading. A heuristic offloading decision algorithm (HODA) was proposed in [32], which was semi-distributed with the joint optimization of offloading decision-making and communication resources to maximize system utility. A task scheduling algorithm based on the Lyapunov optimization theory was proposed in [33], which combined task offloading, device–base station association, and the optimal scheduling of sleep mode on base stations to minimize the overall energy consumption of equipment and base stations. An adaptive sequential offloading task was proposed to solve these problems in [8] that mobile users could make offloading decisions sequentially based on the current interference environment and available computing resources [34–36]. This could achieve good results in reducing delay and energy consumption. All of the above studies have played an important role in promoting the development of computation offloading techniques, but they also have their own limitations. For example, the studies in [29,30,32] focused on the effects of channel reuse and channel competition on offloading decisions. However, the effect of server resources used in computation offloading performance was not considered. The authors in [6,8,33] focused on the study of MEC server resources and scheduling. Nevertheless, they ignore the impact of the task itself such as task data size and computing resource requirements on computation offloading.

In this paper, we propose a computation offloading strategy by considering factors such as computing task size, computing requirements, server computing power, and network bandwidth as

the basis of the offloading decision. After considering the time-saving ratio, the energy saving ratio, and the remaining resources of the server, the improved analytic hierarchy process was adopted. Analytic Hierarchy Process (AHP) conducts multi-objective decision-making to evaluate the server and selects the server that undertakes the offloading task. By considering the time requirement of the computing task and the computing performance of the MEC server comprehensively, a task scheduling model based on improved auction was proposed. Experiment results show that the proposed offloading strategy can effectively reduce service delay, reduce the energy consumption of smart devices, and improve the user experience.

2. Offloading Policy Model

In light of the fact that the computing performance of mobile devices cannot meet the needs of computing-intensive applications, we propose a computation offloading method to solve the problem by offloading the tasks of mobile devices to MEC servers. Computation offloading technology mainly solves three problems: (a) Should the task be offloaded? When a task needs to be offloaded, there are several nearby servers that satisfy the offloading conditions; (b) Which server should be selected to execute the task? (c) Which virtual machine (VM) is needed to handle the task after selecting the server? To address these three problems, we propose a computation offloading strategy that can be divided into three stages.

The first stage is the offloading decision-making stage, which decides whether a task needs to be offloaded by considering both the time and energy consumption of local computing and offloading computation.

In the second stage, there are multiple servers which satisfy the offloading conditions. We leverage the improved AHP method to select the most suitable server by considering factors such as time, energy consumption, and server CPU resources.

The third stage is to schedule the task by improving the auction algorithm after the task determines the uploaded server. Then, tasks are performed on the appropriate VM to achieve the global optimum, that is, MEC servers can perform more tasks over a period of time. At the same time, a compensation model for bidding failure has been proposed to ensure the fairness of task scheduling by considering the impact of task deadlines to compensate for the buyer's price.

3. Offloading Decision Model

There are two main criteria for determining whether to carry out computation offloading, that is, whether it can reduce the running time of tasks and whether it can save the energy consumption of mobile devices. This paper presents a model for the time consumption and energy consumption of the task local execution and offloading execution.

3.1. Time-Consuming and Energy Consumption Model for Local Execution

When the task is running locally, the time required to process the task is:

$$T_L = \frac{C_n}{V_L}. \tag{1}$$

In Equation (1), C_n is the computation resource required by task n. The computation resource here is expressed by the number of CPU cycles needed to perform the task, and V_L is the execution rate of the local CPU. When the task is running locally, the energy E_L used to process the task is:

$$E_L = T_L \times P_L, \tag{2}$$

where P_L is the computing power of the mobile device.

3.2. Time-Consuming Model and Energy Consumption Model for Offloading Execution

When the task runs on the MEC server, the time $T_{offload}$ needed to process the task consisting of three parts: the transmission time T_{up} of the task uploaded to the server, the time T_{exe} of the task running on the server, and the time T_{down} of the task result downloaded to the mobile device. Therefore, the computation offloading time $T_{offload}$ is expressed as:

$$T_{offload} = T_{up} + T_{exe} + T_{down}. \tag{3}$$

Many studies [30,37,38] have neglected the effect of the result return time T_{down} on the offloading time. Because the output of most applications is much smaller than the input, the return time has little effect on the total time consumption, which is analyzed in detail in [30]. Therefore, the offloading time can be simplified as:

$$T_{offload} = T_{up} + T_{exe}. \tag{4}$$

The transmission time T_{up} required for the task upload can be expressed as:

$$T_{up} = \frac{D_n}{W \log_2(1 + \frac{P_{up} \times Los}{N})}, \tag{5}$$

where D_n is the amount of data that needs to be uploaded for computing tasks, N represents the Gauss noise power in the channel, W is the channel bandwidth, and P_{up} is the upload power of the mobile device. Los is the channel gain. The wireless interference model based on cellular wireless environment defines Los as a distance-based function $Los = d^{-\alpha}$ with the α value of 4.

The running time T_{exe} of a task on the MEC server is expressed as:

$$T_{exe} = \frac{C_n}{V_{cloud}}, \tag{6}$$

where V_{cloud} is the execution rate of the server CPU.

According to Equations (5) and (6), the offloading time of tasks can be obtained as follows:

$$T_{offload} = \frac{D_n}{W \log_2(1 + \frac{P_{up} \times Los}{N})} + \frac{C_n}{V_{cloud}}. \tag{7}$$

When a task runs on the MEC server, the energy consumed by the mobile device is only the energy consumed when the task is uploaded, so, when the task is offloaded, the energy consumption E_{up} is expressed as:

$$E_{up} = T_{up} \times P_{up} = \frac{D_n}{W \log_2(1 + \frac{P_{up} \times Los}{N})} \times P_{up}. \tag{8}$$

3.3. Decision-Making Basis for Offloading

When the computation offloading takes less time than the local execution and energy consumption is less than the local energy consumption, computation offloading can improve the quality of the user experience. That is to say, when Equations (9) and (10) are satisfied, computation offloading can be carried out:

$$T_{offload} < T_L, \tag{9}$$

$$E_{up} < E_L. \tag{10}$$

Equations (11) and (12) can be derived from Equations (1), (2), and (7)–(10):

$$V_{cloud} > \frac{V_L C_n}{C_n - \frac{D_n V_L}{W \log_2(1 + \frac{P_{up} \times Los}{N})}}, \tag{11}$$

$$W > \frac{V_L P_{up} D_n}{P_L C_n \log_2(1 + \frac{P_{up} \times Los}{N})}. \quad (12)$$

Equations (11) and (12) indicate that the computation can be offloaded when the computing power V_{cloud} provided by the server is larger than the value on the right side of inequality (Equation (11)), and the bandwidth of the network environment in which the user lives can satisfy Equation (12).

For a given task n, the computing speed V_L of the mobile device, the computing resource C_n, the data size D_n, and the upload power P_{up} of the mobile device can be obtained. Los is a function based on distance. As long as the distance between the mobile device and the base station is known, it can be calculated. For Equations (11) and (12), the unknown parameter is only N. In this paper, N is defined as –100 dBm according to document [39]. So far, the values of the right half of the inequalities (Equations (11) and (12)) can be calculated.

The first step in computation offloading is environmental awareness. In this step, we can obtain the computing power V_{cloud} of the MEC server and the network bandwidth W of the environment, and then compare the results with those on the right side of the inequality. If Equations (11) and (12) are satisfied, computation offloading can be performed.

4. Server Selection Model Based on Improved AHP

When the task satisfies the computation offloading conditions, the next step needs to be considered: When servers on multiple base stations satisfy the computation offloading conditions, which server will the task be offloaded to? For mobile devices, we need to consider two factors: power consumption and service time. For servers, we should consider two factors: the computing power of the servers and the residual CPU resources of the servers. As each factor will affect the choice of server, the evaluation of servers should be based on all of these factors. Analytic Hierarchy Process (AHP) is a classical model for multi-objective decision-making. Subjective judgment is used when AHP is applied to compare the importance of two factors. However, subjective judgment is often uncertain. Therefore, this paper improved on the basis of AHP and proposed a server selection model based on improved AHP. This model consisted of three steps: designing the hierarchical structure model, constructing the pairwise comparison matrix, and integrating the server evaluation index. The subjective judgment method was changed to the parameter computation method, which reduces subjective participation, measures, and judges the server more accurately, and chooses the most suitable server for the computation offloading.

4.1. Hierarchical Model

As shown in Figure 1, the hierarchical model in this paper consisted of three layers: the target layer, criterion layer, and scheme layer. The target layer is the final conclusion of the model as the comprehensive evaluation of servers meets the requirements of computation offloading. The criteria layer is the factor that is relied on when evaluating the servers. We selected three factors: time and energy saved by task offloading, and the remaining CPU resources of the servers. The reason that the server computing power was not taken into account is that task running time and computing power are closely related, and the computing power determines the task running time. The scheme layer is a server that can be used for the computation offloading, and a server represents a solution.

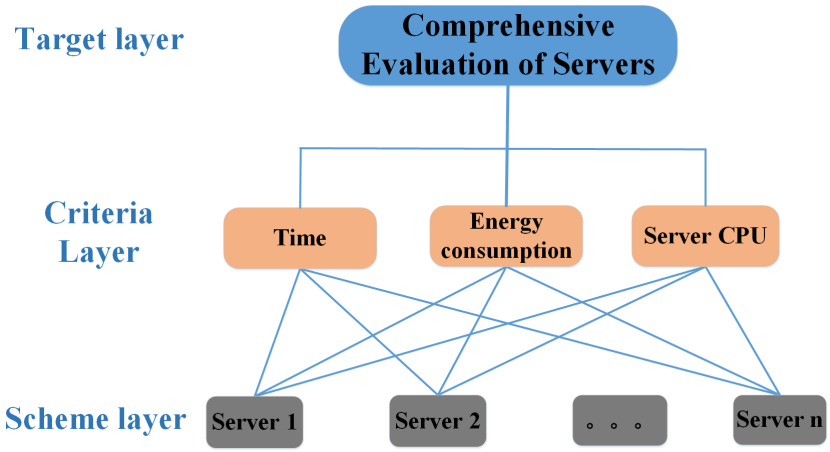

Figure 1. AHP Hierarchy model for choosing a suitable server. The hierarchical model consisted of three layers: the target layer, criterion layer, and scheme layer. The target layer is the final conclusion of the model as the comprehensive evaluation of servers that meet the requirements of computation offloading. The criteria layer is the factor that is relied on when evaluating the servers. The scheme layer is a server that can be used for the computation offloading, and a server represents a solution.

4.2. Pairwise Comparison Matrix

The meaning of a pairwise comparison matrix is to express the relative importance of all elements in each layer to a specific element in the previous layer in the form of a matrix. In this paper, the hierarchical structure consisted of three layers, so we established the paired matrix of the criterion layer to each element in the target layer and the paired matrix of the scheme layer to the criterion layer. The construction method of a pairwise comparison matrix is a pairwise comparison.

First, the paired matrix of the criterion layer to the target layer is established. In this paper, there are n elements in the target layer and three elements in the criterion layer. Based on the pairwise comparison, n 3×3 matrices needed to be established. The importance of each factor does not depend on subjective judgment, but first obtains the proportion of time saved, the proportion of energy consumption saved, and the proportion of the remaining CPU of the server when different servers are used for the computation offloading. Then, the matrix can be established by the way of Two-to-Two comparisons. The time and energy consumption are calculated from the above, and the proportion of the remaining CPU needs to be monitored.

In this paper, because the construction of the comparison matrix was obtained by comparing the percentages of different elements, the fluctuation between values was small. If the original 1–9 scale of the AHP algorithm is used to construct the matrix, the accuracy is less. Therefore, in this paper, the 1–2 scale method has been used to construct the matrix. The constructed scales are shown in Table 1.

Table 1. Scaling implications.

Scaling	Meaning
1 1.2 1.4 1.6 1.8	The former is more important than the latter as the numerical value increases.
reciprocal	If the importance ratio of factor i to factor j is a_{ij}, then the importance ratio of factor j to factor i is $1/a_{ij}$.

The rules for establishing matrices are as follows: each value in the matrix is obtained by comparing the actual values. According to the scale in Table 1, the nearest value is filled in the matrix, as shown in Equation (13):

$$A = \begin{bmatrix} 1 & a_1/a_2 & a_1/a_3 \\ a_2/a_1 & 1 & a_2/a_3 \\ a_3/a_1 & a_3/a_2 & 1 \end{bmatrix}. \tag{13}$$

The eigenvectors corresponding to the maximum eigenvalues of n matrices are obtained:

$$w_i = \{a_{i1}, a_{i2}, a_{i3}\}, i = 1, 2, ..., n. \tag{14}$$

By averaging the n eigenvectors, the relative importance of the criterion layer to the target layer is obtained:

$$w_a = \{a_1, a_2, a_3\} = \{\frac{\sum_{i=1}^{n} a_{i1}}{n}, \frac{\sum_{i=1}^{n} a_{i2}}{n}, \frac{\sum_{i=1}^{n} a_{i3}}{n}\}, \tag{15}$$

where w_a is the weight of each factor in the criterion layer relative to the target layer. For example, to choose a server, more attention should be paid to saving time, followed by energy consumption, and then the remaining CPU of the server, which can be calculated as {0.6, 0.3, 0.1} as shown above.

Next, we considered the importance of the scheme layer to the various factors of the criterion layer. For example, the weight of time mentioned above was 0.6. When considering the importance of each scheme to time, assuming that the three servers satisfy the requirements, where server 1 saves the most time and server 3 saves the least time, it is possible to divide 0.6 into three servers according to the importance degree {0.4, 0.15, 0.05}. There are three factors in the criterion layer. Under these three factors, comparing two servers requires three $n \times n$ matrices.

The rules for establishing matrices are as follows.

The first line of the matrix can be obtained by comparing the actual values. For example, under the time factor, the first line of the matrix compares the time parameters separately. In addition, under the energy factor, it compares the energy consumption separately. That is to say, the comparison parameters are different under different factors. The other data are completed by $a_{ij} = a_{ik}/a_{jk}$ transmission, as shown in Equation (16), taking time as an example:

$$B_1 = \begin{bmatrix} 1 & b_1/b_2 & \dots & b_1/b_n \\ b_2/b_1 & 1 & \dots & b_2/b_n \\ \vdots & \vdots & \ddots & \vdots \\ b_n/b_1 & \dots & \dots & 1 \end{bmatrix}. \tag{16}$$

Similarly, the matrix satisfies the consistency condition and can obtain the eigenvectors of the matrix under the time factor as follows:

$$w_t = \{b_{11}, b_{12}, \dots, b_{1n}\}. \tag{17}$$

The eigenvector of the energy consumption factor matrix is obtained by the same theorem:

$$w_p = \{b_{21}, b_{22}, \dots, b_{2n}\}. \tag{18}$$

The eigenvector of CPU factors is

$$w_c = \{b_{31}, b_{32}, \dots, b_{3n}\}. \tag{19}$$

The characteristic matrix of the three components is as follows:

$$w_b = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ b_{31} & b_{32} & \dots & b_{3n} \end{bmatrix}. \quad (20)$$

4.3. Integration of Server Evaluation Indicators

The weights, w_a, of each factor of the criterion layer on the target layer, w_b, of each scheme in the scheme layer to the criterion layer are obtained from the first two sections. However, the comprehensive evaluation of the server is to calculate the weight of different schemes relative to the target. That is to say, the weight of the scheme layer to the target layer can be calculated by Equation (21):

$$W = w_a \times w_b. \quad (21)$$

The value of W is the weight of the different servers, and the server with the most weight is selected for the offloading computation.

5. Task Scheduling Based on the Improved Auction Algorithm

When a task determines the server to be offloaded, the next two problems to consider are: Which VM is the task to be offloaded on? How should tasks be scheduled when the number of offloaded tasks is large and the virtual machine resources are limited? To solve these two problems, this paper proposes a task scheduling strategy based on the improved auction algorithm.

5.1. Auction Models and Formalization of Problems

The auction model for task scheduling is shown in Figure 2. In the auction model, there are two roles: the buyer and the seller. The seller provides goods, and the buyer participates in the auction of products that meet their own needs; the higher price wins.

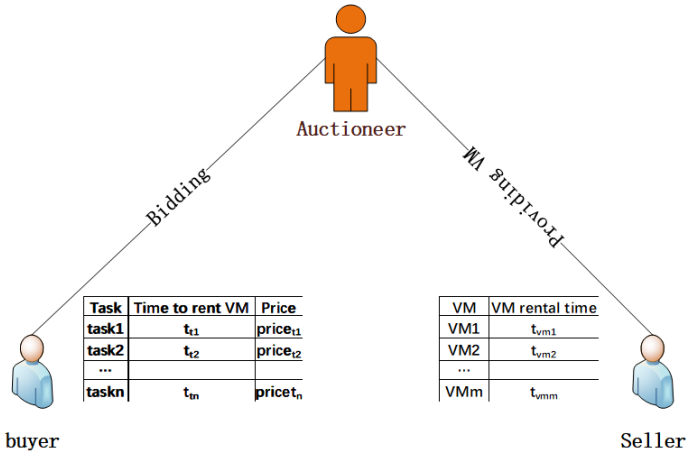

Figure 2. Auction model for task scheduling. There are two roles in the auction model: the buyer and seller. The seller provides goods, and the buyer participates in the auction of products that meet their own needs, and the higher price wins.

The seller in Figure 2 refers to the MEC server, and the merchandise sold is the lease time of the computing resources. The computing resources here are refined into the free VM in the MEC server, that is, the rental time of the free VM is auctioned by the seller. Buyers refer to tasks that require offloading computations, the buyer's requirement is the time required to run the task on VM, namely T_{exe}, and the buyer's bid criteria are $price = \frac{1}{T_L - T_{offload}}$. In this equation, $T_{offload}$ takes time to offload tasks and T_L takes time to run the tasks on the mobile device, which is subtracted from the time saved by the offloading computation. At the same time, T_L can be regarded as the deadline to

complete the task, so the difference also represents the degree of urgency to complete the task. Since the offloading computation has been chosen, it is necessary to make the time of the task offload satisfy $T_L - T_{offload} > 0$ as much as possible. That is, the closer the task is to the deadline, the more urgent the task is, so the reciprocal of the difference is calculated. The less time there is left, the greater the reciprocal, which means that the task is willing to pay a higher price.

In this auction model, it is assumed that the computing power of VM is the same, that is, the only factor that users need to consider when renting VM is the time to complete the tasks. We define a time slice as s, the user's time to use VM and the server's time to rent VM is integral times of s. Each time slice is auctioned once, and each time the users participating in the auction are not only the users who have failed in the previous auction but also the new users who have joined the auction.

When the auction begins, the rental time of the VM participating in the auction is t_{vm}, and the time required for the task to rent the virtual machine is t_t. Only when the task of $t_t < t_{vm}$ is satisfied can the VM participate in the competition, and the higher bidder obtains the right to use the VM. At the end of the rental period, the user's right to use the VM is canceled, and the virtual machine will be free to participate in the next round of the competition.

If we only compete for virtual machines according to the rules mentioned above, in an unsatisfactory situation, some users will satisfy $t_t < t_{vm}$ when each VM is auctioned. For these users having low bids, they may participate in m-round auctions at most. In the worst case, the time complexity of the algorithm is $O(nm)$. In light of the high time complexity of the original auction algorithm, we further propose an improved auction algorithm.

5.2. Improved Auction Algorithms

We define the buyer set as $TASK = \{task_1, task_2, \ldots, task_n\}$. Each task in the set contains two attributes, $task_n \times t$ and $task_n \times p$. The $task_n \times t$ denotes the time required to complete $task_n$, and the $task_n \times p$ denotes the bid for task n to participate in the auction. The elements in the set are arranged in ascending order according to the time $task.t$ required for the task to run.

We define the set of sellers as $VM = \{vm_1, vm_2, \ldots, vm_m\}$. Each VM in the collection contains an attribute $vm_n \times t$, which represents the rental time of virtual machine n. The elements in the set are arranged in ascending order of rental time $vm.t$.

We define the auction set as $CA = \{task_1, task_2, \ldots, task_m\}$. The set is defined as $CA = \{task_1, task_2, \ldots, task_m\}$ for storing tasks that satisfy auction conditions and participate in bidding. The size of the collection is limited, and up to m elements (the number of competing for virtual machines) are stored. The rule of inserting elements into a set is that, when an element is inserted, the elements in the set are arranged in descending order according to the bid of the task by using the semi-insertion sort. When the number of competing tasks is larger than m, only the first m elements are put into the set. The remaining tasks mean that, when competing for the same VM, their price has no advantage (only m VM), so the bidding fails.

Traverse the set VM, auction the virtual machine in turn, such as auction vm_1, traverse the set $TASK$ in turn, and store the elements satisfying $task_n \times t < vm_n.t$ into the set CA in turn. Until the task that does not satisfy the $task_n \times t < vm_n \times t$ condition, the traverse stops, the first task of the collection is taken out and the use right of the Vm_1 is given to the task. Then, continue to auction vm_2, vm_3,..., Vm_m, and finally, m virtual machines are bid on by m users, while the remaining unsuccessful bidders wait for the next round of bidding.

5.3. Auction Compensation Strategy

In this paper, we let $\frac{1}{T_L - T_{offload}}$ denote the bidding price. Although the urgency of task completion is considered, it is unfair. The time-saving task of the offloading computation loses its price advantage in bidding, which is unfair in the auction. Therefore, this paper proposed a compensation strategy for bidding failure. After the task bidding failed, the task was re-priced:

$$price = \frac{1}{T_L - T_{offload} - sX}. \tag{22}$$

In the equation above, s is the length of a time slice, and X is the number of times the task failed to participate in the bidding. One of the meanings of the equation is that each round of bidding for a task fails, which means that the deadline of the task is minus the length of a time slice (the auction frequency is once for each time slice). On the other hand, the derivation of the price on the number of times of failure can be obtained:

$$price' = \frac{S}{(T_L - T_{offload} - sX)^2}. \tag{23}$$

It can be seen from the derivative formula that the derivative increases with an increase of X. The larger the derivative, the faster the price increases. Therefore, the other meaning of the equation is that, with the increase in the number of bidding failures, the task price compensation also increases.

The Algorithm 1 of the whole auction process is described as follows:

Algorithm 1 One-Round Auction Algorithms for Multi-Tasking and Multi-Virtual Machines.

Input: Tasks and VM Participating in Auctions
Output: Tasks of unsuccessful bidding

Calculate the time and bid for each task to rent a virtual machine to get the set $TASK$.
According to the time required by the task, set the lease time of the virtual machine to get the collection VM.
Sort the elements in the collection $TASK$ in ascending order according to the lease time of the virtual machine.
Sort the elements in the collection VM in ascending order according to the rental time of the virtual machine.
for vm in set VM **do**
  for $task$ in set $TASK$ **do**
    if $task_n$.t < $vm_n.t$ **then**
      Add the task to the set CA
    else
      Add the task to the set EL
      auction failed task set
    end if
  end for
  Assign the first task in the collection CA to the virtual machine vm
end for
Add the remaining tasks in the set CA to the set EL

In the multi-task multi-virtual machine bidding model proposed in this paper, the core algorithm is the process of auction competition. Each VM participates in the auction, and it is necessary to perform a half-fold insertion sort on the set CA. The complexity of the half-fold insertion sort algorithm is $O(\log m)$ and m-round auction together, which means that the time complexity of the algorithm is $O(m \log m)$. Compared with the auction algorithm without improvement (time complexity is $O(nm)$, the time complexity will be significantly improved. Due to the explosive growth of Mobile Smart terminals, the number of tasks n participating in computation offloading in the future will certainly be much larger than the number m of VMs providing computing resources. At the same time, compared with the disordered VM participating in the auction, the proposed method of arranging the VM set

and task set in ascending order and then competing will make the difference between the time of the user request and the time of the virtual machine rental small, which means that the virtual machine can be quickly recovered after completing the computing task and participate in the next round of competition. For the server, it will handle more tasks at the same time.

6. Experiments and Results Analysis

In this section, we use MATLAB to simulate the proposed offloading strategy. The simulation scenario is as follows: 1–2 base stations were defined, 30 virtual machines were run on MEC servers at each base station, and the computing power of the virtual machines was set to 10 GHz [40]. The communication bandwidth between the user and the base station was 5 MHz [29], and the user had only one computing task at the same time.

In this paper, the face recognition algorithm [40] has been used as a computation task. The size of the task data is distributed randomly from 1000 KB to 5000 KB, and the computation resources required are set from 200 megacycles to 1000 megacycles. According to the survey, the computing power of the user's mobile device is set to 1.5 GHz to 2.5 GHz random distribution. The rental time of the virtual machines is evenly distributed between the maximum and the minimum according to the task's requirement time. All of the experiment data in this paper are the average of 20 experiments.

6.1. Energy Consumption Comparison between Task Offloading Computation and Local Execution

In this section, we consider a Mobile Edge Computing scenario, which is covered by a base station with 30 virtual machines. The number of tasks involved in the decision-making of offloading was 20–70. Figures 3 and 4 show the number of tasks involved in the offloading computation and the energy consumption of the mobile terminals when the users are 50, 70, and 100 m away from the base station, respectively.

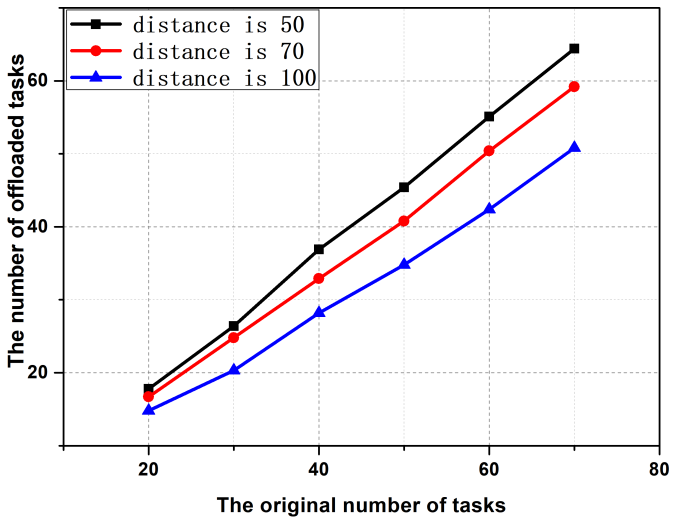

Figure 3. The offloading performance under different distances.

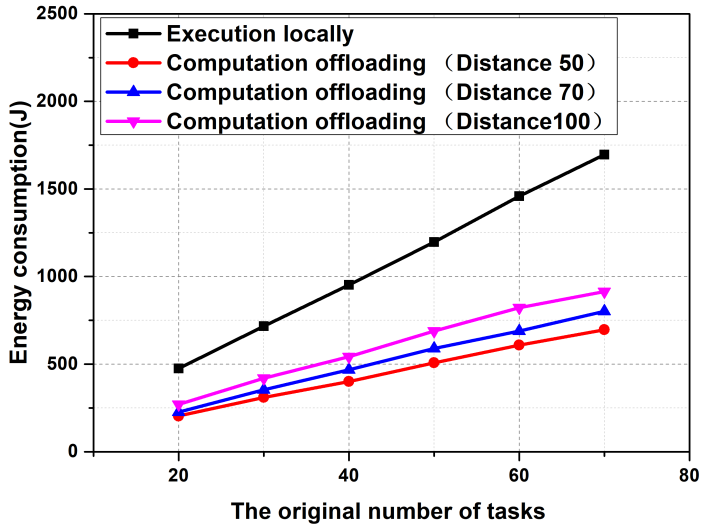

Figure 4. The energy consumption of the mobile device under a different number of tasks.

As can be seen from Figure 3, the closer the mobile device is to the base station, the more tasks it can be offloading. Because of the closer the distance, the lesser the bandwidth loss of wireless communication. This also shows that edge computing is more suitable for computation offloading than traditional centralized cloud computing. Figure 4 shows the energy consumption of the mobile terminal when all tasks are running on the mobile terminal and performing offloading computations. The energy consumption of the offloading computation consists of two parts: one is the energy consumption of tasks that cannot be offloaded and the other is the upload energy consumption of tasks that can be offloaded. As can be seen from Figure 4, offloading computation can save at least 50% of the power for mobile devices, which means that mobile devices have longer endurance.

6.2. Performance Improvement of AHP Algorithms

In this section, we set the server performance parameters closer to the real situation. The performance of the server has its advantages and disadvantages. The specific parameters are shown in Table 2.

Table 2. Server performance parameters.

Server	CPU Computing Speed (GHZ)	Bandwidth (M)	Number of VM
Server1	8	5	40
Server2	10	7	20
Server3	12	4	30

When a mobile device generates a task, the task needs to be offloaded through the offloading decision model, and then the time and energy saving ratio of the task to each server is calculated separately. At the same time, the load status of the server is obtained by monitoring. According to these three factors, the server comprehensive evaluation model proposed in this chapter was used to evaluate the server comprehensively, and the weight of each server was obtained. Figure 5 shows the weight fluctuations of different servers over a period of time.

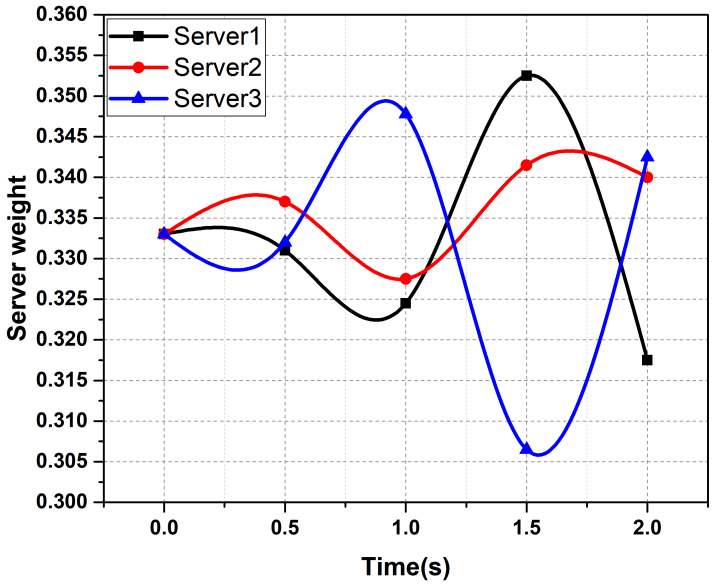

Figure 5. The weight fluctuations of different servers over time.

As can be seen from Figure 5, in the beginning, the weight of each server is the same, which is 0.333. Then, because of the change of server load and the different parameters of each task, the weight of the server also changes over time. At the same time, a comprehensive evaluation based on these three factors can also achieve the effect of load balancing to a certain extent.

Figures 6–8 show the fluctuation of the server load in the case of server selection based on different algorithms. From the figures, it can be seen that server selection based on the improved AHP algorithm could effectively achieve the effect of balancing the server load. There was little difference in the load of the servers, and the load fluctuation of the servers was small.

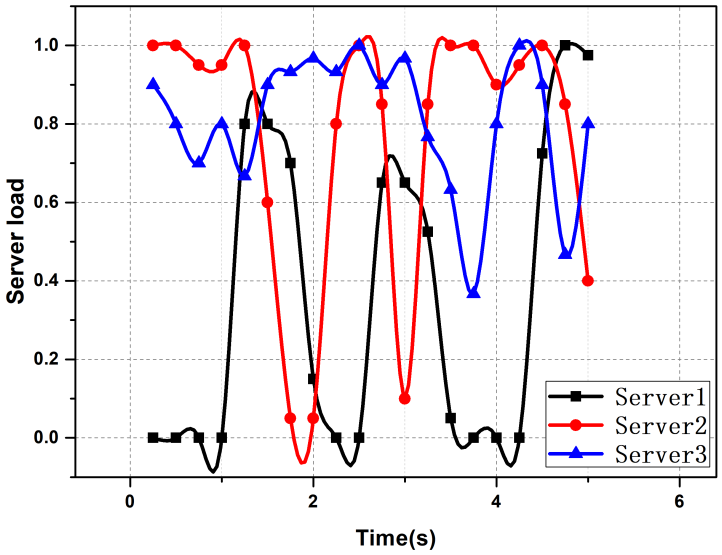

Figure 6. The fluctuation of the server load in the case of server selection based on a random algorithm.

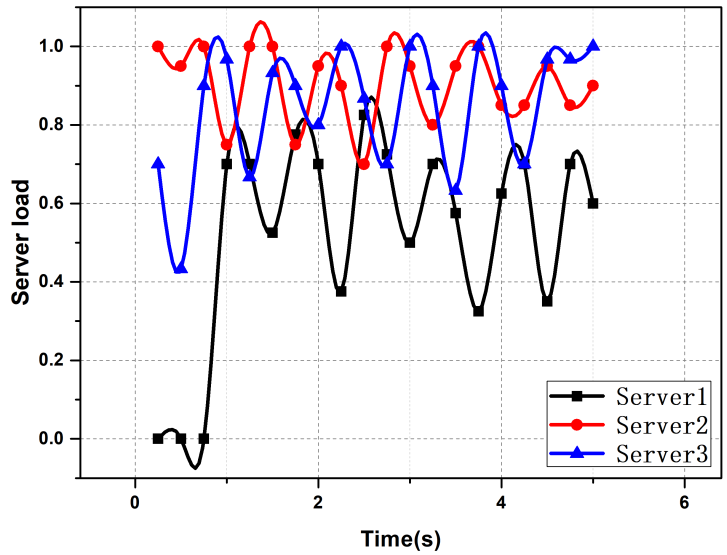

Figure 7. The fluctuation of the server load in the case of server selection based on the polling algorithm.

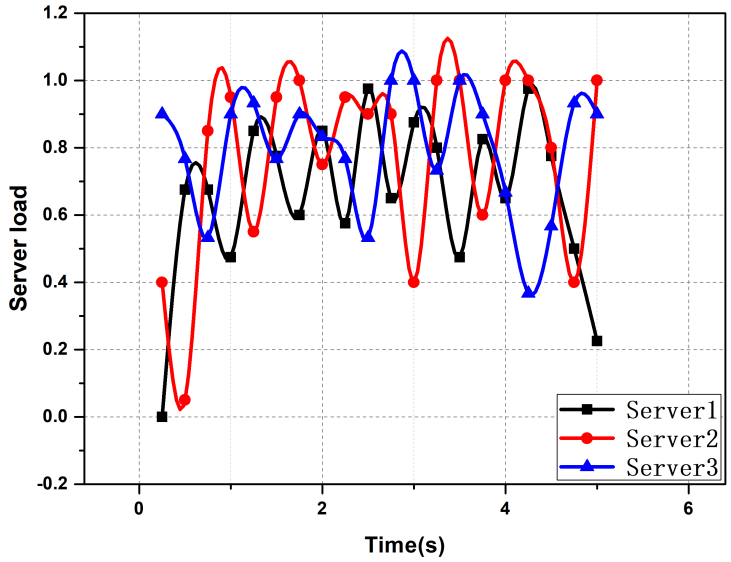

Figure 8. The fluctuation of the server load in the case of server selection based on the improved AHP algorithm.

6.3. Performance Comparison between Improved Auction Algorithm and Classical Auction Algorithm

In the task scheduling algorithm proposed in this paper, time slice s runs through the whole algorithm. The rental time of the virtual machines and task requirement time were taken as integer times of the time slice. The auction frequency is based on the time slice. The auction compensation algorithm also introduces the variable of the time slice, so the value of the time slice directly affects the whole algorithm. In this section, we mainly study the effect of the time slice on task completion and then select the value of the time slice. Task completion degree is defined as the ratio of the number of tasks completed and the total number of tasks completed in a certain period of time. Figure 9 shows the effect of different time slices on task completion.

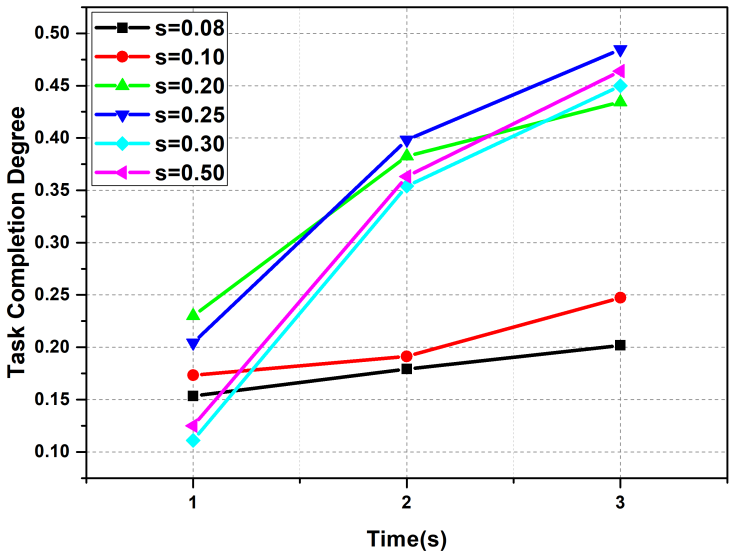

Figure 9. Task completion degree over time.

In theory, the smaller the time slice, the shorter the idle time of the virtual machine and the shorter the waiting time of the task. On the other hand, every round of time slice is auctioned once, and an auction is also costly. If the time interval is very small and the number of tasks generated in each time slice is very small or even nonexistent, then this auction wastes expenses. At the same time, the smaller the time slice within the same time interval, the higher the number of auctions, so the greater the cost. In order to balance the idle time of virtual machines and auction cost, this section evaluated the value of time slices based on task completion degree. The higher the task completion degree, the more coordinated the task generation speed, and the task completion speed. As can be seen from Figure 9, when the value of time slice s was 0.25, the task completion degree was higher, which not only guaranteed the task completion speed but also reduced the idle time of the virtual machine. Therefore, the value of the follow-up experiment time slice in this paper was 0.25.

The experiment scenario set in this section is as follows: there are two base stations, with a total of 60 virtual machines. The number of tasks generated in each round of time slice is 20–70, and 10 rounds of time slice were conducted in one experiment.

Figure 10 shows the idle time of the VM for task scheduling based on the improved auction algorithm and the classical auction algorithm. In this paper, the idle time of the VM was defined as the difference between the time when the VM completes the computing task and the time when the next time slice arrives. As can be seen from the figure, the improved auction algorithm can effectively reduce the idle time of the virtual machine. The reason for this is that, when a VM is auctioned in the classical auction algorithm, all tasks that meet the runtime conditions will compete. The VM may be bid high, but it needs to obtain tasks with little runtime. In short, a long lease of a VM is bound to a task that requires a short run time. In this case, a longer idle time of the virtual machine is generated. In the improved auction algorithm proposed in this paper, virtual machines and tasks are arranged in ascending order according to the rental time and running time before the auction. Auctions are based on the rental time of the virtual machines from short to long. That is to say, virtual machines with a short lease time are competing for tasks with a short demand time, and virtual machines with a long lease time are competing for tasks with a long demand time, which can effectively shorten the idle time of virtual machines. Therefore, the improved auction algorithm proposed in this paper not only has advantages in time complexity but also improves the efficiency of virtual machines.

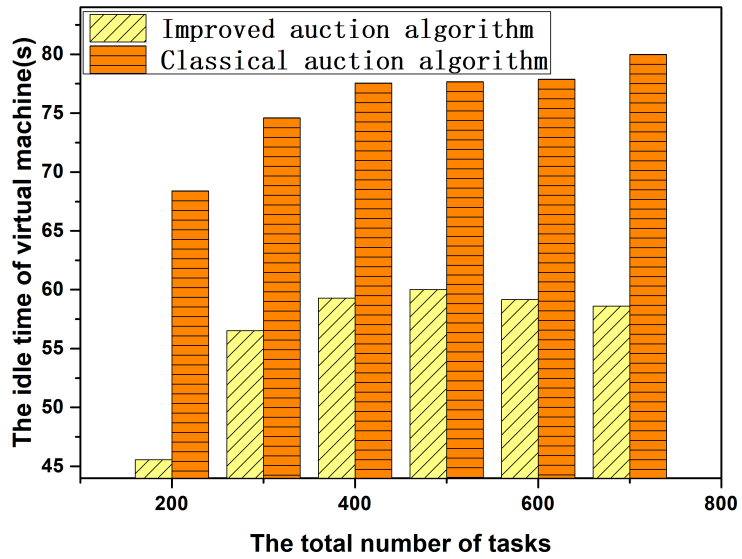

Figure 10. The idle time of the virtual machine under the different total number of tasks.

Figure 11 shows the number of tasks that fail in offloading computation when the number of tasks is much larger than the number of virtual machines when task scheduling is based on improved auction algorithm and classical auction algorithm. In this paper, a failed task is defined as a task that takes more time to compute by offloading than to execute locally. As can be seen from the figure, the number of failed tasks in the scheduling algorithm based on improved auction is lower than that without improvement. The reason is that the idle time of the virtual machine in the scheduling algorithm based on improved auction is lower. That is to say, the utilization of the virtual machine in the algorithm will be higher.

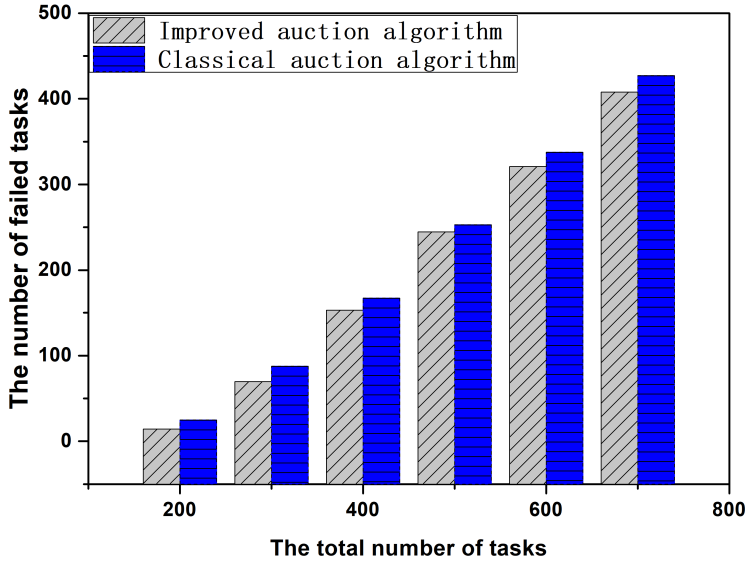

Figure 11. The number of failed tasks under the different total number of tasks.

6.4. Comparison of Time Consumption between Offloading Decision Based on Improved Auction and other Methods

The time scenario set in this section was as follows: there were three base stations, a total of 90 virtual machines, and the number of tasks generated by each round of time slice was 20–70. We have

compared the computation time of four different strategies in different task numbers. In addition, the stability of the four algorithms has been analyzed by the standard deviation.

Figure 12 shows the time overhead of the whole system based on different algorithms. In this paper, the time overhead of the whole system was defined as three parts: the time consumed by locally executed tasks, the time spent on offloading computations, and the time spent on task response (when bidding fails, the response time increases). As can be seen from the figure, offloading-based computation has obvious advantages over local execution in terms of time. As the number of users increases, the time overhead of computation offloading was reduced by nearly 50% compared with that of running it all locally. A cooperative scheduling scheme for mobile cloud computing has been proposed in [41]; the authors design a threshold-based policy which takes the advantages of the low latency of the local cloud and abundant computation resources of the Internet cloud simultaneously. This policy also applies a priority queue in terms of delay requirements of applications. And the time overhead of computation offloading was reduced 25% compared with local computing. Furthermore, in [42], the authors propose three clustering strategies in order to minimize delay and power consumption. The strategies take into account many challenging limitations such as radio resources availability, base stations deployment scenarios, delay constraints, and power consumption limitations. The time overhead of computation offloading was reduced to 22%. However, our proposed computation offloading strategy can achieve a 50% reduction of running time. In addition, we can find from the error bar of computation time that our proposed strategy is the most stable in most cases compared with other methods. At the same time, the computation cost of the system was further reduced by improving the auction algorithm. The experiment data showed that the proposed computation offloading strategy based on an improved auction algorithm could effectively reduce the service delay and improve user satisfaction.

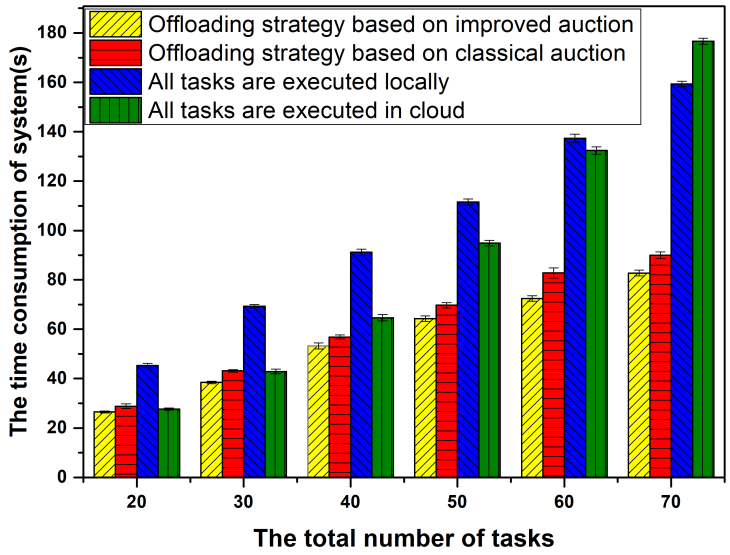

Figure 12. The time consumption of four algorithms over the different number of tasks.

7. Conclusions

In light of the fact that the computation power of mobile devices cannot meet the needs of computation-intensive applications, this paper leverage offloading the tasks of mobile devices onto mobile edge servers to solve the problem of modification, and put forward the strategy of computation offloading for the computation-intensive applications. Considering the requirements of computation tasks and the computation capacity of servers, network bandwidth, and other factors, the basis of offloading decision-making was put forward: the multi-objective decision-making through improved AHP, the comprehensive evaluation of the server, and selecting the appropriate server to offload. A task

scheduling model has been proposed based on the improved auction to optimize the time requirements of the tasks and computation performance of the MEC servers. Simulation results demonstrate that the proposed method can effectively improve the offload task execution speed, reduce the service delay, improve the mobile device battery life, and improve customer satisfaction.

Author Contributions: Conceptualization, J.S. and B.W.; Software, J.H. and X.T.; Supervision, J.S. and B.W.; Validation, X.T., J.H. and X.P.; Writing, J.S. and J.H.

Funding: This research was funded by the National Science and Technology Major Project of China (2017ZX06002005) and the Fundamental Research Funds for the Central Universities of Central South University (2018zzts614).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kumar, K.; Liu, J.; Lu, Y.H.; Bhargava, B. A survey of computation offloading for mobile systems. *Mob. Netw. Appl.* **2013**, *18*, 129–140. [CrossRef]
2. Sharifi, M.; Kafaie, S.; Kashefi, O. A survey and taxonomy of cyber foraging of mobile devices. *IEEE Commun. Surv. Tutor.* **2012**, *14*, 1232–1243. [CrossRef]
3. Cuervo, E.; Balasubramanian, A.; Cho, D.; Wolman, A.; Saroiu, S.; Chandra, R. MAUI: Making smartphones last longer with code offload. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, San Francisco, CA, USA, 15–18 June 2010.
4. Kosta, S.; Aucinas, A.; Hui, P.; Mortier, R.; Zhang, X. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In Proceedings of the T2012 Proceedings IEEE Infocom, Orlando, FL, USA, 25–30 March 2012; pp. 945–953.
5. Xia, F.; Ding, F.; Li, J.; Kong, X.; Yang, L.T.; Ma, J. Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Inf. Syst. Front.* **2014**, *16*, 95–111. [CrossRef]
6. Mao, Y.; Zhang, J.; Letaief, K.B. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 3590–3605. [CrossRef]
7. You, C.; Huang, K.; Chae, H.; Kim, B. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Trans. Wirel. Commun.* **2017**, *16*, 1397–1411. [CrossRef]
8. Deng, M.; Tian, H.; Lyu, X. Adaptive sequential offloading game for multi-cell mobile edge computing. In Proceedings of the 2016 23rd International Conference on Telecommunications (ICT), Thessaloniki, Greece, 16–18 May 2016; pp. 1–5.
9. Fernando, N.; Loke, S.W.; Rahayu, W. Mobile cloud computing: A survey. *Future Gener. Comput. Syst.* **2013**, *29*, 84–106. [CrossRef]
10. Rahman, M.; Gao, J.; Tsai, W.T. Energy Saving Research on Mobile Cloud Computing in 5G. *Chin. J. Comput.* **2017**. . [CrossRef]
11. Sanaei, Z.; Abolfazli, S.; Gani, A.; Buyya, R. Heterogeneity in mobile cloud computing: taxonomy and open challenges. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 369–392. [CrossRef]
12. Orsini, G.; Bade, D.; Lamersdorf, W. Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading. In Proceedings of the Ifip Wireless & Mobile Networking Conference, Colmar, France, 11–13 July 2016.
13. Zhang, W.; Wen, Y.; Wu, D.O. Collaborative Task Execution in Mobile Cloud Computing Under a Stochastic Wireless Channel. *IEEE Trans. Wirel. Commun.* **2015**, *14*, 81–93. [CrossRef]
14. Min, C.; Hao, Y.; Yong, L.; Lai, C.F.; Di, W. On the computation offloading at ad hoc cloudlet: architecture and service modes. *IEEE Commun. Mag.* **2015**, *53*, 18–24.
15. Satyanarayanan, M.; Bahl, P.; Caceres, R.; Davies, N. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Comput.* **2009**, *8*, 14–23. [CrossRef]
16. Ziming, Z.; Fang, L.; Zhiping, C.; Nong, X. Edge Computing:Platforms,Applications and Challenges. *J. Comput. Res. Dev.* **2018**, *55*, 327–337.
17. Sun, X.; Ansari, N. Latency Aware Workload Offloading in the Cloudlet Network. *IEEE Commun. Lett.* **2018**, *21*, 1481–1484. [CrossRef]

18. Shi, Y.; Sun, H.; Cao, J. Edge Computing: A New Computing Model in the Age of Internet of Things. *J. Comput. Res. Dev.* **2017**, *54*, 907–924.
19. Zhang, Y.; Niyato, D.; Wang, P. Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Trans. Mob. Comput.* **2015**, *14*, 2516–2529. [CrossRef]
20. Jia, M.; Cao, J.; Liang, W. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. *IEEE Trans. Cloud Comput.* **2017**, *5*, 725–737. [CrossRef]
21. Sardellitti, S.; Scutari, G.; Barbarossa, S. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Trans. Signal Inf. Process. Over Netw.* **2015**, *1*, 89–103. [CrossRef]
22. Mach, P.; Becvar, Z. Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
23. Taleb, T.; Samdanis, K.; Mada, B.; Flinck, H.; Dutta, S.; Sabella, D. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Architecture & Orchestration. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1657–1681.
24. Yu, Y. Mobile Edge Computing Towards 5G: Vision, Recent Progress, and Open Challenges. *IChina Commun.* **2017**, *13*, 89–99. [CrossRef]
25. Tian, H.; Fan, S.; Lyu, Y. Mobile Edge Computing for 5G Demand. *J. Beijing Univ. Posts Telecommun.* **2017**, *42*, 1–10.
26. Li, W.; Wang, B.; Sheng, J.; Dong, K.; Li, Z. A resource service model in the industrial iot system based on transparent computing. *Sensors* **2018**, *18*, 981. [CrossRef] [PubMed]
27. Ahmed, A.; Ahmed, E. A Survey on Mobile Edge Computing. In Proceedings of the 10th IEEE International Conference on Intelligent Systems and Control, Coimbatore, India, 7–8 January 2016.
28. Wang, S.; Xing, Z.; Yan, Z.; Lin, W.; Yang, J.; Wang, W. A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications. *IEEE Access* **2017**, *5*, 6757–6779. [CrossRef]
29. Chen, X.; Jiao, L.; Li, W.; Fu, X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Netw.* **2016**, *24*, 2795–2808. [CrossRef]
30. Cheng, K.; Teng, Y.; Sun, W.; Liu, A.; Wang, X. Energy-Efficient Joint Offloading and Wireless Resource Allocation Strategy in Multi-MEC Server Systems. In Proceedings of the IEEE ICC, Kansas City, MO, USA, 20–24 May 2018.
31. Shi, C.; Habak, K.; Pandurangan, P.; Ammar, M.; Naik, M.; Zegura, E. COSMOS: Computation offloading as a service for mobile devices. In Proceedings of the IAcm Mobihoc, Philadelphia, PA, USA, 11–14 August 2014.
32. Lyu, X.; Tian, H.; Zhang, P.; Sengul, C. Multiuser joint task offloading and resource optimization in proximate clouds. *IEEE Trans. Veh. Technol.* **2017**, *66*, 3435–3447. [CrossRef]
33. Yu, B.; Pu, L.; Xie, Y. Research on mobile edge computing task offloading and base station association collaborative decision making problem. *J. Comput. Res. Dev.* **2018**, *55*, 537–550.
34. Yang, L.; Cao, J.; Cheng, H.; Ji, Y. Multi-User Computation Partitioning for Latency Sensitive Mobile Cloud Applications. *IEEE Trans. Comput.* **2015**, *64*, 2253–2266. [CrossRef]
35. Huerta-Canepa, G.; Lee, D. An Adaptable Application Offloading Scheme Based on Application Behavior. In Proceedings of the 22nd International Conference on Advanced Information Networking and Applications, Okinawa, Japan, 25–28 March 2008.
36. Wen, Y.; Zhang, W.; Luo, H. Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones. In Proceedings of the IEEE Infocom, Orlando, FL, USA, 25–30 March 2012.
37. Lin, X.; Zhang, H.; Ji, H.; Leung, V.C.M. Joint computation and communication resource allocation in mobile-edge cloud computing networks. In Proceedings of the 2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC), Beijing, China, 23–25 September 2016; pp. 166–171.
38. Huang, D.; Wang, P.; Niyato, D. A dynamic offloading algorithm for mobile computing. *IEEE Trans. Wirel. Commun.* **2012**, *11*, 1991–1995. [CrossRef]
39. Rappaport, T.S. *Wireless Communications: Principles And Practice*; Prentice Hall PTR: Upper Saddle River, NJ, USA, 1996.
40. Muraleedharan, R. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In Proceedings of the 2012 IEEE Symposium on Computers and Communications (ISCC), Cappadocia, Turkey, 1–4 July 2012; pp. 59–66.

41. Zhao, T.; Zhou, S.; Zhao, Y.; Niu, Z. A cooperative scheduling scheme of local cloud and Internet cloud for delay-aware mobile cloud computing. In Proceedings of the IEEE Globecom Workshops, San Diego, CA, USA, 6–10 December 2015; pp. 1–6.
42. Oueis, J.; Strinati, E.C.; Barbarossa, S. Small cell clustering for efficient distributed cloud computing. *IEEE Annu.* **2014**, 1474–1479. [CrossRef]