sensors

Article

Mobility-Aware Service Caching in Mobile Edge Computing for Internet of Things

Hua Wei, Hong Luo * and Yan Sun

Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing 100876, China; weihua2015@bupt.edu.cn (H.W.); sunyan@bupt.edu.cn (Y.S.)
* Correspondence: luoh@bupt.edu.cn

Received: 30 December 2019; Accepted: 20 January 2020; Published: 22 January 2020

Abstract: The mobile edge computing architecture successfully solves the problem of high latency in cloud computing. However, current research focuses on computation offloading and lacks research on service caching issues. To solve the service caching problem, especially for scenarios with high mobility in the Sensor Networks environment, we study the mobility-aware service caching mechanism. Our goal is to maximize the number of users who are served by the local edge-cloud, and we need to make predictions about the user's target location to avoid invalid service requests. First, we propose an idealized geometric model to predict the target area of a user's movement. Since it is difficult to obtain all the data needed by the model in practical applications, we use frequent patterns to mine local moving track information. Then, by using the results of the trajectory data mining and the proposed geometric model, we make predictions about the user's target location. Based on the prediction result and existing service cache, the service request is forwarded to the appropriate base station through the service allocation algorithm. Finally, to be able to train and predict the most popular services online, we propose a service cache selection algorithm based on back-propagation (BP) neural network. The simulation experiments show that our service cache algorithm reduces the service response time by about 13.21% on average compared to other algorithms, and increases the local service proportion by about 15.19% on average compared to the algorithm without mobility prediction.

Keywords: sensor networks; internet of things; mobility sensor; mobile edge computing; service cache; mobility-aware; service response time

1. Introduction

With the rapid development of large-scale cloud computing, more and more service providers have chosen to deploy content and services on the cloud. However, the traditional cloud computing environment is not suitable for Internet of Things (IoT) services [1], especially for delay-sensitive services that involve sensors [2], such as cognitive assistance, image recognition, mobile games and augmented/virtual reality (AR/VR) [3]. Generally speaking, such services are not only highly delay-sensitive, but also have complex computing requirements. For this problem, the researchers propose a mobile edge computing model, i.e., deploying an edge-cloud server on the base station which is close to the end user to perform some computing jobs. It can significantly reduce network transmission delay [4].

Based on the mobile edge computing (MEC) architecture, there are many related research [5], including computation offloading, energy efficiency, security mechanisms [6], etc. [7,8]. Now, the current research is mainly focus on computation offloading. The computation offloading means that the mobile device offloads its computing jobs to the edge-cloud network. It solves the problems of resource storage, computational performance and efficiency. However, service caching, another topic which is as important as computation offloading receives less attention.

In contrast to computation offloading, service caching refers to caching the corresponding service and its associated database/library in an edge-cloud server (for example, a MEC-enabled base station (BS)) so that it can perform the corresponding computing job [3]. It needs to download associated database/library/program from the remote cloud in advance. Service caching is a relatively long-term decision compared to computation offloading [9]. At the same time, since the edge-cloud server deployed on the BS is smaller than the remote cloud server, the edge-cloud resources and computing power are limited compared with the remote cloud center, so that any edge-cloud server cannot support a wide range of applications. Therefore, to make the local edge-cloud serve more users, we must decide which services should be cached.

To guarantee the quality of service, service providers (SP) often establish service level agreements (SLA) with customers to ensure quality of service. Among them, service response time, which defines the time from request generation to service result return, is the most important indicator in SLA. Therefore, an important problem in cloud computing is how to complete service deployment between multiple data centers to reduce the service response time. However, for delay-sensitive sensor services, the transmission time of the link is difficult to guarantee due to the long distance between the cloud centers. MEC architecture effectively solves this problem by deploying edge-cloud servers on base stations (BSs) close to the end user. To enable more users to get lower service response time, we should choose popular services for caching. Therefore, how to effectively determine popular services is one of the key problems in this paper.

At present, a small amount of research on service caching has focused on: joint optimization of computing offloading and how to coordinate service caching in dense cellular networks. There is a lack of consideration for user or sensor mobility. However, with the development of the 4/5G network, the coverage of a single BS is getting smaller and smaller. Due to the influence of terrain, the coverage radius of the BS ranges from 500 m to 1 km. In such a dense range, frequent switching of the BS will result in frequent switching of available edge-cloud servers, thereby affecting service performance. Therefore, we must consider the user mobility. In this paper, we propose a mobile-aware service caching mechanism in mobile edge computing.

The traditional edge computing research idealizes the coverage area of each BS as a regular hexagon, while assuming that the user does not move between areas, as shown in Figure 1. However, when we consider user mobility, the overlap of coverage areas must be taken into account. Therefore, we consider the edge computing environment shown in Figure 2, where each circle represents the coverage of the BS, the black dots indicate users, the direction of the user arrow indicates its direction of movement, and the length indicates its speed of movement. First, we make predictions about the user's target location. When the target location is in the overlapping area, the service request will be forwarded preferentially to the BS which has cached this service. Otherwise, we forward the service request to the edge-cloud server corresponding to the target location as much as possible, thus reducing unnecessary stateful migration [10]. Effective mobile prediction and service allocation can avoid invalid service requests. Then, we propose a service cache selection algorithm based on BP neural network. It can train and predict the most popular services online. The main contributions of this paper are summarized below.

- We propose a new edge-cloud computing model, considering the BS coverage overlap and user mobility, to ensure that user requests are executed as much as possible in local edge server.
- We design a user location prediction model. It uses the existing information to determine the BS where the user is located when the service is completed. Since it is difficult to obtain all the

information needed by the model in practical applications, we use frequent patterns to mine local moving track information.

- We propose a service cache selection algorithm based on BP neural network for the edge-cloud server. It uses the historical information and existing service requests to predict the most popular services online.

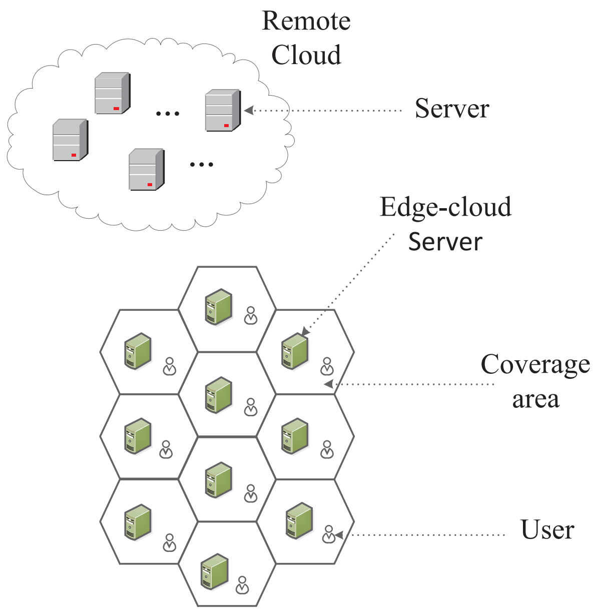

Figure 1. Traditional Edge-Cloud Computing Coverage Model.

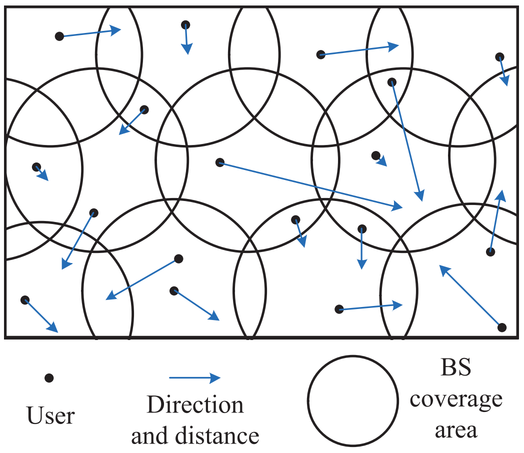

Figure 2. Mobility-Aware Edge Computing Environment.

The rest of the paper is organized as follows. Section 2 deals with related work. We give the system model in Section 3. In Section 4, we describe the user location prediction and service allocation, and Section 5 describes the service cache selection algorithm in edge-cloud based on back-propagation algorithm. The experiment and performance evaluation is show in Section 6. Finally, conclusions and future work are given in Section 7.

2. Related Work

Cloud computing platform and gradually developed into the Internet of Things (IoT) application processing efficient platform [11]. However, since the limitations of cloud models and the emergence of highly sensitive IoT applications (e.g., cognitive assistance, image recognition, mobile games and AR/VR), high transmission delays in cloud computing have hampered the development of such IoT applications.

To solve the above problem, the researcher proposed an mobile edge computing (MEC) paradigm [12], the mobile edge device (server) is set at the BS, and the computing job of the mobile device can be offloaded on the edge device. The researchers also proposed many similar concepts, such as cloudlet [13], follow-me cloud [14], fog computing [15], small cell cloud [16] and mobile micro-cloud [17]. Although these different concepts produce slightly different implementations, they all recommend placing a small cloud infrastructure at the edge of the network so that users can seamlessly connect to cloud services. In particular, MECs are typically placed only on one or several network hops from mobile users, thus effectively reducing communication delays [12].

Based on MEC model, researchers have done much research work. The existing research mainly focuses on computation offloading [18,19], Yang et al. [20] proposed to use the user's mobile mode and service access mode to predict the distribution of user requests in order to minimize the problem of user access delay and service provider cost; and then adjust the service layout and load schedule on the network according to the prediction result. Tan et al. [21] propose a generic model for minimizing job response time in the edge-cloud, where jobs are generated in any order and time on the mobile device. And the first online job scheduling algorithm in the edge-cloud is proposed, called OnDisc algorithm. Wang et al. [17] studied how to offload computing job to deal with the dynamic changes of these networks in the dynamic micro-cloud. The goal is to find the best layout for the instance to minimize the average cost. The paper proposes an offline algorithm that solves the optimal computation offloading problem in a specific look-ahead time window. Wu et al. [22] developed an analytical queueing model for delayed offloading systems with intermittent connectivity. This method effectively improves the success rate of computing offload in a mobile environment. Wu et al. [23] also proposed an energy-efficient offloading-decision algorithm based on Lyapunov optimization. This algorithm effectively reduces the average energy consumption on mobile devices. Although those solutions for computation offloading can effectively solve the problem that mobile device has limited performance. However, the above research does not give a reasonable solution to the problem that what kind of service (database/lib library) should be cached by the edge-cloud to satisfy the near-end service request. Xu et al. [3] studied the dynamic service caching problem in dense cellular networks with MEC enabled. He proposed an OREO algorithm that can jointly optimize dynamic service caching and computational offloading to address many key challenges in MEC systems, including service heterogeneity, unknown system dynamics, spatial demand coupling and decentralized coordination. Chen et al. [9] studied collaborative service caching in MEC-enabled dense small cell (SC) networks. He proposed a collaborative service caching algorithm to solve the service caching problem in a centralized way. However, these existing service cache studies do not consider user mobility. Since users are mobile, the connected edge-cloud will change, and stateful migration will reduce the system performance. Therefore, we need to study a mobility-aware service caching mechanism.

3. System Model

We consider the system model that consists of a series of mobile users or mobile sensors, BSs with overlapping coverage (each BS deploys edge-cloud servers) and a remote cloud center. User mobility and BS deployment are shown in Figure 2. To support seamless service transport between MECs, we use the three-tier container architecture described in [10]. The edge-cloud server will co-exist and work with our existing centralized remote cloud. Therefore, we only focus on user mobility prediction and service caching decision, but not on service request forwarding, information interaction and service execution. To simplify the model, we believe that each user can seamlessly connect to the required BS in the process of move.

The system architecture is shown in Figure 3. Users who can move arbitrarily generate service requests with random probability. The request is first sent to the connected BS, and the edge-cloud server at the location performs mobility prediction. If the predicted user will remain in the range of the BS, subsequent computing jobs are performed locally; otherwise, we will forward the service request to the edge-cloud server corresponding to the target BS according to the service allocation algorithm. In particular, when the target area is completely in the overlapping area, whether to forward the service request is determined by the current service cache of the surrounding BS. After receiving the service request, the edge-cloud server chooses to perform the service locally or forward it to the remote cloud computing center according to whether the service is currently cached. At the same time, the system records and counts the number of all kinds service requests received locally, and predicts the number of service requests in the future based on the established BP neural network model. The neighboring server is notified by broadcast when the edge-cloud server determines the service cache. Our goal is to enable edge-cloud to serve more users.

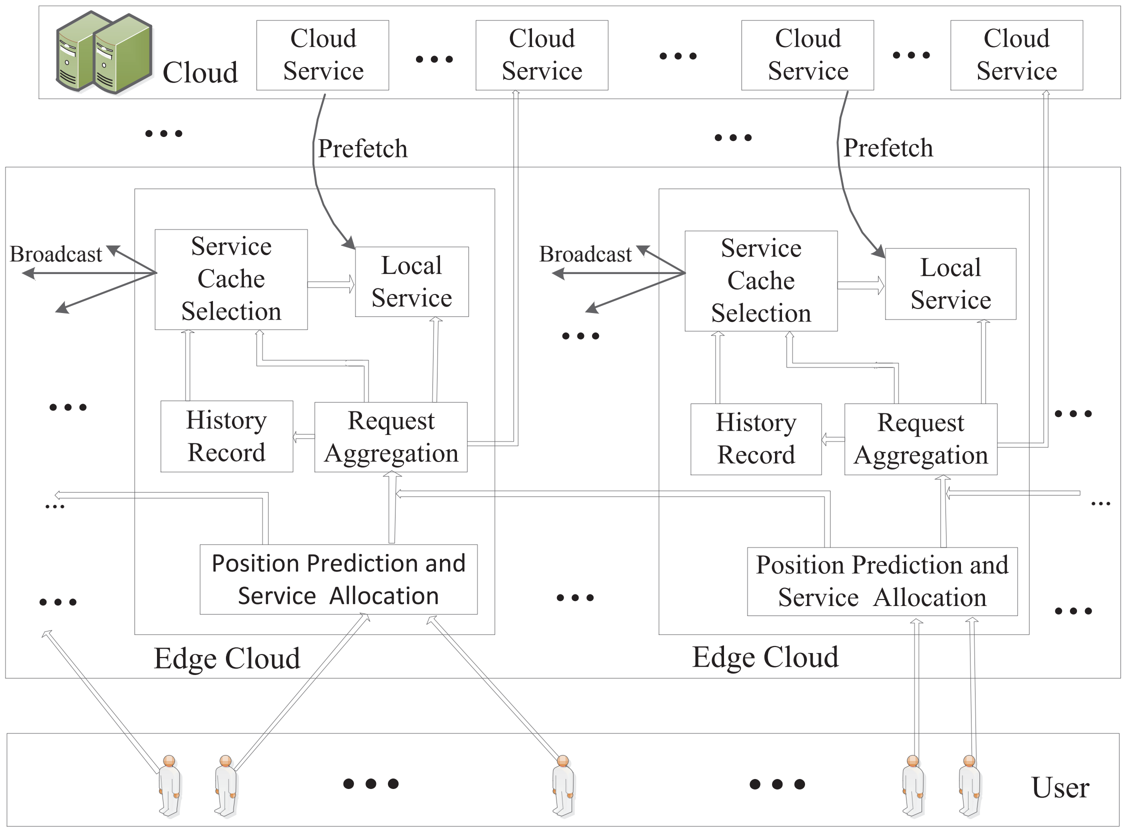

Figure 3. The System Architecture.

The entire mobile network contains U users, indexed by U, each user $u \in U$; N BSs, indexed by N, each BS $n \in N$; and a remote cloud computing center. The system contains K services, indexed by K, each service $k \in K$. If the user u sends a service request to the edge-cloud deployed by BS n, we denote $s(u,n,k) = 1$, otherwise $s(u,n,k) = 0$. Obviously, at any time, the user requests require satisfaction of Equation (1).

$$\sum_{n=1}^{N}\sum_{k=1}^{K} s(u,n,k) \leq U \tag{1}$$

The computing power of a remote cloud computing center can be seen as infinite and can perform a variety of computing services. The resource required for the cache computing service k is c_k, and the capability of each BS to provide a cache computing service is denoted as C_n.

$$x_{kn} = \begin{cases} 0, & \text{computing service } k \text{ is not cached at } n \\ 1, & \text{computing service } k \text{ is cached at } n \end{cases} \tag{2}$$

Obviously, for computing services that can be cached at BS n:

$$\sum_{k=1}^{K} x_{kn} c_k \leq C_n \tag{3}$$

We use u_n^k indicates the service response time that can be provided to the user u when the service k is cached by the BS n, and u_0^k indicates the service response time that can be provided to the user u when the service is provided by the remote cloud. The service response time is defined as the time from the request generation to the result return.

$$u_n^k \leq u_0^k \tag{4}$$

Since the local edge-cloud service response time is less than remote cloud significantly, our goal is maximizing the number of users which served by the local edge-cloud. Our goal is to find the most suitable $[x_{kn}]$, the objective function of our service caching can be formally expressed as:

$$\max_{k\in K, n\in N} \quad \sum_{u\in U} s(u,n,k) x_{kn} \tag{5}$$
$$\text{subject to} \quad (1)\text{–}(4).$$

4. User Location Prediction and Service Allocation

For the user mobility problem, first, we propose an idealized geometric model to predict the target area of the user's movement. Then, considering that it is difficult to obtain all the data needed for the model in practical applications, we use frequent patterns to mine local moving track information. Finally, we propose a service allocation algorithm that forwards service requests to the target area as much as possible while considering the existing service cache.

4.1. Idealized Geometric Model

To predict the target area of the user's movement, we need to obtain the user's current location information and motion status. We represent the obtained information as the following 6-tuple:(Position, Position error, Speed, Speed error, Direction, Direction error). Among them, the Position is obtained by the positioning system, the Position error is generated according to the influence of the positioning system and the environment; the Speed refers to the average speed from the current position to the service completion position, and the Speed error is determined by the motion mode and the speed accuracy of the positioning system; similarly, the Direction indicates the angle from the current position to the position at which the service is completed. Since the service we are considering is completed in a relatively short time, we believe that the Direction is a linear direction and the Direction error is the possible direction deviation.

It should be noted that this paper focuses on the service cache using the mobility prediction results. We do not consider how to obtain user mobility information, the protection of personal location privacy and how to complete information estimation when relevant information is not available. Relevant researchers have provided mature solutions to the above problems, and interested researchers can refer to [24–28].

As shown in Figure 4, the blue area is the possible position after the user moves, radius is the size of the position precision, represented as *rad*; L_1 represents the shortest moving distance (estimated service response time $*$ minimum moving speed), and L_2 represents the maximum moving distance (estimated service response time $*$ maximum moving speed). β is the angle of direction error twice. The position accuracy and speed accuracy provided by the existing positioning system (including the one under construction) are given in Table 1. In this paper, we use GPS performance parameters, and all the above performance parameters are civil performance parameters [29].

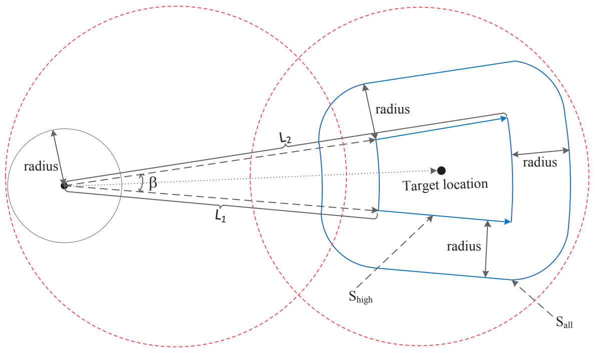

Figure 4. User Location and Target area.

Table 1. Positioning system performance comparison.

Name	Position Accuracy	Timing Accuracy	Speed Accuracy
GPS	2–10 m	20 ns	0.1 m/s
Galileo	1–5 m	20 ns	0.1 m/s
Glonas	10–20 m	25 ns	0.1 m/s
Beidou	5–15 m	50 ns	0.2 m/s

Therefore, the area of the target area that the user may move to can be approximated as follows:

$$S_{all} \approx \pi * rad^2 + \frac{\beta * \pi}{360}[(L_2 + rad)^2 - (L_1 - rad)^2] + 2 * rad * (L_2 - L_1) \tag{6}$$

Among them, the middle small fan shape is a region with a higher probability of the user's location, and the area can be expressed as:

$$S_{high} = \frac{\beta * \pi}{360}(L_2^2 - L_1^2) \tag{7}$$

The process by which we calculate the target location and target area is as follows.

step1: Calculate the target location using position, direction, speed, and estimated service response time.

step2: Calculate S_{all} by Equation (6).

step3: Calculate S_{high} by Equation (7).

step4: Find all the BSs near the target area through the target location and S_{all}, denoted as $B = \{n_1, n_2, \ldots, n_{all}\}$.

4.2. Mining Frequent Patterns

To use the geometric model proposed in the previous section to predict the target location, we need to collect the user's movement track information, i.e., the 6-tuple information. However, in practical applications, it is often difficult to obtain all the information due to the unreliability of the mobile device and the limitations of the positioning system. Therefore, we use the data mining frequent pattern to analyze the trajectory data in the same range to obtain the common behavior pattern at the group level. In this paper, we study the potential association rules of user movement trajectory information, so that the trajectory information can be complemented for the model to perform location prediction.

Our basic assumption is that people often follow the crowd: individuals tend to follow a common path. For example, people go to work every day through similar routes, pedestrians walk at similar speeds and public transportation goes through similar routes in different time periods [30]. Therefore, if we have enough data to simulate typical behavior, we can use this knowledge to predict the missing user movement information for most people. Existing research has proved that this assumption is reasonable [30]. In this paper, we use the movement trajectory of all users/objects in the base station area to learn motion information.

For position predictions where all mobile information has been acquired, we directly use the geometric model presented in the previous section. If the information obtained is incomplete, we build T-pattern Tree [31]. The T-pattern Tree constructed is shown in Figure 5. The Root is a virtual node. And the number in the node indicates the number of supports. We use the existing mobile information as a transaction data set. The element items in the transaction include the movement track information. First, we traverse the data set, calculate the frequency of each individual element, and filter out the unsatisfied elements based on the minimum support. Then, the data set is processed. Each piece of data is sorted by the absolute frequency of occurrence of the elements and filters out elements that do not satisfy the minimum support. After processing, iterate through the data set, insert each piece of data into the T-pattern Tree, and recursively add the path from the root node. If it exists, the value will be increased. If it does not exist, a new node will be created. We can find frequent item sets by looking up the condition base of the element item. Repeat the above process until the T-pattern Tree contains only one element [32]. This element is substituted into the model as a missing element to predict the user's location.

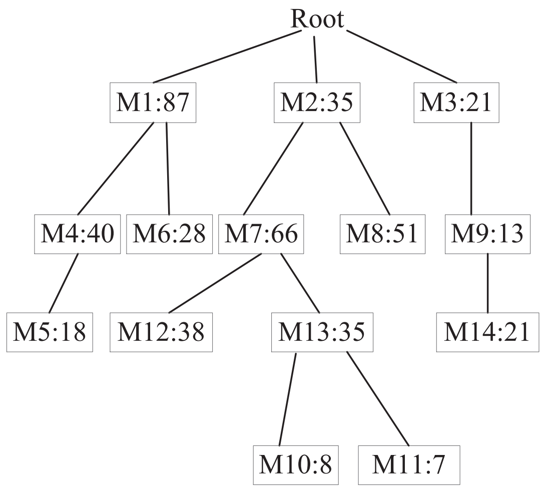

Figure 5. T-pattern Tree.

4.3. Service Allocation Algorithm

Due to the uncertainty of the location and direction of movement when the user requests the service, the area where the user may eventually appear has multiple cases. Figure 6 shows some of the possible cases. How to forward the service requests in different cases is a problem that we need to solve.

To send the service request to the most suitable edge-cloud server, we design a service allocation algorithm based on existing service cache. When the user is in the overlapping area, the service request is preferentially forwarded to the BS which has cached this service. Otherwise, by calculating the proportion of target area within the coverage of each base station, the BS with high probability is selected to provide services in priority. The service allocation algorithm is shown in Algorithm 1. Among them, *Max* represents the maximum iteration number.

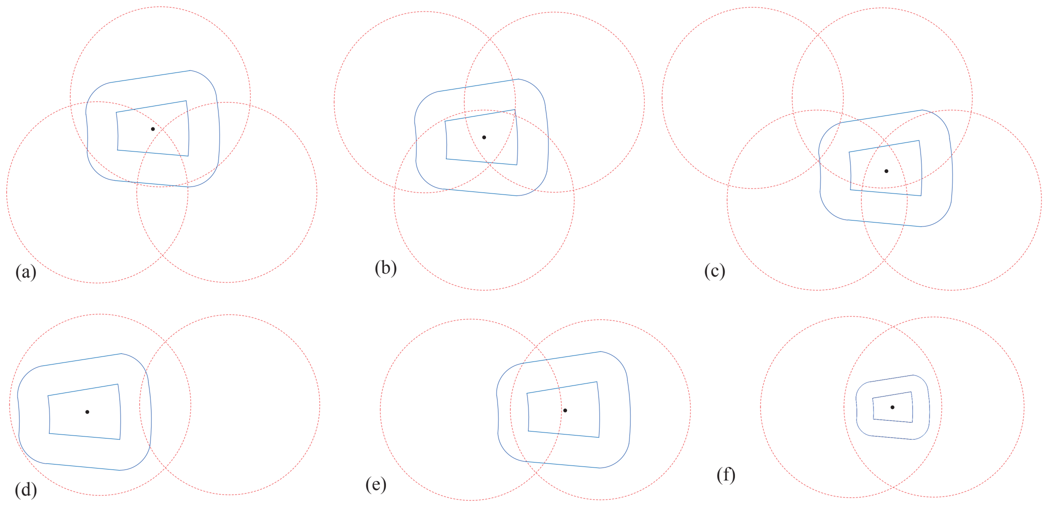

Figure 6. Target area and BS coverage area.

Algorithm 1 Service Allocation Algorithm

```
1:  Initialize $Max = 0$.
2:  if $S_{high}$ is all in the overlap area then
3:      for $n_i = n_1$ to $n_{all}$ do
4:          if $n_i$ has cached this service then
5:              target BS = $n_i$;
6:              Send request to $n_i$;
7:              break;
8:          end if
9:      end for
10: else
11:     for $n_i = n_1$ to $n_b$ do
12:         $A(n_i)$ = the size of $S_{all}$ in $n_i$
13:         if $Max < A(n_i)$ then
14:             $Max = A(n_i)$;
15:             target BS = $n_i$;
16:         end if
17:     end for
18:     Send request to target BS.
19: end if
```

We give priority to the distribution of the S_{high} area. When S_{high} is all in the overlap area, we prefer to forward the request to the nearby base station which has cached this service. We denote the size of S_{all} in base station n_i as $A(n_i)$. When S_{high} is not all in the overlap area, we select the BS which has largest $A(n_i)$ as the target BS.

5. Service Cache Selection Algorithm

Edge-cloud servers can aggregate to more accurate local service requests through mobility prediction. And it also avoids invalid service requests. Using the aggregated service requests to accurately predict the amount of service requests is the key to determine the service cache. Since the trends of service requests from different locations are different, we need to make predictions on service requests online. Considering that the edge server has limited computing resources, we must use a simple and efficient model for prediction.

BP neural network effectively meets the above requirements. In this paper, first, we use historical data to count the top k service types of local edge server service requests. After that, the BP neural network model is used to predict the number of the above k service requests in the future. Finally, the service cache is implemented based on the predicted results.

5.1. BP Neural Network Model

The BP network implements the mapping function from input to output, which can prove that the three-layer neural network can approximate any nonlinear continuous function with arbitrary precision, which makes it especially suitable for solving complex internal mechanisms [33]. At the same time, since it can automatically extract the "reasonable rules" between input and output data through learning, and adaptively store the learning content in the weight of the network. Therefore, we can use it to predict the number of requests for a certain service type in the future [34].

In contrast to data caching, service caching usually needs to consider the long-term behavior characteristics of the server. To ensure the stability of the edge server, we cannot change the cached service frequently. Therefore, we start each hour with a prediction of the service for the next hour and cache the service based on the prediction. We use 24 h as a cycle, denoted by h. According to [35], we can obtain prior knowledge that the request frequency of different services is related to whether it is a working day. Therefore, we consider the pattern of service requests at different periods in a week, using $d = 1, 2..., 7$ to represent Monday to Sunday.

As shown in Figure 7, the BP network that we constructed consists of an input layer, a hidden layer and an output layer. It is a $10 \times m \times 1$ three-layer BP network model. When constructing a BP neural network model, it is important to determine the number m of hidden neurons. Excessive number of hidden layer neurons will increase the amount of network calculation and easily cause over-fitting problems; if the number of neurons is too small, it will affect network performance and will not achieve the expected results. The number of hidden layer neurons in the network is directly related to the complexity of the actual problem, the number of neurons in the input and output layers, and the setting of the expected error. In this paper, we use the following empirical formula to select the number of hidden layer neurons:

$$m = \sqrt{n + l} + \lambda \tag{8}$$

where λ is a constant between 1–10, n is the number of input layer nodes, and l is the number of output layer nodes.

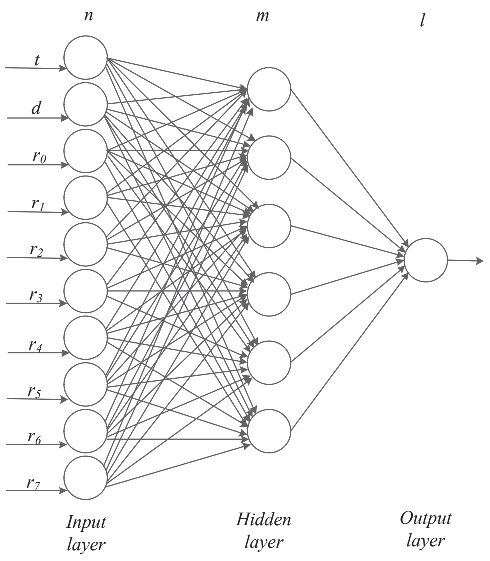

Figure 7. BP Neural Network Model.

Therefore, we choose $m = 6$ as the number of nodes in the hidden layer. We have 10 neurons in the input layer and one neuron in the output layer, and the output is the number of service requests in the next period. W^l represents a matrix, b is the offset vector, and L is the L-th layer. A^l represents the output value of the l-th layer. We use the mean square deviation to construct the loss function, hoping to minimize Equation (9).

$$J(W, b, In, Out) = \frac{1}{2}\|a^L - Out\|_2^2 \tag{9}$$

where In represents the input 10-dimensional vector, a^L represents the calculation result of the L-th layer of the output layer, and Out represents the desired output result. $\|S\|$ indicates the $L2$ norm of S. The network model uses the Sigmoid activation function, as shown in Equation (10):

$$f(x) = \frac{1}{1 + e^{-x}} \tag{10}$$

We use the gradient descent method to iteratively solve W, b for each layer. Since $a^L = \sigma(z^L) = \sigma(W^L a^{L-1} + b^L)$, we can get:

$$\frac{\partial J(W, b, In, Out)}{\partial W^L} = \frac{\partial J(W, b, In, Out)}{\partial Z^L}\frac{\partial Z^L}{\partial W^L} = (a^L - Out) * \sigma'(Z^L)(a^{L-1})^T \tag{11}$$

$$\frac{\partial J(W, b, In, Out)}{\partial b^L} = \frac{\partial J(W, b, In, Out)}{\partial Z^L}\frac{\partial Z^L}{\partial b^L} = (a^L - Out) * \sigma'(Z^L) \tag{12}$$

$$\delta^L = \frac{\partial J(W, b, In, Out)}{\partial Z^L} = (a^L - Out) * \sigma'(Z^L) \tag{13}$$

After that, we can recurse the gradient of the front layer. We use the following formula in forward propagation:

$$z^l = W^l a^{l-1} + b^l \tag{14}$$

Therefore, the gradients of W^L and b^L of the L-th layer are as follows:

$$\frac{\partial J(W,b,In,Out)}{\partial W^L} = \frac{\partial J(W,b,In,Out)}{\partial Z^L}\frac{\partial Z^L}{\partial W^L} = \delta^l(a^{l-1})^T \tag{15}$$

$$\frac{\partial J(W,b,In,Out)}{\partial b^L} = \frac{\partial J(W,b,In,Out)}{\partial Z^L}\frac{\partial Z^L}{\partial b^L} = \delta^l \tag{16}$$

Obviously, the recursive relation of δ^l can be obtained by inductive method:

$$\delta^l = \delta^{l+1}\frac{\partial z^{l+1}}{\partial z^l} = (W^{l+1})^T\delta^{l+1} * \sigma'(z^l) \tag{17}$$

The training process of the predictive model based on the back-propagation algorithm is shown in Algorithm 2. We use the batch gradient descent method. The input 10-dimensional vectors represent clock h, day of the week d, last time period request quantity r_0, historical synchronous request quantity $r_1, r_2, r_3, r_4, r_5, r_6, r_7$.

Algorithm 2 BP Neural Network Training Model

```
1:  Initialize coefficient matrix W and the offset vector b as a random value;
2:  for iteration number = 1 to MAX do
3:    for $i = 1$ to $Y$ do
4:      Set input $a^1$ as $In_i$
5:      for $l = 2, 3$ do
6:        Calculation $a^{i,l} = \sigma(z^{i,l}) = \sigma(W^l a^{(l-1)} + b^l)$;
7:      end for
8:      Calculate the output layer's $\delta^{i,3}$ through the loss function;
9:      Calculate $\delta^{i,2} = (W^3)^T\delta^{i,3} * \sigma'(z^{i,2})$
10:     Update $W^2$ and $b^2$:
11:       $W^2 = W^2 - \alpha\sum_{i=1}^{Y}\delta^{i,2}(a^{i,1})^T$
12:       $b^2 = b^2 - \alpha\sum_{i=1}^{Y}\delta^{i,2}$
13:     if The change value of $W, b \le \epsilon$ then
14:       break;
15:     end if
16:   end for
17: end for
```

5.2. Service Cache Selection Algorithm

Based on the above training results, the coefficient matrix W and offset vector b can be obtained. Therefore, we can predict the number of future service requests and design the corresponding service caching algorithm.

We use the historical data to count the top k service types of local service requests, denoted as $top[k]$. Then, using Algorithm 2, at the beginning of each time period, predict the requests for the top k services and sort the requests. Finally, the first i services are cached according to the sorting result, where i represents the maximum number of cacheable services that the current base station can provide.

$$\sum_{i=1}^{i} c_i \le C_n \tag{18}$$

In this way, the number of users supported by the locally cached service can be maximized. The pseudo code of the service cache selection algorithm which we design is shown in Algorithm 3.

Algorithm 3 Service Cache Selection Algorithm

```
Require: $W, b, k$, local historical data.
1: Select the top $k$ services with the largest number of local service requests, denoted as $top[k]$;
2: while each time period begins do
3:    Initialize $c_i = 0$;
4:    for $j = 0; j < k; j++$ do
5:       Get $Out_j$ with trained BP neural network;
6:    end for
7:    Sort the resulting $Out_j$ in descending order, denoted as $top[j]$;
8:    for $j = 0; j < k; j++$ do
9:       if $c_i \leq C_n$ then
10:         Cache service $top[i]$;
11:         $c_i = c_i + c_j$;
12:      else
13:         break;
14:      end if
15:   end for
16: end while
```

5.3. Model Performance Analysis

To choose a suitable prediction model, we have compared the performance of the 3-layer BP network model, the deep neural network (DNN) model, and the recurrent neural network (RNN) model. The model performance comparison is shown in Table 2. Among them, MAE represents the mean absolute error, $RMSE$ represents the root mean squared error.

Table 2. Model Performance Comparison.

Kind	Group	Iterations	MAE	RMSE
3-layer BP	1	2000	26.51	29.17
	2	895	15.17	17.56
	3	1118	26.82	29.25
	4	1690	7.53	8.86
	5	2000	13.86	15.43
	average	1540.6	18.178	20.054
DNN	1	2000	18.92	24.35
	2	1083	26.57	34.58
	3	1601	32.1	43.57
	4	2000	8.98	11.13
	5	1920	6.52	8.52
	average	1720.8	18.618	24.43
RNN	1	1575	13.74	19.04
	2	1119	18.98	25.36
	3	2000	22.45	29.81
	4	2000	14.83	18.18
	5	2000	21.17	23.56
	average	1738.4	18.234	23.19

It can be seen that the MAE of the three models is almost the same. However, the $RMSE$ of DNN and RNN models is higher than the $RMSE$ of 3-layer BP model. In addition, it also can be seen from Table 2, the iteration number of RNN and DNN is significantly higher than the iteration number of 3-layer BP. Therefore, the proposed 3-layer BP model is much more suitable for service cache prediction.

6. Experiment and Performance Evaluation

To verify the effectiveness of the proposed algorithm, we take a series of experiments. It is hard to acquire real service datasets due to the users' data confidentiality issues and policies maintained by service providers (SP) [36]. Therefore, we adopt the Baidu API to obtain the points of interest in each area, and then determine the type of service request in the base station. The service request obeys Poisson distribution. We used the GPS trajectory dataset from Microsoft Research Asia [37–39] and user trajectories extracted from Call Detail Records (CDR) which published by Orange [40] and the University of Minnesota [41] to construct the dataset required for the experiment.

6.1. Experimental Environment

We conduct systematic simulations in practical scenarios, and our system model adopts the widely used stochastic geometry approach. We choose a 15 km $\times$ 7 km rectangular area in the north of Beijing as the simulation experiment area. We assume that the BS coverage radius is 500 m and the distance between adjacent BSs is 750 m. The user's starting position is randomly generated, and the user's moving distance and time are generated from previously constructed dataset.

The dataset is represented by a sequence of time-stamped points, each of which contains the information of latitude, longitude, high, speed and heading direction, etc. To construct the experimental data set, we use the spherical body distance formula to calculate the distance between adjacent points. Here, we take 6,371,000 m as the radius of the earth. We refer to the trajectory of adjacent points in the data set as the moving path, and randomly select 200,000 moving paths. The moving distance distribution is shown in Figure 8. The y-axis represents the moving distance and the x-axis represents the track number. It can be seen that the length of most trajectories does not exceed 1 km.

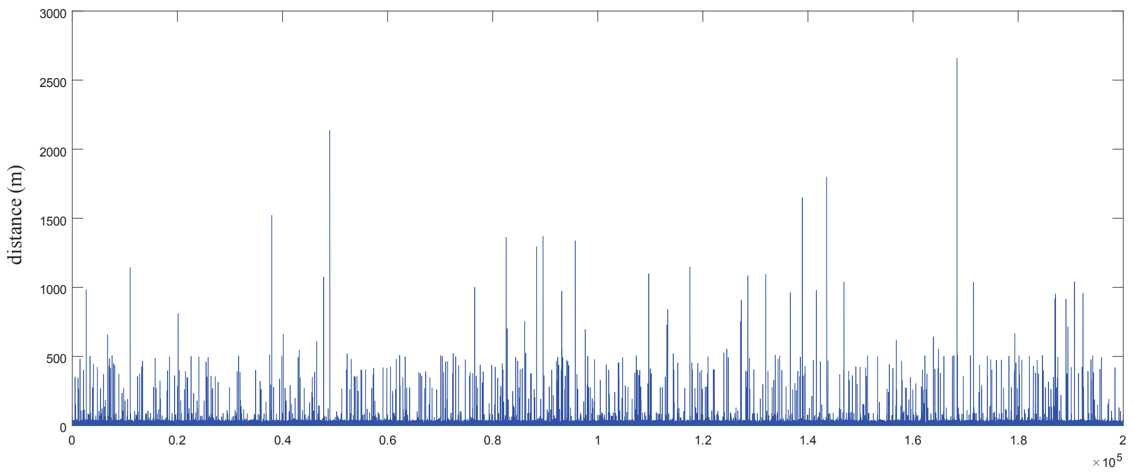

Figure 8. Moving Distance Distribution.

To facilitate the calculation, we use the user's movement time between two points as the service execution time. The distribution of service execution time is shown in Figure 9. Among them, 81.88% of the movement time is less than 5 s.

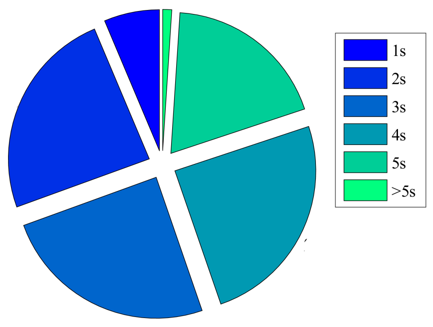

Figure 9. Service Execution Time.

As described in [42], the main traffic type under a BS is strongly correlated with the location of the BS and points of interest (POIs) around. Inspired by this, we use POIs around BSs to determine the main type of service request within the BS. Service type is Zipf-like distribution. The frequency F of any service is inversely proportional to its rank in the frequency table r. It can be expressed as $F = C/r^{z_p}$, where C denotes the normalization factor, z_p is a parameter. Since the different z_p values imply the different content distribution, we take experiment to evaluate the algorithm performance under different z_p values [43]. In our experiments, z_p is set to 0.5, 0.7, and 0.9, respectively.

The delay from terminal device to the edge server is set between 0.02 s and 0.05 s, and the delay from the terminal device to the remote cloud server is set between 0.4 s and 1 s. We assume that all the edge-cloud serves are the same, and there are a total of 100 different types of services. If not specified explicitly, we assume that the cache space in the edge-cloud server can cache 10 different services.

6.2. Performance Comparison

In the comparison experiment, we used three different algorithms to compare with our method (MASC). **Service caching strategy without mobility-aware (Non-MASC):** BSs cache services are determined by our service cache selection algorithm, but lack user mobility prediction. The service request is completed by the edge-cloud server at the starting location. **Myopic service caching (MSC) [3]:** This method has user mobility prediction. However, it does not consider long-term service request rules, and the cache services is only determined by the request amount of the previous period. **Collaborative service caching (CSC) [9]:** Collaborative service caching assume that SCs are obedient and fully cooperative. The service caching decisions of the SCs will reshape the workload distribution in the network to better use the limited computing resources of individual BSs. If the service is not executed locally, it needs to generate multi-hop data transmission. In this case, we assume that the data transmission delay between different base stations is 0.5 times the execution time.

We divide all the service request into separate data groups. In each experiment, we randomly select 20,000 moving tracks from 200,000 paths. We evaluate the performance of the service caching algorithm from the average service response time and local service proportion. Among them, the local service proportion refers to the proportion of service requests completed by the local edge-cloud server.

To verify the performance of our service cache selection algorithm, we compare MASC, MSC, and CSC algorithms. Figure 10 depicts the performance comparison of different algorithms under different service request distributions, where the x-axis denotes the number of users and the y-axis denotes the average service response time.

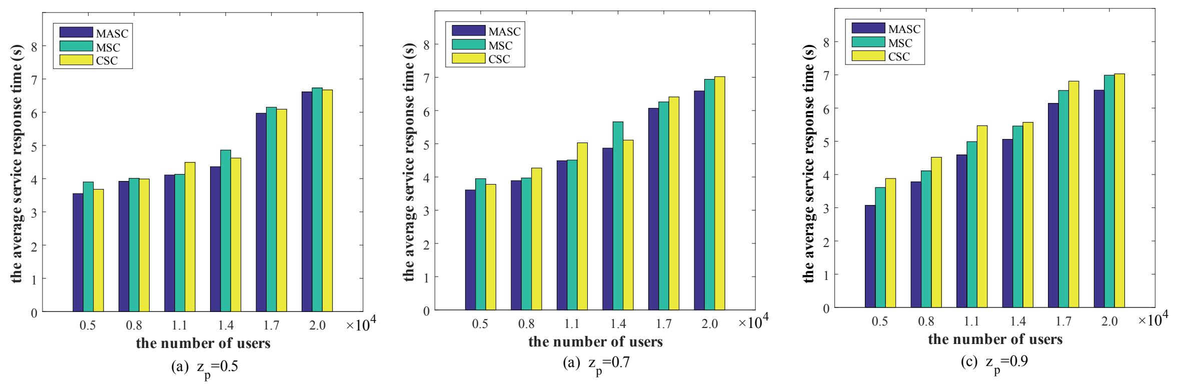

Figure 10. Performance comparison under different user numbers.

We observe that when the number of users increased from 5000 to 20,000, the average service response time has increased. Especially when the number of users exceeds 14,000, the average service response time increases significantly. This shows that after the number of users exceeding the edge server capacity, many service requests are forwarded to the remote cloud for execution. Moreover, it can be seen in Figure 10c, the average service response time when $z_p = 0.9$ is significantly less than the other two cases. This means that our algorithm performs better when the service request obeys long-tailed distributions.

By comparing the experimental results when $z_p = 0.9$, the proposed Mobility-Aware Service Caching Strategy can save the service response time by about 13.21% on average compared to the collaborative service caching. Compared to the Myopic Service Caching algorithm, it can save the service response time by about 8.46% on average, the biggest gap is 14.95%. Therefore, our algorithm is significantly better than other algorithms.

To verify the performance of our mobility prediction and service allocation algorithms, we compare MASC, Non-MASC, MSC, and CSC algorithms. Figure 11 depicts the performance comparison under different BSs, where the x-axis denotes the BS and the y-axis denotes the local service proportion. We selected 15 BSs with more service requests for comparison. In this experiment, z_p is set to 0.8 and the number of users is set to 12,000.

As shown in Figure 11, the MASC algorithm is significantly better than the MSC algorithm, with an average 45% higher than the MSC algorithm. Compared with the CSC algorithm, the local service proportion of the MASC algorithm is higher in most locations. Only in a few locations, such as BS 4,7,10,11, the local service proportion of our algorithm is slightly lower than the CSC algorithm. This is caused by the service load being forwarded to nearby cooperating edge servers. However, this significantly increases service response time (as shown in Figure 10). And the local service proportion of our proposed MASC algorithm is 15.19% higher on average than the CSC algorithm. Therefore, our algorithm is still better than the CSC algorithm.

We observe that our algorithm performs significantly better than the Non-MASC algorithm in most cases. And we also find that in a few locations, such as BS 10 and BS 11, the performance of our proposed algorithm is almost the same as the Non-MASC algorithm. Through analysis, we find that it is due to the complex road network within the range of the BS, which is difficult to effectively complete the prediction of the moving trajectory. However, the local service proportion of our proposed MASC algorithm is 23.6% higher on average than the Non-MASC algorithm, the biggest gap is 30.8%. Therefore, the performance of our algorithm is higher than the Non-MASC algorithm.

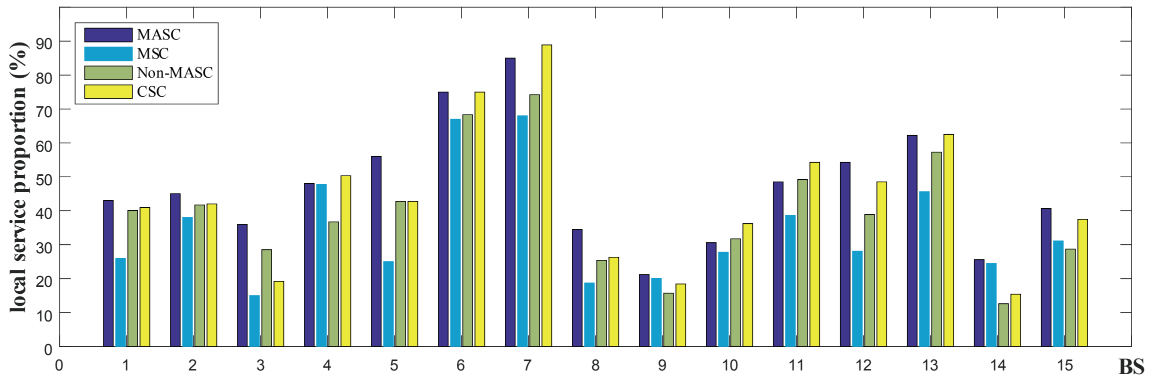

Figure 11. Performance comparison ($z_p = 0.8$).

To observe the impact of mobility prediction on service cache performance, we select different trajectory data for comparison experiments. First, we filter the trajectory with the moving distance greater than 250 m from the Microsoft Research Asia's GPS Trajectories dataset. Then, we add non-mobile users to the dataset in proportions of 10%, 20%, 30%, 40%, 50%, 60%, and 70%, respectively. Non-mobile users mean that the user is in a fixed location, it receives and generates information that is always in the same location. Finally, we compare the performance of MASC and Non-MASC algorithms under different non-mobile user proportions. In this experiment, z_p is set to 0.8.

Figure 12 depicts the performance comparison under different non-mobile user proportions, where the x-axis denotes the non-mobile user proportions and the y-axis denotes the average service response time.

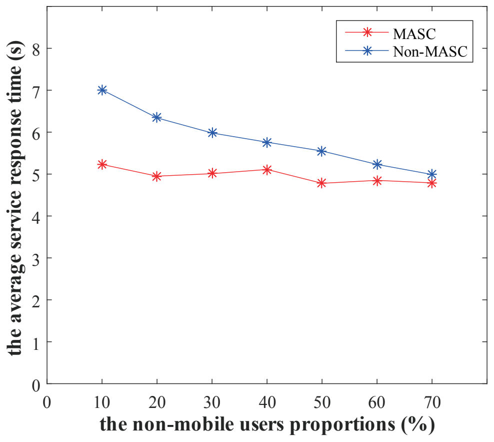

Figure 12. Performance comparison under different non-mobile user proportions ($z_p = 0.8$).

As shown in Figure 12, the average service response time of the MASC algorithm is stable at different proportions. And the service response time of Non-MASC algorithm is significantly larger than the MASC algorithm when the non-mobile user proportion is less than 70%. The Non-MASC algorithm lacks mobility prediction and service allocation, resulting in many service requests being executed in non-local areas or retransmitted. Therefore, our algorithm has good performance for environments with high user mobility.

7. Conclusions and Future Work

In this paper, we focus on the service caching strategy in IoT. The strategy takes user mobility into account. Based on the user location and mobility information, the base station forwards the service request to the target area as much as possible. In addition, we propose a service allocation algorithm to ensure that service is completed by the appropriate edge-cloud server, when the user is in the overlapping areas. Finally, we propose a service cache selection algorithm based on BP neural network, and choose the appropriate service for caching.

By comparing the experimental results, the proposed Mobility-Aware Service Caching Strategy can save the service response time by about 13.21% on average compared to the collaborative service caching. Compared to the Myopic Service Caching algorithm, it can save the service response time by about 8.46% on average. The local service proportion of our proposed MASC algorithm is 15.19% higher on average than the CSC algorithm and an average 45% higher than the MSC algorithm. The simulation results show that our method can effectively reduce the service response time and increase local service proportion. On the one hand, service allocation algorithm and user location prediction effectively avoid the situation of request failure. On the other hand, effective service predictions are made for the service cache, so the edge server can serve more users for a considerable period of time.

In the future, we will optimize the user location prediction, especially in the absence of GPS location information. Moreover, we will further look for a suitable service request dataset to train the neural network model with real data. In addition, we will explore the prediction algorithm for service cache sequence.

Author Contributions: H.W. and H.L. conducted the sensor and whole system modeling; H.W. and Y.S. contributed the BP algorithms; H.W. performed the experiments; H.W. and H.L. analyzed the experiment results; H.W. wrote the paper; writing–review and editing, H.L. and Y.S.; project administration, H.L. and Y.S. funding acquisition, H.L. and Y.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by National Natural Science Foundation of China under Grant 61772085, Grant 61532012, National Key R&D Program of China under Grant 2018YFB2100300, National Natural Science Foundation of China under Grant 61672109.

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

References

1. Li, S.; Da Xu, L.; Zhao, S. 5G Internet of Things: A survey. *J. Ind. Inf. Integr.* **2018**, *10*, 1–9. [CrossRef]
2. Wang, T.; Mei, Y.; Jia, W.; Zheng, X.; Wang, G.; Xie, M. Edge-based differential privacy computing for sensor–cloud systems. *J. Parallel Distrib. Comput.* **2020**, *136*, 75–85. [CrossRef]
3. Xu, J.; Chen, L.; Zhou, P. Joint service caching and task offloading for mobile edge computing in dense networks. In Proceedings of the 2018 IEEE Conference on Computer Communications (INFOCOM), Honolulu, HI, USA, 16–19 April 2018; pp. 207–215.
4. Patel, M.; Naughton, B.; Chan, C.; Sprecher, N.; Abeta, S.; Neal, A. Mobile-Edge Computing Introductory Technical White Paper. White Paper, Mobile-Edge Computing (MEC) Industry Initiative. 2014. Available online: https://www.scirp.org/reference/ReferencesPapers.aspx?ReferenceID=2453997 (accessed on 1 January 2020).
5. Abbas, N.; Zhang, Y.; Taherkordi, A.; Skeie, T. Mobile edge computing: A survey. *IEEE Internet Things J.* **2017**, *5*, 450–465. [CrossRef]
6. Wang, T.; Bhuiyan, M.Z.A.; Wang, G.; Qi, L.; Wu, J.; Hayajneh, T. Preserving Balance between Privacy and Data Integrity in Edge-Assisted Internet of Things. *IEEE Internet Things J.* **2019**. [CrossRef]
7. Wang, T.; Luo, H.; Zheng, X.; Xie, M. Crowdsourcing mechanism for trust evaluation in cpcs based on intelligent mobile edge computing. *ACM Trans. Intell. Syst. Technol.* **2019**, *10*, 62. [CrossRef]
8. Wu, Y.; Huang, H.; Wu, N.; Wang, Y.; Bhuiyan, M.Z.A.; Wang, T. An incentive-based protection and recovery strategy for secure big data in social networks. *Inf. Sci.* **2020**, *508*, 79–91. [CrossRef]
9. Chen, L.; Xu, J. Collaborative service caching for edge computing in dense small cell networks. *arXiv*, **2017**, arXiv:1709.08662.
10. Machen, A.; Wang, S.; Leung, K.K.; Ko, B.J.; Salonidis, T. Live service migration in mobile edge clouds. *IEEE Wirel Commun.* **2017**, *25*, 140–147. [CrossRef]

11. Zhang, Q.; Zhu, Q.; Zhani, M.F.; Boutaba, R.; Hellerstein, J.L. Dynamic service placement in geographically distributed clouds. *IEEE J. Sel. Areas Commun.* **2013**, *31*, 762–772. [CrossRef]
12. Nan, Y.; Li, W.; Bao, W.; Delicato, F.C.; Pires, P.F.; Dou, Y.; Zomaya, A.Y. Adaptive energy-aware computation offloading for cloud of things systems. *IEEE Access* **2017**, *5*, 23947–23957. [CrossRef]
13. Satyanarayanan, M.; Schuster, R.; Ebling, M.; Fettweis, G.; Flinck, H.; Joshi, K.; Sabnani, K. An open ecosystem for mobile-cloud convergence. *IEEE Commun. Mag.* **2015**, *53*, 63–70. [CrossRef]
14. Taleb, T.; Ksentini, A. Follow me cloud: Interworking federated clouds and distributed mobile networks. *IEEE Netw.* **2013**, *27*, 12–19. [CrossRef]
15. Hou, X.; Li, Y.; Chen, M.; Wu, D.; Jin, D.; Chen, S. Vehicular fog computing: A viewpoint of vehicles as the infrastructures. *IEEE Trans. Veh. Technol.* **2016**, *65*, 3860–3873. [CrossRef]
16. Becvar, Z.; Plachy, J.; Mach, P. Path selection using handover in mobile networks with cloud-enabled small cells. In Proceedings of the 2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC), Washington, DC, USA, 2–5 September 2014; pp. 1480–1485.
17. Wang, S. Dynamic Service Placement in Mobile Micro-Clouds. Ph.D. Thesis, Imperial College, London, UK, 2015.
18. Wu, H.; Knottenbelt, W.; Wolter, K. An Efficient Application Partitioning Algorithm in Mobile Environments. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 1464–1480. [CrossRef]
19. Wu, H.; Wolter, K. Stochastic analysis of delayed mobile offloading in heterogeneous networks. *IEEE Trans. Mob. Comput.* **2017**, *17*, 461–474. [CrossRef]
20. Yang, L.; Cao, J.; Liang, G.; Han, X. Cost aware service placement and load dispatching in mobile cloud systems. *IEEE Trans. Comput.* **2015**, *65*, 1440–1452. [CrossRef]
21. Tan, H.; Han, Z.; Li, X.Y.; Lau, F.C. Online job dispatching and scheduling in edge-clouds. In Proceedings of the 2017 IEEE Conference on Computer Communications (INFOCOM), Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.
22. Wu, H. Performance Modeling of Delayed Offloading in Mobile Wireless Environments With Failures. *IEEE Commun. Lett.* **2018**, *22*, 2334–2337. [CrossRef]
23. Wu, H.; Sun, Y.; Wolter, K. Energy-efficient decision making for mobile cloud offloading. *IEEE Trans. Cloud Comput.* **2018**. [CrossRef]
24. Varadharajulu, P.; Saqiq, M.; Yu, F.; McMeekin, D.; West, G.; Arnold, L.; Moncrieff, S. Spatial data supply chains. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.* **2015**, *40*, 41–45. [CrossRef]
25. Corner, M.D.; Levine, B.N.; Ismail, O.; Upreti, A. Advertising-based Measurement: A Platform of 7 Billion Mobile Devices. In Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom), Snowbird, UT, USA, 16–20 October 2017; pp. 435–447.
26. Hu, P.; Dhelim, S.; Ning, H.; Qiu, T. Survey on fog computing: Architecture, key technologies, applications and open issues. *J. Netw. Comput. Appl.* **2017**, *98*, 27–42. [CrossRef]
27. Rao, B.; Minakakis, L. Evolution of mobile location-based services. *Commun. ACM* **2003**, *46*, 61–65. [CrossRef]
28. Beresford, A.R.; Stajano, F. Location privacy in pervasive computing. *IEEE Pervasive Comput.* **2003**, *1*, 46–55. [CrossRef]
29. Cai, C.; Gao, Y.; Pan, L.; Zhu, J. Precise point positioning with quad-constellations: GPS, BeiDou, GLONASS and Galileo. *Adv. Space Res.* **2015**, *56*, 133–143. [CrossRef]
30. Monreale, A.; Pinelli, F.; Trasarti, R.; Giannotti, F. Wherenext: A location predictor on trajectory pattern mining. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Paris, France, 28 June–1 July 2009; pp. 637–646.
31. Han, J.; Pei, J.; Yin, Y. Mining frequent patterns without candidate generation. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD), Dallas, TX, USA, 15–18 May 2000; pp. 1–12.
32. Li, H.; Wang, Y.; Zhang, D.; Zhang, M.; Chang, E.Y. Pfp: Parallel fp-growth for query recommendation. In Proceedings of the 2008 ACM conference on Recommender systems (RecSys), Lausanne, Switzerland, 23–25 October 2008; pp. 107–114.

33. Jin, W.; Li, Z.J.; Wei, L.S.; Zhen, H. The improvements of BP neural network learning algorithm. In Proceedings of the 2000 5th International Conference on Signal Processing Proceedings (ICSP), Beijing, China, 21–25 August 2000; pp. 1647–1649.
34. Wang, L.; Zeng, Y.; Chen, T. Back propagation neural network with adaptive differential evolution algorithm for time series forecasting. *Expert Syst. Appl.* **2015**, *42*, 855–863. [CrossRef]
35. Zeng, M.; Lin, T.H.; Chen, M.; Yan, H.; Huang, J.; Wu, J.; Li, Y. Temporal-spatial mobile application usage understanding and popularity prediction for edge caching. *IEEE Wirel Commun.* **2018**, *25*, 36–42. [CrossRef]
36. Hussain, A.; Aleem, M. GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures. *Data* **2018**, *3*, 38. [CrossRef]
37. Zheng, Y.; Li, Q.; Chen, Y.; Xie, X.; Ma, W.Y. Understanding mobility based on GPS data. In Proceedings of the 10th International Conference on Ubiquitous Computing (UbiComp), Seoul, Korea, 21–24 September 2008; pp. 312–321.
38. Zheng, Y.; Liu, L.; Wang, L.; Xie, X. Learning transportation mode from raw gps data for geographic applications on the web. In Proceedings of the 17th International Conference on World Wide Web (WWW), Beijing, China, 21–25 April 2008; pp. 247–256.
39. Zheng, Y.; Chen, Y.; Li, Q.; Xie, X.; Ma, W.Y. Understanding transportation modes based on GPS data for web applications. *ACM Trans. Web* **2010**, *4*, 1. [CrossRef]
40. Gramaglia, M.; Fiore, M.; Tarable, A.; Banchs, A. Preserving mobile subscriber privacy in open datasets of spatiotemporal trajectories. In Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.
41. Zhang, D.; Huang, J.; Li, Y.; Zhang, F.; Xu, C.; He, T. Exploring human mobility with multi-source data at extremely large metropolitan scales. In Proceedings of the 20th Annual International Conference on Mobile Computing and Networking (MobiCom), Maui, HI, USA, 7–11 September 2014; pp. 201–212.
42. Xu, F.; Li, Y.; Wang, H.; Zhang, P.; Jin, D. Understanding mobile traffic patterns of large scale cellular towers in urban environment. *IEEE ACM Trans. Netw.* **2017**, *25*, 1147–1161. [CrossRef]
43. Che, H.; Tung, Y.; Wang, Z. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE J. Sel. Areas Commun.* **2002**, *20*, 1305–1314. [CrossRef]