algorithms

MDPI

Article

Selective Offloading by Exploiting ARIMA-BP for Energy Optimization in Mobile Edge Computing Networks

Ming Zhao and Ke Zhou *

School of Software, Central South University, Tianxin District, Changsha 410075, China; meanzhao@csu.edu.cn
* Correspondence: zhou_ke@csu.edu.cn; Tel.: +86-186-7038-6718

Received: 21 January 2019; Accepted: 19 February 2019; Published: 25 February 2019

Abstract: Mobile Edge Computing (MEC) is an innovative technique, which can provide cloud-computing near mobile devices on the edge of networks. Based on the MEC architecture, this paper proposes an ARIMA-BP-based Selective Offloading (ABSO) strategy, which minimizes the energy consumption of mobile devices while meeting the delay requirements. In ABSO, we exploit an ARIMA-BP model for estimating computation capacity of the edge cloud, and then design a Selective Offloading Algorithm for obtaining offloading strategy. Simulation results reveal that the ABSO can apparently decrease the energy consumption of mobile devices in comparison with other offloading methods.

Keywords: task offloading; mobile edge computing (MEC); ARIMA-BP; energy efficient

1. Introduction

With the popularity of mobile devices, a growing number of mobile applications are striving for computation capacity to provide various services. Nevertheless, mobile devices generally have limited computation resources and short battery lifetime, so some applications which is computationally massive cannot be successfully implemented on mobile devices [1,2]. This conflict between the applications which is scarce of resources and mobile devices whose resources are limited hence presents a formidable challenge.

A new technique called Mobile Cloud Computing (MCC) is likely to solve the aforementioned challenge. Cloud computing [3] offers enormous storage space and computation resources. Through transferring tasks from mobile devices to the resource-rich server, it could overcome the shortage of computation resources in mobile devices. Because mobile devices are far away from the remote cloud, large delays for mobile users has become a critical challenge for cloud computing.

Mobile Edge Computing (MEC) is envisioned as an emerging technique to handle this challenge. It provides cloud-computing service at the mobile edge network close to mobile devices [4]. The main advantages of MEC are as follows: (i) compared with local computing [5], it can avoid the disadvantage of insufficient computation capacity of mobile devices; (ii) compared to Mobile Cloud Computing, it can overcome the large latency that occurs while tasks are transferred to the remote cloud. Thus, MEC presents a better compromise between tasks which is sensitive to delay and tasks is intensive to computation.

Mobile devices can offload their tasks to the edge cloud for computing, or they can choose to finish it locally. The computation capability of the edge cloud is crucial for this issue. In existing works on task offloading, the computation capability of the edge cloud is assumed to be known perfectly and is supposed to be a fixed value. However, the computation capability of the edge cloud varies over time due to the dynamic of the number of computation tasks. For instance, the computation capability of the edge cloud would decline when much more tasks are processed at edge cloud.

In this paper,we focus on the problem of computation capacity of the edge cloud, and propose an ARIMA-BP-based Selective Offloading strategy. The strategy minimizes the total energy consumption by mobile devices and simultaneously meets the tasks' delay constraints. The major contributions of this paper are summarized as follows:

- We propose a multi-device framework for task offloading in MEC networks, and we formulate an optimization problem which minimizes the energy consumption and concurrently meets the delay constraints.
- To solve this problem, we devise an efficient strategy, called ABSO (ARIMA-BP-based Selective Offloading). In ABSO, we propose an ARIMA-BP model to estimate computation capacity of the edge cloud, and then design a Selective Offloading Algorithm for obtaining offloading strategy.

The rest of this paper is organized as follows. In Section 2, we introduce the relevant research works review. The system model is in Section 3. In Section 4, we elaborate an ARIMA-BP model for estimating computation capacity of the edge cloud and further present a Selective Offloading Algorithm to solve offloading problem. Section 5 provide the simulation results. Finally, we come to the conclusion in Section 6.

2. Related Works

Computation offloading has attracted significant attention for both MCC [6,7] and MEC [8,9] in recent years. A lot of works has been done on the task offloading, meanwhile many strategies have been put forward. They can be divided into two classes: (i) latency-based offloading strategies [10–14] and (ii) energy-based offloading strategies [15–19].

The goal of latency-based offloading strategies is to minimize execution time of tasks. Liu et al. [10] formulated a latency minimization problem, then proposed a search algorithm to obtain the optimal task scheduling strategy. Chen et al. [11] designed a distributed task offloading method based on game theory, which can achieve a Nash equilibrium. Yang et al. [12] raised a novel technique based on compiler code analysis, which optimizes the execution time through offloading some of the code on the phone to the cloud dynamically. Mao et al. [13] presented a dynamic offloading strategy based on Lyapunov optimization, in order that execution time can be reduced. Yang et al. [14] designed a heuristic partitioning method to minimize the average execution time. However, these works do not consider energy consumption issues.

The purpose of energy-based offloading policies is to decrease energy consumption of mobile devices. Kamoun et al. [15] proposed a computation offloading strategy, in which the optimization problem is expressed as a Markov decision process. Tao et al. [16] apply KKT conditions to solve the energy minimizing optimization problem, and further propose a request offloading method. Lyu et al. [17] developed a lightweight framework and then designed a selective offloading strategy to minimize the energy consumption of devices. In [18], Huang et al. presented a dynamic offloading strategy through exploiting Lyapunov to reduce energy consumption. Munoz et al. [19] simultaneously optimized transmission time and data volume so as to minimize the mobile devices' energy consumption. However, these works only consider energy consumption and do not pay attention to the delay problem.

Furthermore, few works have addressed the problem of computation offloading jointly considering energy consumption and task execution time. Zhang et al. in [20] introduced an online offloading strategy so as to minimize energy consumption while meeting low delay. Guo et al. in [21] proposed a distributed computation offloading algorithm for MEC networks, in order that it can optimize energy consumption and time delay simultaneously. Neverless, they ignored that the computation capability of the edge cloud (details are provided in Sections 3.1 and 4.2.4) vary with the number of computation tasks for mobile edge computing. Baccarelli et al. [22] analyzed the energy efficiency of big data stream, and then the energy of batteries is measured through a power monitor. Taherizadeh et al. in [23] presented a distributed architecture, which can manage dynamic

IOT environments where edge nodes possible overload because of increased workloads. Although they take into account the situation when workloads changes over time, they cannot describe the load of the cloud relatively accurately.

Various models have been applied to prediction of resource in the cloud environment. PRESS [24] adopts the Markov model and the signature-driven methods for resource prediction, which regards it as a linear time prediction problem. Martin et al. [25] implemented a Recurrent Neural Network to estimate CPU utilization. In this paper, we consider the two aspects together. We develop an ARIMA-BP model to estimate the usage of the edge cloud and then calculate the computation capacity. Since the computation resources of the edge cloud can be obtained by ARIMA-BP in real time, the offloading strategy based on this is relatively accurate and effective.

3. System Model

3.1. Scenario Description

The system model is shown in Figure 1. There is a mobile-edge computing system, including multiple mobile devices which have computation tasks and an edge cloud. Edge cloud is a relatively large data center with computation resources, which provides computing capacities in proximity to mobile devices. Its computing capacity is smaller than that of cloud computing and larger than that of mobile devices. There is a wireless base station, through which mobile devices could offload their computing tasks to the edge cloud. Suppose there are N mobile devices and K computation tasks in the MEC system. Then, we define the set of devices and tasks as $N = \{1, 2, \ldots, N\}$ and $K = \{1, 2, \ldots, K\}$. Suppose time is divided into equal-sized slots. We assume that a mobile device can only request one task during a time slot, and different mobile devices can request the same task. Suppose that the mobile devices are static in the system.

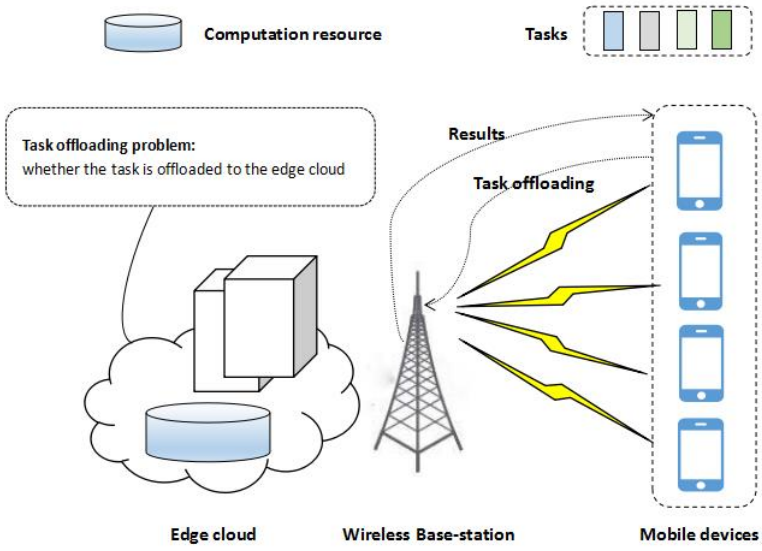


Figure 1. A multi-user system for MEC.

We define $u_{n,k}$ as the task k that mobile device n requests, and adopt three parameters representing the computation task. For task $u_{n,k}$, its requirement is denoted as a tuple $\{w_k, s_k, D_{n,k}\}$, where w_k is the amount of computation resource required for $u_{n,k}$, (i.e., the number of CPU cycles required to finish the task), and s_k is the data size of $u_{n,k}$, i.e., the amount of data content (e.g., the program codes and input parameters) which is transferred to the edge cloud, and $D_{n,k}$ represents completion time requirement for the task $u_{n,k}$. Furthermore, due to the limitation of computation resources in the edge cloud, we denote the computation capacity of the edge cloud is c_s.

3.2. Communication Model

When mobile device n transfers task k to the edge cloud, the uplink data transfer rate for mobile device n is shown below:

$$r_n = Blog_2(1 + \frac{P_n H_n}{\sigma + I_n}) \tag{1}$$

where B is channel bandwidth, σ is the noise power, P_n is the transmitted power of mobile devices, and H_n denotes the channel gain between edge cloud and mobile device n. I_n indicates the interference between edge cloud and mobile device n.

Similar to many studies [16–18,26], we neglect the downlink transmission delay, because the output data of many applications is usually much smaller than the input data.

3.3. Computation Model

Considering the computation tasks can be processed locally or at the edge cloud, we will introduce this in detail.

3.3.1. Local Computing

Let f_n^l denote the CPU computation capacity of mobile device n. The local completion latency of $u_{n,k}$ can be given below:

$$T_{n,k}^l = \frac{w_k}{f_n^l} \tag{2}$$

The computational energy consumption is expressed as:

$$E_{n,k}^l = w_k \epsilon^l \tag{3}$$

where ϵ^l is the coefficient. According to work [27], ϵ^l can be obtained by:

$$\epsilon^l = k(f_n^l)^2 \tag{4}$$

where k is the energy coefficient. We set $k = 10^{-11}$ according to the work [28].

3.3.2. Mobile Edge Cloud Computing

If mobile device n choose to offload task k to the edge cloud, the process contains task transmission and task execution. The total completion time has two parts: (i) task transmission time $T_{n,k}^{tra}$ (ii) task process time $T_{n,k}^{pro}$.

The total latency of $u_{n,k}$ for mobile-edge computing is obtained by:

$$T_{n,k}^c = T_{n,k}^{tra} + T_{n,k}^{pro} = \frac{s_k}{r_n} + \frac{w_k}{f_{n,k}^c} \tag{5}$$

where $f_{n,k}^c$ denotes the computation resource of the edge cloud assigned to $u_{n,k}$. Meanwhile, the energy consumption of mobile device n can be defined as $E_{n,k}^c$, which only consists of the transmission energy consumption while transferring tasks to the edge cloud. The energy consumption of mobile device as shown below:

$$E_{n,k}^c = P_n T_{n,k}^{tra} = P_n \frac{s_k}{r_n} \tag{6}$$

3.4. Problem Formulation

For the problem of computational task offloading, the variable $x_{n,k} \in \{0, 1\}$ is defined, in order to express whether to offload the task $u_{n,k}$ to the edge cloud $(x_{n,k} = 1)$ or not $(x_{n,k} = 0)$.

Based on the aforementioned descriptions, the goal of the presented problem is to minimize the total energy consumption consumed by mobile devices, can be shown below:

$$\begin{aligned} &\min_{x_{n,k}} \sum E_{n,k}, \\ &s.t. \quad C1: \sum_{n=1}^{N} x_{n,k} f_{n,k}^{c} \le c_s, \quad \forall k \in K \\ &\qquad C2: T_{n,k} \le D_{n,k}, \qquad \forall n \in N, \forall k \in K \ . \\ &\qquad C3: x_{n,k} \in \{0,1\}, \qquad \forall n \in N, \forall k \in K \end{aligned} \tag{7}$$

The constraint condition (C1) guarantees the total computation resources required to offload tasks must be less than total capacity of the edge cloud during each time slot. The constraint (C2) shows that the task $u_{n,k}$ must be completed within the time limit. The constraint (C3) indicates that the decision variable for task offloading is a binary data.

The objective function of our presented problem is defined as follows:

$$E_{n,k} = \begin{cases} E_{n,k}^{l}, & x_{n,k} = 0 \\ E_{n,k}^{c}, & x_{n,k} = 1 \end{cases} \tag{8}$$

4. ARIMA-BP-Based Selective Offloading Strategy

4.1. Research Motivation

In MEC, the edge cloud has relatively large storage and computation resources. Compared with the computation capacity of mobile devices themselves, the computation capacity of the edge cloud is huge. Mobile edge computing brings computation resources to the mobile devices and reduces energy consumption in comparison with local execution, but leading to increased duration of transmission. Hence we need to consider both energy consumption and time delay comprehensively. For this issue, the computation capacity of the edge cloud is a key factor.

In the experimental environment, we usually set the computation capacity of the edge cloud as a fixed value. However, the computation capacity of the edge cloud cannot be static in the process of task offloading. As the number of tasks processed at the edge cloud increases, its remaining computation resources will decrease, reducing its computation capacity. When the number of tasks processed at the edge cloud decrease, its computation capacity will increase. Therefore, It is very critical and effective to obtain the computation capacity of the edge cloud accurately for task offloading.

In practice, the computation capacity of the edge cloud is hard to represent and is not available ahead of time due to the dynamic of tasks. We hence can only get its estimated value by prediction. The increasingly advanced big data analysis technology makes it possible to accurately predict computation resources. By predicting the resource usage of the edge cloud, we can get the estimated value of its the computation capacity.

4.2. Estimation for Computation Capacity of Edge Cloud by ARIMA-BP

Resource prediction in cloud can be regarded as a time-series prediction problem. So far, the weighted average method has been widely used in the existing models to predict the trend of resource change. However, the compound model method for predicting CPU resource by time-series analysis techniques is rarely used. Moreover, it is found that the time-series process of CPU resource prediction is composed of linear structure and non-linear structure by researching.

The time-series prediction method mainly adopts a linear prediction model, which makes the method unable to process nonlinear data more accurately. Compared with the conventional time-series model, BP neural network is an efficacious nonlinear modeling measure. It has obvious advantages in handling the data with inconspicuous characteristics that with much randomness and nonlinearity, but it is a little poorer in processing the linear data compared to the conventional time-series model. Therefore, the combination method of time-series prediction and BP neural network can improve and perfect the prediction results.

4.2.1. Time Series Prediction

ARIMA Model

The Auto-Regressive Integrated Moving Average (ARIMA) model is composed of the auto-regressive model (AR) and the moving average model (MA). It is the most commonly used non-stationary time-series prediction model. The basic idea of the modeling is to use the difference method to smooth the non-stationary time-series. The time-series is predicted and analyzed by observing the three parameters of the correlation function truncation and trailing characteristic auto-regressive order (p), difference frequency (d) and moving average order (q). The structure of the ARIMA model is as follows:

$$x'_{t+1} = \sum_{i=0}^{p-1} a_i x_{t-i} + \sum_{i=0}^{q-1} b_i \epsilon_{t-i} \tag{9}$$

where x_i is the real value of time slot i. p is the number of autoregressive terms, and q represents the order of moving average. a, b are the relevant weights of this model. The term ϵ are the error terminologies relevance to the model.

Prediction of Resource Usage Using Fractional Differencing

PRESS [24] uses the Markov model and the signature-driven methods for resource prediction. They divided the use of resource measures into different intervals, then further calculate the conversion probability matrix. Finally they estimated the probability of the next interval by exploiting the Chapman-Kolmogorov equations:

$$\pi_t = \pi_{t-1} P \tag{10}$$

where π_t and π_{t-1} indicate the probability at time instant t and $t-1$. Moreover, AGILE [29] improves PRESS through exploiting wavelets to predict usage of resources in cloud. Also, ARIMA is used to estimate the resource usage in the cloud.

The methods introduced above suppose that time-series are not dynamically changing, and there is no memory. Neverless, cloud environments is constantly changing, and its workloads are highly dynamic. The real workload conditions cannot be catched efficiently by traditional time-series models. It creates a need for a complicate time-series model that can be memorized over time. Thus, the method of rescaled range analysis [30] can be employed for fractional difference [31] in this paper. Fractional difference can be expressed as the following formula.

$$x'_t = x_t - d x_{t-1} + \frac{d(d-1)}{2!} x_{t-2} - \frac{d(d-1)(d-2)}{3!} x_{t-3} \tag{11}$$

where x_i denotes the value during interval i, x'_t represents the value of x_t which is obtained after fractional difference. d denotes the parameter related to the difference. Then the method is applied to the analysis for multi-step ahead prediction of the above-mentioned methods.

The different models will be validated on Google cluster trace. Google cluster trace is a cluster usage dataset and is divided into six tables. Among them, the table of task resource usage offers the usage of various resources (such as memory, disk, and CPU) during different time intervals. Here, we analyze CPU usage at different time intervals.

Table 1 shows the RMSE (Root Mean Squared Error) for the multi-step ahead predictions of the above methods (PRESS, AGILE and ARIMA) without and with fractional difference. We can observe that fractional difference contributes to a significant improvement in the prediction results for all models. As the increase of prediction horizon, the RMSE of the prediction for CPU usage increases. Because errors accumulate gradually in the process of prediction. We can also observe that ARIMA has smaller RMSE than other models. Figure 2. shows the results of predictions of PRESS, AGILE and ARIMA on CPU usage. It can be seen from the figure that ARIMA reflects the trend of changes in

CPU usage more accurately than PRESS and AGILE. Thus we adopt ARIMA model using fractional difference for time-series prediction.

Table 1. RMSE of Prediction on CPU Usage.

	Without Fractional Difference			With Fractional Difference		
	30 Step Ahead	**60 Step Ahead**	**90 Step Ahead**	**30 Step Ahead**	**60 Step Ahead**	**90 Step Ahead**
PRESS	0.0294	0.0419	0.0657	0.0167	0.0260	0.0512
AGILE	0.0247	0.0386	0.0526	0.0150	0.0198	0.0453
ARIMA	0.0213	0.0340	0.0463	0.0122	0.0159	0.0306

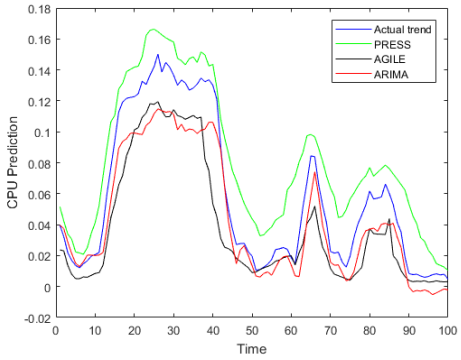


Figure 2. Prediction on CPU usage for different models.

4.2.2. Modification of the Residual Error Correction by BP Neural Network

Back Propagation

Back Propagation (BP) is a neural network trained by error back propagation algorithm, which is multi-layer feedforward. It is the most widely used neural network training method in combination with optimization method such as gradient descent.

BP neural network consists of two processes, including forward propagation of information and back propagation of errors. Using the error of the output layer to estimate the error of the direct predecessor layer of the output layer, and then using this error to estimate the error of the previous layer. It can continuously learn and store a large number of mappings about input-output modes without describing the equation of this relationship directly.

Modifying the Residual Error Correction

CPU usage has been fitted by ARIMA prediction techniques using fractional difference, then we use Back Propagation to obtain the nonlinear residual. It involves following steps:

1. Compute the remaining sequence used by the CPU.
2. Normalize the residual sequence.
3. Define the structure of the BP neural network.
4. Define the trained parameters of the BP neural network.
5. Obtain the residual sequence by the network simulation.

4.2.3. Prediction of CPU Usage by ARIMA-BP

According to the ARIMA-BP method, the final prediction data of CPU usage can be got through adding the original data on CPU usage which is estimated by exploiting ARIMA using fractional difference and BP neural network is used to predict the residual sequence.

Table 2 indicates the RMSE of the two methods (ARIMA, ARIMA-BP), which is obtained by exploiting fractional difference. We can see that the RMSE of ARIMA-BP is lower than ARIMA leading to more accurate predictions. Figure 3 shows the results of predictions on CPU usage by ARIMA and

ARIMA-BP. It can be observed that the ARIMA-BP predictions are much nearer to the actual values of CPU usage than ARIMA predictions. Therefore, we adopt the ARIMA-BP model using fractional difference to predict the computation capacity of the edge cloud in this paper.

Table 2. RMSE of ARIMA and ARIMA-BP on CPU Usage.

	30 Step Ahead	60 Step Ahead	90 Step Ahead
ARIMA	0.0139	0.0163	0.0317
ARIMA-BP	0.0120	0.0126	0.0224

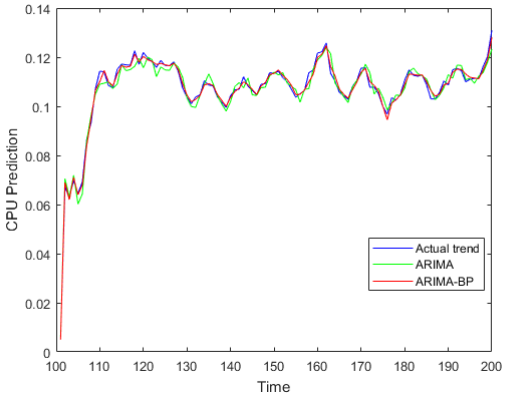


Figure 3. Prediction on CPU usage by ARIMA and ARIMA-BP.

4.2.4. ARIMA-BP for Prediction of Computation Capacity in Edge Cloud

In mobile edge computing, computation capacity of the edge cloud is usually represented by computation resources f^c that can be allocated to mobile devices. We use $f^c(t)$ indicate computation resources that edge cloud have been allocated to mobile devices during the time slot $[t - \Delta t, t]$, then the remaining computation resources of the edge cloud is $\tilde{f}^c(t) = c_s - f^c(t)$. Suppose that m tasks $u_{n,k}$ are processed at the edge cloud during the time slot $[t - \Delta t, t]$.We have

$$f^c(t) = \sum_{i=1}^{m} f^c_{n,k}, \quad \forall n \in N, \forall k \in K \tag{12}$$

We use the ARIMA-BP model for predicting the usage of computation resources of the edge cloud during a time slot. First, we build the ARIMA model. The usage of computation resources to be estimated during $[t, t + \Delta t]$, denoted by $\hat{f}^c(t + \Delta t)$, is expressed below:

$$\hat{f}^c(t + \Delta t) = a_0 f^c(t) + a_1 f^c(t - \Delta t) + \cdots + a_{p-1} f^c(t - (p-1)\Delta t) + b_0 \epsilon_t + \cdots + b_{q-1} \epsilon_{t-(q-1)\Delta t} \tag{13}$$

Then, we apply fractional difference method for catching long-term dependencies in the data. Finally, we use BP neural network to correct the residual error.

4.3. A Selective Offloading Algorithm

Based on the estimation of computation resources utilization of the edge cloud, we can obtain the remaining computation resources of the edge cloud by $\tilde{f}^c(t) = c_s - f^c(t)$. Then we develop a selective offloading algorithm.

Initially, all mobile devices need to set their initial decisions during a time slot.

Then, we lead into the condition below for prioritizing urgent tasks that need to be offloaded. This is known as selective offloading.

Condition 1. *if* $T^l_{n,k} > D^l_{n,k}$, *the task* $u_{n,k}$ *offloads to the edge cloud for processing.*

When Condition 1 is satisfied, it is usually the devices which are constrained to resource, and their tasks are sensitive to latency. We give priority to offload these tasks. Because computing locally cannot meet tasks' delay constraints. (i.e., $T^l_{n,k} > D^l_{n,k}$).

Tasks must be accomplished within the specified time (i.e., the delay requirements) $D_{n,k}$. Therefore, we can calculate the minimum computation resources needed for each task under the latency requirements when computed at edge cloud by:

$$f^{min}_{n,k} = \frac{w_k}{D_{n,k} - T_{n,k}} \tag{14}$$

The edge cloud server gets the remaining computation resources $\tilde{f}^c(t)$ and broadcasts it to the mobile devices. Mobile devices which do not meet Condition 1 will be judged by Condition 2 after they get $\tilde{f}^c(t)$.

Condition 2. *if* $\tilde{f}^c(t) \geq f^{min}_{n,k}$ *, the task merges into the set* $\Theta(t)$.

For all tasks that satisfy Condition 2, they will merge into the set $\Theta(t)$, since the remaining computation resources of the edge cloud can meet the minimum resources they need (i.e., $\tilde{f}^c(t) \geq f^{min}_{n,k}$), and the tasks not satisfying the Condition 2 still to judge in the next time slot.

While $\Theta(t) \neq \varnothing$, tasks in $\Theta(t)$ compete for decision-making opportunity by sending requests to the edge cloud. During each time slot, only one task can get the opportunity to make the decision. Suppose the mobile device i obtain the chance, it shows to other devices that it gains the chance for decision-making by broadcasting. Then, the energy consumption of local computing and edge cloud processing of task $u_{n,k}$ will be compared. If the energy consumption of the edge cloud computing is smaller (i.e., $E^c_{n,k} < E^l_{n,k}$), the task would be offloaded to the edge cloud for processing; If not, the task would be processed locally. For mobile devices don't get the opportunity, they will continue to judge the Condition 2 in the next time slot. Algorithm 1 shows the process of getting offloading decision.

Algorithm 1 A Selective Offloading Algorithm

loop
  initialize each task computation offloading decision $x_{n,k} = 0$;
  for each task $u_{n,k}$ **do**
    compare local processing time $T^l_{n,k}$ with the latency requirements $D_{n,k}$;
    if $T^l_{n,k} > D_{n,k}$ **then**
      update $x_{n,k} = 1$;
    else
      calculate $f^{min}_{n,k}$ according to (16) and compare $f^{min}_{n,k}$ with $\tilde{f}^c(t)$;
      if $\tilde{f}^c(t) \geq f^{min}_{n,k}$ **then**
        merge into the set $\Theta(t)$;
      else
        judge Condition 2 again for next time slot ;
      end if
    end if
  end for
  while $\Theta(t) \neq \varnothing$ **do**
    tasks in $\Theta(t)$ send requests to the edge cloud to compete for the chance of decision-making;
    if task $u_{n,k}$ gain the chance **then**
      compare the energy consumption of local execution $E^l_{n,k}$ and edge cloud process $E^c_{n,k}$;
      if $E^c_{n,k} < E^l_{n,k}$ **then**
        update $x_{n,k} = 1$;
      else
        set $x_{n,k} = 0$;
      end if
    else
      judge Condition 2 again for next time slot;
    end if
  end while
end loop

5. Simulation Results

5.1. Experiment Setup

We do simulations to prove the performance optimization that our presented computation offloading strategy can bring. An edge computing system is considered in this paper. The system is composed of the edge cloud and mobile devices, in which mobile devices have relatively intensive computation tasks. The simulation settings are listing as follows. We set the transmission bandwidth $B = 5$ MHz and transmitting power $P_n = 0.5$ W, and the relevant noise power $\sigma = 10^{-10}$. The channel gain H_n is denoted by $H_n = 127 + 30 \times \log d$, where d is the distance between edge cloud and mobile device n.

By default, the mobile devices have 50 computation tasks in total. For task $u_{n,k}$, the number of CPU cycles w_k subjects to normal distribution, which its average is 1000 M Cycles. The data size s_k obeys normal distribution, and its average value is 3 MB, and the completion time $D_{n,k}$ is generated by a normal distribution with an average of 1 s. Furthermore, the computational capability of mobile devices is assumed 1 GHz. Relative to the research field, the above parameters are typical values, and will not affect the results obtained.

5.2. Task Offloading Evaluation

To evaluate the offloading, we compare our proposed scheme ABSO with following task offloading strategies.

- Local execution: All computational tasks are processed locally.
- Full offloading: All computational tasks are executed on edge cloud by offloading.
- Branch and Bound Algorithm (BBA) : The objective function is transformed into a question that is binary and linear, and it can be resolved availably by using a branch-and-bound algorithm.

We observe the energy consumption of mobile devices and the completion time of tasks under various quantities of computation tasks in Figure 4. Obviously, we can see that the completion time and energy consumption will gradually increase as computational tasks increase. We can see that local execution consumes the most energy, because the CPU computation capacity of a mobile device is relatively small. Also, full offloading has higher completion time compared to others; this can mainly be attributed to the reason that full offloading needs to transmit data to the edge cloud before tasks can be processed. Moreover, it can be observed that the presented offloading strategy has better results compared to others on decreasing the expenditure of energy and time, and as the number of tasks increases, its advantages become more obvious. For the metric of the completion time, ABSO can achieve up-to 14% and 26%, and 7% performance improvement over the local execution, full offloading, and BBA, respectively. For the metric of the energy consumption, ABSO can reach up to 42% and 22%, and 9% decrement of energy compared to the local execution, full offloading, and BBA, respectively.

Figure 5a depicts the influence of data size of tasks on the energy consumption. We can observe that our presented ABSO outperforms other methods. The energy consumption of local execution is relatively steady, and it is smaller than the value of full offloading as s_k increases. We can also see that as the data size of tasks increase, the energy consumption starts to increase. That's because with the data size of tasks rise, the communication time become larger, and the energy consumption of the edge cloud computing become larger than that of before. In the meantime, the difference of energy consumption between different strategies is subtle while the data size of tasks is small. Therefore, we infer that tasks with small amount of data are processed in the edge cloud when the other conditions of tasks are consistent. On the contrary, tasks would be supposed to processed locally.

The energy consumption with the different number of CPU cycles w_k are shown in Figure 5b. According to the comparison, it can be seen that the energy consumption of our presented method ABSO is most optimal. It can be observed that the energy consumption of full offloading is relatively steady with little fluctuation. Also, as the increasing of required CPU cycles, the energy consumption

are getting larger. This is because that when the required CPU cycles of tasks rise, the energy consumption of local execution turns larger. From Figure 5b, we can conclude that in the case of the consistent amount data size of tasks, the needed CPU cycles is small and tasks ought to be processed locally. Conversely, the CPU cycles needed is large, tasks are transferred to the edge cloud for processing.

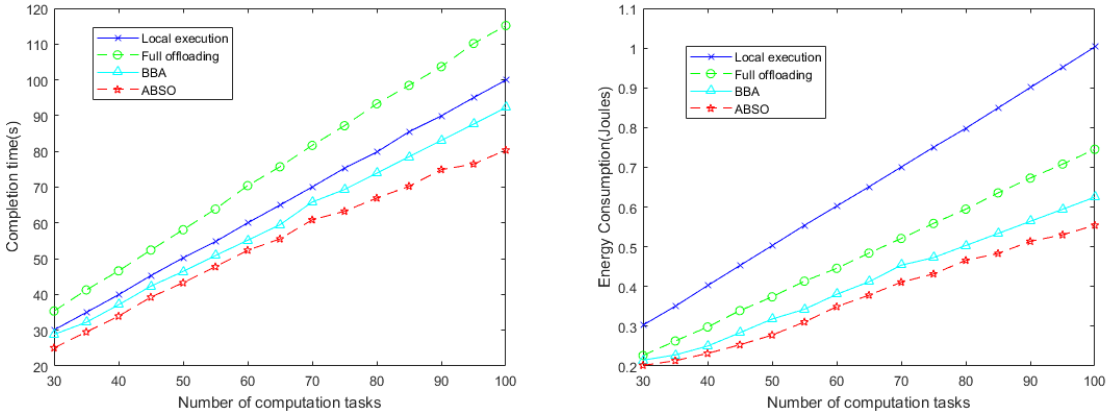
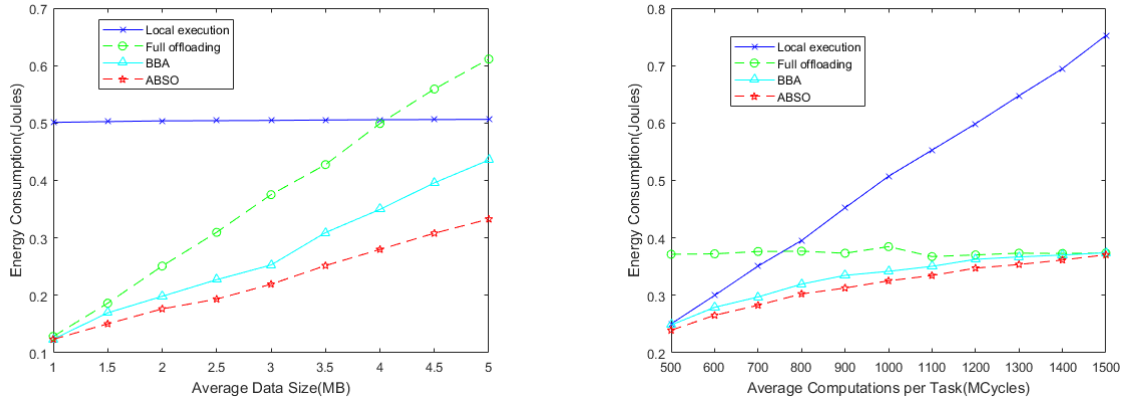
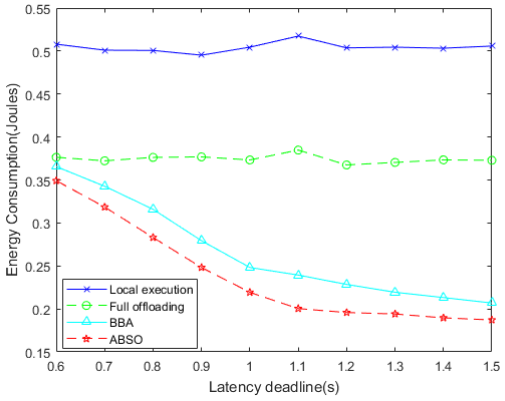


Figure 4. Energy consumption of mobile devices and completion time of tasks under various quantities of computation tasks.



(**a**) (**b**)



(**c**)

Figure 5. Comparison of energy consumption for different offloading strategies. (**a**) Effect of task size on energy consumption; (**b**) Effect of required computation cycles on energy consumption; (**c**) Effect of latency deadline on energy consumption.

In Figure 5c, we consider the energy consumption as the variance of latency requirements. The energy consumption of local computing and full offloading is little changed. This is because that latency requirements are not taken into account when processing locally and offloading entirely, and energy consumption is independent of latency requirements. Obviously, offloading tasks to the edge cloud is helpful to cut down energy consumption. While tasks' delay deadlines are small, the performance of our proposed ABSO resembles full offloading and BBA. Since most tasks are offloaded to the edge cloud for processing. It seems the energy consumption of ABSO is plunged and tends to be gentle after 1.1 s. This is because that as latency turns larger, the presented ABSO uses an optimized strategy, in order that the energy consumption can be reduced slightly.

6. Conclusions

In this paper, we committed to designing an energy-efficient offloading strategy, which also meets the constraints of users' delay requirements. We propose an offloading strategy named the ARIMA-BP-based Selective Offloading (ABSO) strategy and use the two-step framework for designing it. We first design an ARIMA-BP model to estimate computation capacity of the edge cloud. Then, we propose a Selective Offloading Algorithm for obtaining offloading strategy. The simulation results show that the presented ABSO can obtain superior performance on reducing energy consumption of mobile devices. In future research work, we plan that more common situations will be taken into account, such as mobile user movement during use.

Author Contributions: Conceptualization, M.Z.; Data curation, K.Z.; Formal analysis, K.Z.; Funding acquisition, M.Z.; Investigation, K.Z.; Methodology, K.Z.; Project administration, M.Z.; Resources, M.Z.; Software, K.Z.; Supervision, M.Z.; Validation, M.Z.; Visualization, K.Z.; Writing—original draft, K.Z.; Writing—review and editing, M.Z.

Funding: This research received no external funding.

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

References

1. Fu, Z.; Ren, K.; Shu, J.; Sun, X.; Huang, F. Enabling personalized search over encrypted outsourced data with efficiency improvement. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 2546–2559. [CrossRef]
2. Kumar, K.; Lu, Y.H. Cloud computing for mobile users: Can offloading computation save energy. *IEEE Comput.* **2010**, *43*, 51–56. [CrossRef]
3. Abolfazli, S. Cloud-based Augmentation for MobileDevices: Motivation, Taxonomies, and Open Challenges. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 337–368. [CrossRef]
4. Milan, P.; Jerome, J.; Valerie, Y.; Sadayuki, A. Mobile-Edge Computing Introductory Technical White Paper. White Paper. Available online: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf (accessed on 14 September 2018).
5. Chen, M.; Hao, Y.; Li, Y.; Lai, C.-F.; Wu, D. On the computation offloading at ad hoc cloudlet: Architecture and service modes. *IEEE Commun. Mag.* **2015**, *53*, 18–24. [CrossRef]
6. Fernando, N.; Loke, S.W.; Rahayu, W. Mobile cloud computing: A survey. *Future Gener. Comput. Syst.* **2013**, *29*, 84–106. [CrossRef]
7. Buyya, R.; Yeo, C.S.; Venugopal, S.; Broberg, J.; Brandic, I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.* **2009**, *25*, 599–616. [CrossRef]
8. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. Mobile edge computing: Survey and research outlook. *arXiv* **2017**, arXiv:1701.01090.
9. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
10. Liu, J.; Mao, Y.; Zhang, J.; Letaief, K.B. Delay-optimal computation task scheduling for mobile-edge computing systems. In Proceedings of the 2016 IEEE International Symposium on Information Theory (ISIT), Barcelona, Spain, 10–15 July 2016; pp. 1451–1455.

11. 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]
12. Yang, S.; Kwon, D.; Yi, H.; Cho, Y.; Kwon, Y.; Paek, Y. Techniques to Minimize State Transfer Costs for Dynamic Execution Offloading in Mobile Cloud Computing. *IEEE. Trans. Mob. Comput.* **2014**, *13*, 2648–2660. [CrossRef]
13. Mao, Y.; Zhang, J.; Letaief, K.B. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE J. Sel. Areas Common.* **2016**, *34*, 3590–3505. [CrossRef]
14. 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]
15. Kamoun, M.; Labidi, W.; Sarkiss, M. Joint resource allocation and offloading strategies in cloud enabled cellular networks. In Proceedings of the IEEE International Conference on Communications (ICC), London, UK, 8–12 June 2015; pp. 5529–5534.
16. Tao, X.; Ota, K.; Dong, M.; Qi, H.; Li, K. Performance Guaranteed Computation Offloading for Mobile-Edge Cloud Computing. *IEEE Commun. Mag.* **2017**, *6*, 2162–2345. [CrossRef]
17. Lyu, X.; Tian, H.; Jiang, L.; Vinel, A.; Maharjan, S.; Gjessing, S.; Zhang, Y. Selective Offloading in Mobile Edge Computing for the Green Internet of Things. *IEEE Netw.* **2018**, *32*, 54–60. [CrossRef]
18. Huang, D.; Wang, P.; Niyato, D. A Dynamic Offloading Algorithm for Mobile Computing. *IEEE Trans. Wirel. Commun.* **2012**, *11*, 1991–1995. [CrossRef]
19. Munoz, O.; Pascual-Iserte, A.; Vidal, J. Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. *IEEE Trans. Veh. Technol.* **2015**, *64*, 4738–4755. [CrossRef]
20. Zhang, W.; Wen, Y.; Chen, H.-H. Toward transcoding as a service: Energy-efficient offloading policy for green mobile cloud. *IEEE Netw.* **2014**, *28*, 67–73. [CrossRef]
21. Guo, J.; Zhang, H.; Yang, L.; Ji, H.; Li, X. Decentralized Computation Offloading in Mobile Edge Computing Empowered Small-Cell Networks. In Proceedings of the 2017 IEEE Globecom Workshops (GC Wkshps), Singapore, 4–8 December 2017; pp. 1–6.
22. Baccarelli, E.; Cordeschi, N.; Mei, A.; Panella, M.; Shojafar, M.; Stefa, J. Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: Review, challenges, and a case study. *IEEE Access* **2016**, *30*, 54–61. [CrossRef]
23. Taherizadeh, S.; Stankovski, V.; Grobelnik, M. A Capillary Computing Architecture for Dynamic Internet of Things: Orchestration of Microservices from Edge Devices to Fog and Cloud Providers. *Sensors* **2018**, *18*, 2938. [CrossRef] [PubMed]
24. Gong, Z.; Gu, X.; Wilkes, J. PRESS: PRedictive Elastic Resource Scaling for cloud systems. In Proceedings of the International Conference on Network and Service Management (CNSM), Niagara Falls, ON, Canada, 25–29 October 2010; pp. 9–16.
25. Duggan, M.; Mason, K.; Duggan, J. Predicting host CPU utilization in cloud computing using recurrent neural networks. In Proceedings of the 2017 12th International Conference for Internet Technology and Secured Transactions (ICITST), Cambridge, UK, 11–14 December 2017.
26. Hao, Y.; Chen, M.; Hu, L.; Hossain, M.S.; Ghoniem, A. Energy Efficient Task Caching and Offloading for Mobile Edge Computing. *IEEE Access* **2018**, *6*, 11365–11373. [CrossRef]
27. Wen, Y.; Zhang, W.; Luo, H. Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones. In Proceedings of the 2012 Proceedings IEEE INFOCOM, Orlando, FL, USA, 25–30 March 2012; pp.2716–2720.
28. Chen, M.; Miao, Y.; Hao, Y.; Huang, K. Narrow band Internet of Things. *IEEE Access* **2017**, *5*, 20557–20577. [CrossRef]
29. Nguyen, H.; Shen, Z.; Gu, X.; Subbiah, S.; Wilkes, J. AGILE: Elastic distributed resource scaling for infrastructure-as-a-service. In Proceedings of the 10th International Conference on Autonomic Computing (ICAC), San Jose, CA, USA, 26–28 June 2013; pp. 69–82.
30. Granero, M.S.; Segovia, J.T.; Prez, J.G. Some comments on Hurst exponent and the long memory processes on capital markets. *Phys. A Stat. Mech. Appl.* **2008**, *387*, 5543–5551. [CrossRef]
31. Hosking, J.R. Fractional differencing. *Biometrika* **1981**, *68*, 165–176. [CrossRef]