Hindawi
Mobile Information Systems
Volume 2017, Article ID 8342859, 10 pages
https://doi.org/10.1155/2017/8342859

Research Article

A Location-Aware Service Deployment Algorithm Based on *K*-Means for Cloudlets

Tyng-Yeu Liang and You-Jie Li

Department of Electrical Engineering, National Kaohsiung University of Applied Sciences, No. 415, Jiangong Rd., Sanmin Dist., Kaohsiung City 807, Taiwan

Correspondence should be addressed to Tyng-Yeu Liang; lty@mail.ee.kuas.edu.tw

Received 4 October 2016; Accepted 28 November 2016; Published 9 January 2017

Academic Editor: Jose Juan Pazos-Arias

Cloudlet recently was proposed to push data centers towards network edges for reducing the network latency of delivering cloud services to mobile devices. For the sake of user mobility, it is necessary to deploy and hand off services anytime anywhere for achieving the minimal network latency for users' service requests. However, the cost of this solution usually is too high for service providers and is not effective for resource exploitation. To resolve this problem, we propose a location-aware service deployment algorithm based on *K*-means for cloudlets in this paper. Simply speaking, the proposed algorithm divides mobile devices into a number of device clusters according to the geographical location of mobile devices and then deploys service instances onto the edge cloud servers nearest to the centers of device clusters. Our performance evaluation has shown that the proposed algorithm can effectively reduce not only the network latency of edge cloud services but also the number of service instances used for satisfying the condition of tolerable network latency.

1. Introduction

In the past decades, the application development of mobile devices always was constrained by computation power. Fortunately, the framework of mobile cloud computing (MCC) [1] was proposed to support mobile devices to offload computation tasks to data centers for execution through networks. Because data centers have a lot of powerful servers to finish the offloaded computation tasks quickly, mobile devices became practical to execute computation-intensive applications with the support of data centers. Consequently, many novel mobile applications such as M-gaming [2], M-learning [3], and M-health [4] have been proposed based on the framework of MCC.

However, MCC recently became not effective enough for latency-sensitive applications running on mobile devices because of long network latency. Generally speaking, there are two main reasons for this situation. The first is the rapid growth of IoT [5] results in that the network traffic of Internet increases as quickly and greatly as the amount of things connecting with networks. The second is that distant data centers are far away from mobile devices. The messages exchanged between mobile devices and data centers must be relayed many times by network routers and should be transmitted for a long distance. To resolve this problem, cloudlets [6] were proposed to push data centers towards the edge of networks for reducing the network latency of delivering cloud services to mobile devices. With the support of cloudlets, mobile devices can ask services from nearby edge cloud servers instead of distant data centers. Because of the reduction of network distance, mobile devices can quickly obtain data and computation services from cloudlets for executing latency-intensive applications such as high-quality video streaming, AR, and VR.

Although cloudlets are much closer to end users than distant data centers, they must address the issue of user mobility for reducing the network latency of delivering services to their clients because users may move among different network domains even when they are accessing services. For addressing this issue, one solution is to create a personal service instance such as a VM hosting applications for each user and make the service instance keep

moving with its client among different network domains to maintain one-hop network latency. Another solution is to deploy a shared service instance such as a VM hosting a specific application on each edge cloud server and make mobile devices adapt their connections with the nearest edge cloud servers for acquiring the same service. Obviously, these two solutions are not economic for service providers and resource managers because the number of users or edge cloud servers is enormous. They are also not effective for resource exploitation because of redundant resource allocation for the same service on different edge cloud servers. By contrast, deploying an enough amount of service instances on proper geographical locations to satisfy the network-latency condition of mobile applications is better for saving budget and resources. To achieve this goal, there are three problems needed to be addressed. They are how many service instances are enough, how to map mobile devices to service instances, and where to deploy service instances.

To resolve these three problems, we propose a location-aware service deployment (called LASD) algorithm for cloudlets in this paper. Basically, this algorithm consists of two steps. The first step is to apply K-means to divide mobile devices into a given number of device clusters according to the geographical locations of mobile devices and then deploy a service instance for each device cluster on the edge cloud server nearest to the center of the device cluster. The second step is to decide if the service deployment configuration obtained by the first step is able to satisfy the constraint of network latency by a prediction function. The proposed algorithm will repeat the two steps by increasing the number of used service instances until the condition of network latency is satisfied or the number of used server instances reaches the maximal number allowed by service providers or resource managers. The simulation results show that the proposed algorithm can effectively minimize not only network latency but also the number of service instances used for the reduction of network latency.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 introduces the proposed algorithm. Section 4 discusses the results of performance evaluation. Finally, Section 5 gives a number of conclusions for this paper and our future work.

2. Related Work

Resource management is a critical issue for cloud computing. In recent years, researchers had put lots of effort on the placement and migration of virtual machine (VM) for addressing this issue. Generally speaking, most of proposed VM placement algorithms are focused on reducing energy consumption [7], maintaining QoS [8], and maximizing application performance and resource utilization [9, 10] in a cloud. However, these algorithms never considered the impact of communication cost while this cost is also an important factor for the performance of cloud services especially when a service is composed of other services distributed over different clouds. For resolving this problem, VM allocation and migration approaches [11] cross clouds were proposed for reducing the communication cost of cloud services. Although the past studies have addressed the issue of resource management for different considerations, they seldom considered the network latency of delivering services from cloud centers to end users. The main reason is that clouds are far away from end users and cloud resources are centralized, so the placement position of VM inside a data center has no significant impact on the network latency of delivering services from the data center to end users. However, cloudlets are close to users and they are distributed at network edges. The placement position of VM becomes an important factor on determining the network latency of deliver services to mobile devices when VMs simultaneously handle requests from mobile devices distributed over networks.

Basically, the service deployment of cloudlets can be divided into two levels. One is physical edge cloud server and another is service instance such as VM hosting a given application. Because the number and location of physical edge cloud servers directly influence the response time of user requests, Jia et al. [12] proposed two algorithms, that is, HAF (Heaviest-AP First) and DBC (Density-Based Clustering) to decide the best location of cloudlets. The HAF algorithm is to sort access points (APs) according to their load and then pick up k access points from the most heavily loaded to the most lightly loaded to be the locations of cloudlets. By contrast, the DBC algorithm calculates the area of high user density to decide which location is the best for allocating cloudlets. The simulation result shows that these two algorithms indeed are more effective than the random placement algorithm for reducing the service response time of cloudlets. However, the HAF algorithm does not consider the problem of load balance. Using this algorithm, the service response time of cloudlets may increase but decrease when the cloudlets are assigned with too many users. Conversely, the DBC algorithm takes load balance into account. Consequently, it is more effective than the HAF algorithm for minimizing the service response time of cloudlets in a given snapshot of user locations. However, mobile users frequently change their locations while the locations of cloudlets usually are fixed after they are allocated. To address the issue of user mobility, the previous two algorithms create a set of cloudlet locations for each of the different user-location snapshots at first and then decide the final locations of cloudlets by making a compromise between the cloudlet-location sets created for the different snapshots of user locations. Therefore, the previous two algorithms are not flexible enough for addressing the issue of user mobility because the locations of cloudlets cannot dynamically be adapted according to the distribution of user locations at any time moment. In contrast, adapting the deployment of service instances is more flexible than the deployment of cloudlets for reducing the response time of edge cloud services because service instances can be dynamically migrated from one cloudlet to another. Accordingly, this paper is focused on the deployment of service instances for reducing the service response time of cloudlets.

3. Proposed Algorithm

We take some considerations into account for the design of the proposed algorithm. First, the number of service instances allocated at the same edge cloud server must be constrained. When an edge cloud server is assigned with too many service instances, this server will be overloaded and then the service instances allocated at this server will not have enough resource to handle the requests from their clients in an acceptable time cost. Therefore, we consider load balance into the proposed algorithm to prevent edge cloud servers from being overloaded. Second, we consider the mobility of mobile devices into the proposed algorithm. We make the proposed algorithm cyclically invoked at regular time intervals to continuously adapt the location and number of service instances. Third, the proposed algorithm is focused on network latency because the main purpose of cloudlets is to minimize the network latency of edge cloud services. Basically, the contributors to network latency include propagation delay, transmission delay, routing delay, and computer delay. Upgrading transmission media and computer capability can result in the reduction of transmission and computer delays. As to routing delay, it is dependent on the hop number between the source host and the destination one while the hop number dynamically changes with the locations of source and destination. It is time consuming and not practical to dynamically measure and update the information of routing delay between any pair of mobile devices and edge cloud servers. Based on the investigation of [13] and our experiment in the next section, the network latency between two hosts is positively correlated with the distance between them. From the viewpoint of algorithm implementation, collecting and updating the location information of mobile devices and edge cloud servers are easy and economical in time cost. Therefore, we consider the geographical distance between mobile device and edge cloud server for evaluating the network latency between client and server in the proposed algorithm. Fourth, the proposed algorithm is constructed based on K-means, which is an iteratively clustering algorithm. Basically, the time spent by the K-means algorithm to converge is unpredictable. Therefore, the proposed algorithm uses a predefined condition to limit the iterations of performing the K-means algorithm.

To simplify our work, we make some assumptions for the proposed algorithm. First, the location and number of edge cloud servers are fixed after they are initially set up. Second, the maximal number of instances available for a service is determined by service providers or resource managers in advance. Third, the downtime of service instances is negligible with the support of live VM [14], lightweight task migration [15], and seamless service handoff [16] when service instances move from one edge cloud server to another.

The proposed algorithm basically has two steps. The first step is to create a candidate service deployment configuration including the mapping of mobile devices to service instances and the allocation of service instances for m mobile devices and k service instances. The value of k is initially set as m/M where M is the maximal number of mobile devices acceptable for a service instance. The second step is to use a prediction function of network latency to check if the candidate service development configuration can commit the condition of network latency or not. If it is true, the candidate development configuration is applied; otherwise, the algorithm increases the value of k by one and then go to the first step again. The proposed algorithm will repeat the two steps until the condition of network latency is committed or the value of k reaches the maximal instance number allowed by service providers or resource managers. If the condition of network latency is not committed as k has reached the maximal instance number, the proposed algorithm will apply the candidate service configuration that is predicted with the smallest network latency. When this situation occurs, it is necessary to increase the maximal number of available service instances for committing the condition of network latency. The rest of this section will detail how to create a candidate service deployment and a prediction function of network latency and will describe how to apply a new service development configuration for reducing the delay of service handoff.

3.1. Creating a Candidate Service Deployment Configuration. The first step of the proposed algorithm consists of two phases: device clustering and load balancing. In the phase of device clustering, it applies K-means to divide mobile devices into a number of device clusters and find the center locations of device clusters according to the geographical locations of mobile devices. Theoretically, the average distance between mobile device and service instance will reach the minimal if the center location of each cluster is deployed with a service instance. However, it is not promised that there is an edge cloud server allocated at the center location of each device cluster. Hence, the proposed algorithm alternatively deploys a service instance for each device cluster at the edge cloud server nearest to the center of the device cluster. On the other hand, K-means basically does not consider the problem of load balance. Consequently, it is possible to map too many mobile devices or service instances to service instances or edge cloud servers, respectively. To resolve this problem, the load balancing phase of the proposed algorithm is dedicated to preventing service instances and edge cloud servers from being overloaded. The two phases of the proposed algorithm are described as follows.

3.1.1. Mobile Device Clustering. K-means was proposed MacQueen [17] in 1967. It is a clustering algorithm which is iteratively redistributing data to a number of clusters for minimizing the value of the following:

$$\sum_{i=1}^{k} \sum_{x \in S_i} (x - c_i)^2 . \quad (1)$$

In this formulation, k is the number of clusters, x is a data item belonging to the ith data set, that is, S_i, and c_i is the center of S_i. To achieve this goal, this algorithm iteratively performs two

steps as follows. The first step is to distribute each data item, that is, x_p, to one data set according to

$$S_i = \left\{ x_p : \left\| x_p - c_i \right\|^2 \leq \left\| x_p - c_j \right\|^2 \ \forall_j,\ 1 \leq j \leq k \right\}. \quad (2)$$

This formula implies that x_p is distributed to S_i if c_i is the cluster center closest to x_p. The second step is to update the center of each data set by

$$c_i = \frac{1}{|S_i|} \sum_{x \in S_i} x. \quad (3)$$

In this formula, $|S_i|$ is the number of data items in S_i. After the update of cluster centers is finished, each data set is reset as empty, and then the algorithm runs for the next iteration as previously described. It is worth saying that it is usual to randomly select k data items as initial cluster centers.

In this paper, we applied K-means for the problem of service deployment as previously described. Therefore, we changed the meaning of the above symbols as follows. The value of k is the number of available service instances; x is the location of a mobile device; S_i is the set of mobile devices assigned to the ith service instance; c_i is the center location of S_i. The value of $(x - c_i)^2$ is the square of the distance between a mobile device and the center of S_i. It is worth noting that a location in K-means is represented by Cartesian coordinate system while the geographical position of a mobile device is represented by northeast-down (NED) coordinate system. Therefore, the proposed algorithm replaces Euclidean distance with great-circle distance in (1) and (2). It translates the location presentation of mobile devices from NED coordinate to Cartesian coordinate for updating the center locations of device clusters and transfers the presentation of center locations from Cartesian coordinate to NED coordinate for calculating the great-circle distance between each pair of mobile device and device-cluster center according to the formulas in [18, 19]. Moreover, the proposed algorithm will finish when all the changes of the center locations of device clusters are smaller than one meter to avoid high execution cost.

3.1.2. Load Balancing. The purpose of the second phase of the proposed algorithm is to prevent service instances and edge cloud servers (simply called ECS later) from being overloaded. Assume M is the maximal number of mobile devices acceptable for service instances. When a service instance is mapped with more than M mobile devices, this service instance is regarded as overloaded. The proposed algorithm will try to remap some of mobile devices from the overloaded service instance to underloaded ones until the service instance becomes not overloaded or no service instance is underloaded. For the reduction of network latency, the underloaded service instance nearest to the overloaded one will be selected in the highest priority to take over the remapped mobile devices. Moreover, the mobile devices that are mapped to the overloaded service instance and are close to the selected underloaded service instance will be preferentially remapped to the selected underloaded one. By the same principle, assume I is the maximal number of service instances acceptable for ECSs. When an ECS is allocated with more than I server instances, this ECS is regarded as overloaded. The proposed algorithm will select and move some of the service instances from the ECS to underloaded ones until the ECS becomes not overloaded or no other ECS is able to accept extra server instances. For the reduction of network latency, the proposed algorithm will preferentially select the service instances assigned with the less number of mobile devices from the overloaded ECS and will move the selected service instances to the underloaded ECSs that are closer to the overloaded ECS.

It is worth saying how to determine the values of I and M for the implementation of the proposed algorithm. In the cloud computing environment, service instances often are implemented by VMs. Before creating the image of a VM for hosting an application service, it is necessary to set up the capacities of resources such as processor's core number, memory, and storage space bound to the VM. Moreover, we can collect the resource information including core number, memory capacity, and storage space of physical edge cloud servers. Based on the resource capacities of edge cloud servers and VMs, we can determine the maximal number of service instances (i.e., I) allowed to simultaneously be executed on the same physical edge cloud server. On the other hand, the workload generated by clients to servers can be regarded as the utilization rates of resources such as CPU, memory, and bandwidth used to provide services for the clients. When a server receives a connection request from some client, it will fork a process dedicated to building a connection link with the client and handling the requests coming from the client later in order to concurrently serve multiple clients. Accordingly, monitoring the CPU, memory, and bandwidth of the forked server process is a common solution to estimate the workload generated by the client. Therefore, we can use Linux system commands to collect the information of resources consumed by a process which is dedicated to serving a given client. For example, *nethogs* can show the amount of data sent and received by processes while *ps* can display the usages of CPU and memory consumed by processes. Although applications may use CPU, memory, and bandwidth at the same time, the performance of most applications usually is dominated by one of these resources. In this paper, the proposed algorithm is devoted to reducing the network latency of cloud services such as video streaming. Basically, the quality of video streaming is dependent on bit rate. For instance, 480p (SD) videos need 500 Kbps while 720p (HD) ones require 800 Kbps for a good display quality. Therefore, the proposed algorithm determines the maximal number of clients (i.e., M) allowed to simultaneously watch video streams from the same server with the same display quality by dividing the available bandwidth of service instances by the bit rate of videos and avoid the amount of mobile devices assigned to the same service instance over the maximal number of clients.

3.2. Building a Prediction Function of Network Latency. In this paper, we built a prediction function for the proposed

Table 1: Response time of streaming server.

School name	Distance (km)	Network latency (ms)
KMU	2.05	28.81457
CSU	2.29	33.72251
WZU	2.41	31.77473
NKNU	2.79	35.48795
FYU	8.27	36.09966
NKMU	8.63	36.80031
NUK	10.4	36.29999
NKUHT	10.7	38.49999
NKFUST	11.8	37.4

algorithm to evaluate the network latency of delivering services to mobile devices under a given service deployment configuration. To achieve this goal, we set up a streaming server at our school campus to provide video streaming services for mobile devices. Afterwards, we used mobile devices in other school campuses to ask services from the steaming server located at our school through Wi-Fi by a command called curl and then estimated the response time of the streaming server. The response time to a user request basically consists of network latency, queuing latency, and startup latency. Because the load of the streaming server was very light and users required the same video by the same mobile device no matter where they were, the queue latency was negligible and the startup latency was identical. Therefore, we subtracted the estimated response time by the startup latency to get the network latency. Finally, we applied curve fitting [20] to the measured network latency and client-to-server distance for generating the prediction function as shown in

$$T(x) = \sum_{i=0} a_i x^i, \tag{4}$$

where x is the distance between a pair of device and service instance.

Our measurement result is shown in Table 1. It is obvious that the network latency of the streaming server indeed increases with the distance between our school and the other. Next, we applied curve fitting to the measurement results in six schools including KMU (Kaohsiung Medical University), WZU (Wenzao Ursuline University of Languages), NKNU (National Kaohsiung Normal University), NKMU (National Kaohsiung Marine University), NKUHT (National Kaohsiung University of Hospitality and Tourism), and NKFUST (National Kaohsiung First University of Science and Technology) to create the prediction function of network latency and then used the distance between our school and each of the other nine schools to test the precision of the prediction function. The formula created by curve fitting is listed in Table 2, and the test result is shown in Figure 1.

It can be found that the prediction values of the function built by curve fitting almost are the same as the network latencies measured in nine schools. The proposed algorithm will use the function in our performance simulation later to predict the network latency between a pair of mobile device

Table 2: Coefficients of prediction function.

Item	Coefficient
x^0	−16775.07322
x^1	31566.74874
x^2	−24263.96287
x^3	9822.006891
x^4	−2223.212169
x^5	262.7000707
x^6	−7.995895109
x^7	−1.841602699
x^8	0.250734929
x^9	−0.012888888
x^{10}	0.000249495

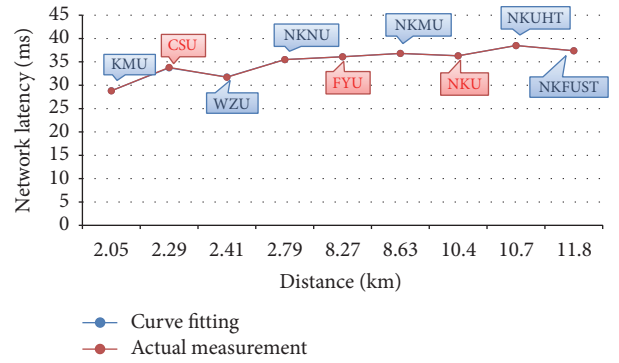

Figure 1: Precision of prediction formula.

and service instance if the distance between them is larger than 2.05 km and smaller than 11.8 km; otherwise, it will use extrapolation to predict the network latency. Here we just demonstrated how to build a prediction function of network latency only with distance for a small range. When users apply the proposed algorithm to real environments, they can build their own prediction function of network latency with extra parameters for a long distance range.

3.3. Applying a Service Deployment Configuration. The simplest way of applying a new service deployment configuration is described as follows. The first is to create a new service instance for each device cluster on the edge cloud server nearest to the center of the device cluster. The second is to let new service instances prepare the data and resources necessary for service requests later and then wait for the connection and service requests of mobile devices. The final is to notify mobile devices of connecting with the new service instances for asking services according to the result of device clustering. This simple way is easy to implement but not efficient because when the amount of necessary data is huge, it requires a long time for service handoff between old service instances and new ones. For reducing the delay of service handoff, an alternative way is to migrate old service instances to the edge cloud servers nearest to the centers of device clusters and then provide services for their old or new mobile clients. Because of data cache, they

can continue to provide the previous services for their old mobile clients without preparing necessary data for a long time. However, the service-instance number of the old service deployment configuration may be different from that of the new configuration. It is necessary to create new service instances if the old service-instance number is smaller than the new one. In addition, how to map the existing service instances to edge cloud servers according to the new service deployment configuration is a key problem for minimizing the delay of service handoff. We proposed a greedy algorithm for addressing this problem as follows.

Let m_{ij} denote the matching degree between a pair of old service instance and device cluster. The values of m_{ij} are equal to the number of mobile devices which are currently served by the ith old service instance and are distributed by the proposed algorithm to the jth device cluster. Let MAT be a two-dimension matching table that consists of $a * b$ matching degrees where a is the number of old service instances and b is the number of device clusters. The process of deploying service instances for device clusters is described as follows.

(1) Let $c = 0$.

(2) Select the m_{xy} with the biggest value from MAT.

(3) Move the xth old service instance to the edge cloud server nearest to the center of the yth device cluster.

(4) Mark the xth row and the yth column of MAT as ignored for matching-degree selection later.

(5) Apply c++.

(6) If $c < b$ and $c < a$ then go to step (2).

(7) If $c < a$ then retrieve which row of MAT is not marked as "ignored" and add the indexes of the retrieved rows into a quitting queue; otherwise, go to step (9).

(8) If the quitting queue is not empty, then delete an index from the quitting queue and store the index to a variable called q; turn off the qth old service instances; go to step (8).

(9) If $c < b$ then retrieve which column of MAT is not marked as "ignored" and add the indexes of the retrieved columns into an waiting queue; otherwise, go to step (11).

(10) If the waiting queue is not empty, then delete an index from the waiting queue and store the index to a variable called w; create a new service instance on the edge cloud server nearest to the center of the wth device cluster; go to step (10).

(11) Terminate the deployment process.

4. Performance Evaluation

We have evaluated the performance of the proposed algorithm by simulation in this paper. For the performance evaluation, we created three modules, that is, device location generator, ECS location generator, and service deployment simulator. The function of the device location generator is to set up the boundaries of service deployment area and create the NED coordinates of mobile devices in the service deployment area. The ECS location generator aimed at creating the NED coordinates of ECSs in the service deployment area. As to the service deployment simulator, it was used to apply a random algorithm or the proposed algorithm to deploy service instances onto ECSs and estimate the average network latency of delivering services to mobile devices. When the random algorithm was applied, service instances were randomly allocated on ECSs while each of mobile devices was mapped to the nearest service instances. Moreover, the deployment simulator used six parameters and one flag, that is, *deployment_area*, *instance_maxnumber*, *device_number*, *ecs_number*, *ecs_maxload* (i.e., I), *instance_maxload* (i.e., M), and *load_balancing* to control the factors which affect the network latency of services and decide to do or not to do the load balancing phase of the proposed algorithm.

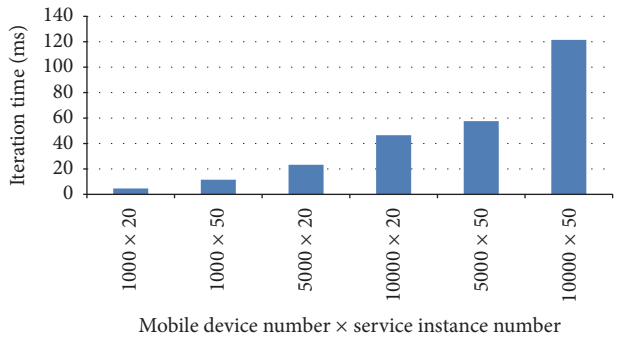

Figure 2: Cost of the proposed algorithm.

4.1. Cost of Proposed Algorithm. The cost of the proposed algorithm mainly is determined by the phase of device clusters. The time complexity of the phase of the device clustering is dependent on the product of m and k where m is the mobile device number and k is the service instance number. We estimated the average iteration time of the proposed algorithm on a workstation with Intel Xeon E5645 2.4 Ghz CPU and 24 GB RAM. The estimation result shows that the iteration time of the proposed algorithm almost increases as linearly as the product value of m and k as depicted in Figure 2.

Even for 10,000 mobile devices and 50 service instances, the iteration time of the proposed algorithm is only 120 ms. In fact, the proposed algorithm can create a candidate service configuration within 10 iterations, namely, 1.2 seconds in most of our simulation cases. Therefore, the proposed algorithm is efficient and practical for the problem of service deployment in edge clouds.

4.2. Effectiveness of Service Deployment. In this performance evaluation, we did three simulations for evaluating the effectiveness of the proposed algorithm (denoted as LASD) under different conditions. In the three simulations, the values of *ecs_maxload, instance_maxload,* and *device_number* are set as 2, 100, and 1000, respectively.

First, we evaluated the network latency of services in different deployment area sizes. In this evaluation,

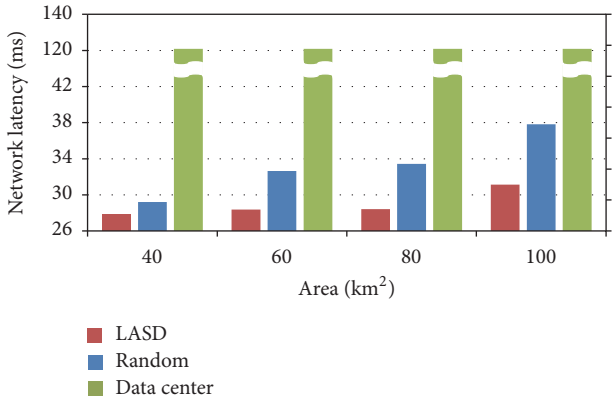

Figure 3: Effectiveness of service deployment in altered deployment areas.

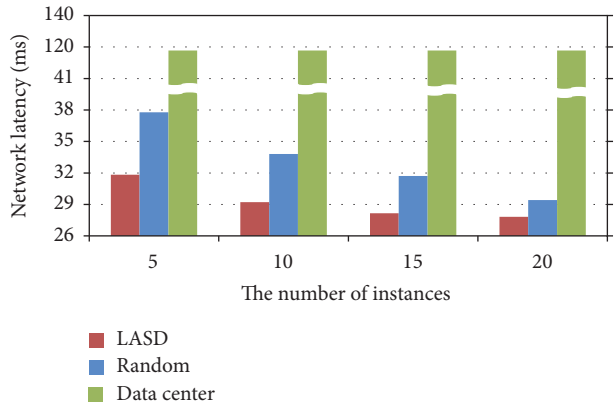

Figure 4: Effectiveness of service deployment in altered instance number.

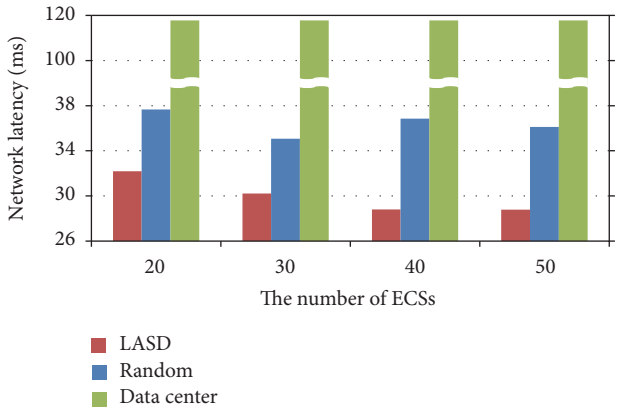

Figure 5: Effectiveness of service deployment in altered ECS numbers.

instance_maxnumber is 10, and *ecs_number* is 20. The simulation result is shown in Figure 3. In this figure, "data center" represents the network latency of delivering services from distant data center to mobile devices. For obtaining this result, we built up a streaming server in the data center located at Tokyo by Amazon EC2. The server is hosted by a virtual machine which has 1 Core vCPU (up to 3.3 GHz), 1 GB RAM, and 8 GB storage space. We used a mobile device to ask services from the streaming server located at the data center and then estimated the network latency of the asked services. This result shows that when mobile devices ask services from ECSs instead of data centers, they can save a lot of time on waiting the responses of cloud servers and even the service instances are randomly allocated on ECSs. However, the network latency of services raises up with the increase of deployment area size if service instances always are randomly allocated. The reason for this situation is that the distribution density of ECS decreases when the service deployment area becomes larger. Consequently, the average distance between mobile device and ECS significantly increases when service instances are not properly deployed. By contrast, the proposed algorithm can effectively improve this situation and successfully degrade the growth rate of network latency because it properly deploys service instances on ECSs. Consequently, the proposed algorithm is more effective than the random one when the size of deployment area increases.

Second, we evaluated the effectiveness of service deployment in different instance numbers. In this evaluation, *deployment_area* is 100 km^2 and *ecs_number* is 30. As shown in Figure 4, the network latency degrades with the increase of the number of services instances. Again, the proposed algorithm is superior to random instance deployment for reducing the network latency of services. However, the performance gap between these two deployment algorithms becomes not obvious when the instance number increases. This result is reasonable because mobile devices have a higher chance to connect with a service instance close to them even when service instances randomly allocated on ECSs. It is worth saying that the average network latency of 20 service instance is almost the same as that of 15 service instances when the proposed algorithm is applied. This situation implies that using 15 service instances has resulted in the minimal network latency, and extra service instances can be saved. In other words, the proposed algorithm is helpful for reducing not only the network latency of delivering services to mobile devices but also the number of service instances used for chasing the minimal network latency.

Finally, we evaluated the effectiveness of service deployment in different ECS numbers. In this evaluation, *deployment_area* is 100 km^2, and *instance_maxnumber* is 10. As shown in Figure 5, the network latency of services does not absolutely degrade with the increase of the number of ECSs if service instances are randomly allocated onto ECSs. After applying the proposed algorithm, this situation disappears. When the number of ECSs increases, the network latency can be degraded further. It is worth noting that the network latency does not obviously change when the number of ECS increases from 40 to 50. This implies that it is possible to use less number of ECSs for satisfying the same condition of network latency with the support of the proposed algorithm.

4.3. Impact of Load Balancing. This simulation is to evaluate the impact of the load balancing phase of the proposed algorithm. In this simulation, *device_number* is 50 and *ecs_number* is 20, *instance_maxnumber* is 10, and *deployment_area* is

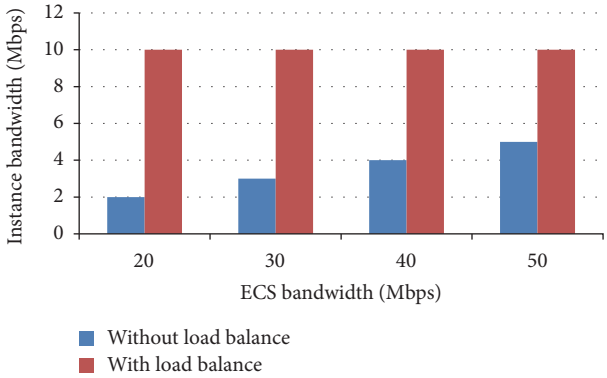

FIGURE 6: Impact of load balancing in service deployment.

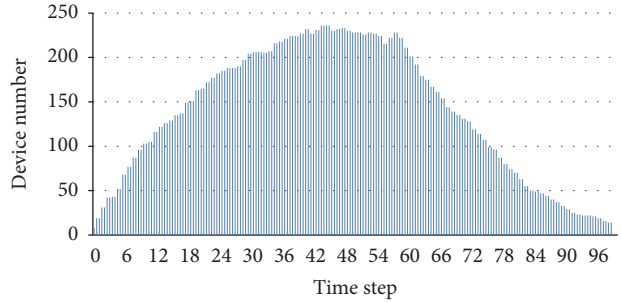

FIGURE 7: Online device numbers in 100 time intervals.

20 km^2. Assume the network bandwidth required by a service instance is 10 Mbps. To create extreme load imbalance, we made the locations of mobile devices close to the same ECS. Consequently, all the service instances were allocated onto the same ECS to achieve the minimal network latency when only the first phase of the proposed algorithm was applied. However, this service deployment results in that all the service instances cannot obtain enough bandwidth no matter how big the network bandwidth of ECS is, as shown in Figure 6.

The main reason for this situation is allocating too many service instances onto the same ECS while the ECS has not enough bandwidth to satisfy the need of service instances. By contrast, all the service instances can obtain enough bandwidth even when an ECS has only 20 Mbps after applying the load balancing phase of the proposed algorithm. Of course, load balancing results in the increases of network latency because some of service instances are reallocated from the closest ECS to others. However, the previous simulation has shown that the proposed algorithm still is more effective than the random deployment for minimizing the network latency of services even when the load balancing phase is applied.

4.4. Effectiveness of Addressing User Mobility. In this simulation, *ecs_maxload* is 2, *instance_maxload* is 100, *ecs_number* is 50, *deployment_area* is 100 km^2, *instance_maxnumber* is 10, and *device_number* is 500. To create user mobility, we made online mobile devices change their location in each time interval while their x-axis and y-axis moving distances in a time interval are smaller than ±5 km. If we intend to drive cars on city roads for 5 km, we need to spend 15~20 minutes when the traffic load is light. Therefore, the length of a time interval in this simulation is set as 20 minutes. In addition, we randomly set up the online and offline times of mobile devices to randomize the number of mobile devices which are served in each time interval, as shown in Figure 7.

On the other hand, we applied the random deployment algorithm, the full deployment algorithm, and the proposed algorithm for each time interval and estimated the network latency of services by two different policies, that is, minimal latency and minimal instance. When the random deployment algorithm was applied, 10 service instances were randomly deployed onto edge cloud servers before the first time interval, and their locations never changed in 100 time intervals. In addition, each mobile device searched which service instance is the nearest to its current location and then connected with the nearest server instance in each time interval. By contrast, the proposed algorithm adapted not only the location of service instances but also the number of service instances used in each time interval. When the minimal latency policy was adopted, the proposed algorithm was devoted to chasing the minimal network latency. When the minimal instance policy is adopted, the proposed algorithm was aimed at committing the condition of network latency with the least number of service instances. In this simulation, the condition of network latency is set as 30 ms. Finally, the full deployment algorithm is to deploy a services instance on each of 50 ECSs while it assumes ECSs are never overloaded even when all of the online mobile devices connect with the same ECS.

In this paper, we also compared the proposed algorithm with HAF (Heaviest-AP First) and Density-Based Clustering (DBC) where the tolerable network latency, that is, T_{net}, is set as 30 ms by simulation. To apply the HAF and DBC algorithms in our simulation, we regarded the locations of edge cloud servers as the positions of access points and viewed the locations of users in each time interval as a snapshot of user distribution. In addition, we individually applied the two algorithms to obtain a set of cloudlet locations for each of the 100 snapshots of user locations at first and then determined a set of cloudlet locations to better suit for all the 100 snapshots of user locations based on the obtained cloudlet-location sets as described in [12]. Finally, we evaluated the effectiveness of the two algorithms for 50 and 10 cloudlets, respectively. No matter what cloudlet number is, edge cloud servers are evenly distributed over cloudlets and each edge cloud server is deployed with one service instance at least.

The simulation result is shown in Figure 8. When the random algorithm is applied, the network latency is not stable and always is larger than 30 ms in 100 time intervals even though it uses the maximal number of service instances. This result implies that although mobile devices keep connecting the nearest service instance, it is not effective enough for them to reduce the network latency of getting services from service instances if service instances do not properly adapt their locations according to the distribution of mobile devices. Conversely, the full deployment algorithm enables mobile devices to connect with closer services instances due to

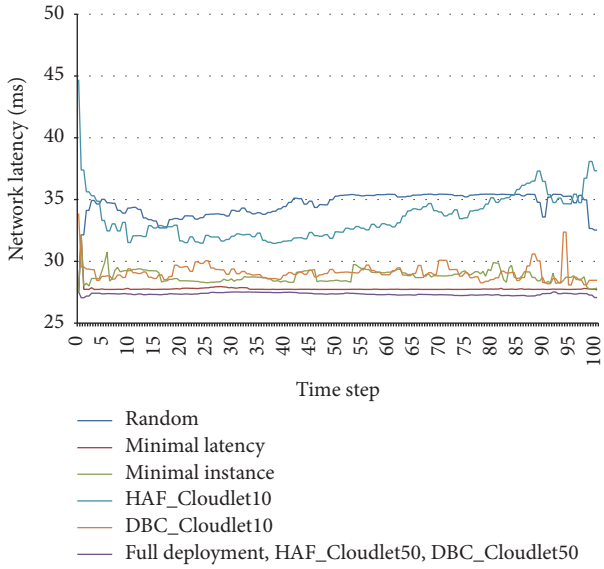

Figure 8: Network latency in 100 time intervals.

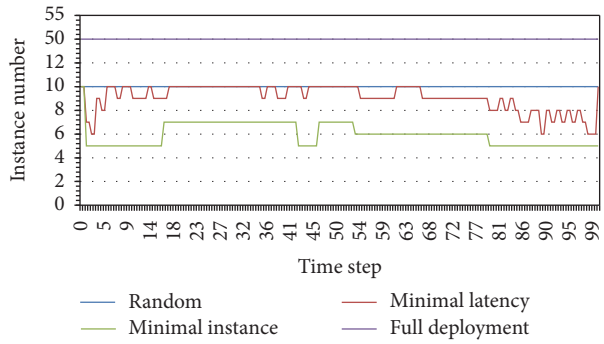

Figure 9: Number of service instances used in 100 time intervals.

the increase of service instances compared to the other algorithms. Consequently, it results in the minimal network latency between mobile devices and service instances no matter where mobile devices move. However, this algorithm is not economic for cloud service providers because it uses the most service instances.

By contrast, the proposed algorithm with the minimal latency policy keeps the network latency between mobile devices and service instances stable no matter how the number and location of mobile devices change. What is more important is that it results in a network latency close to that obtained by the full deployment while it uses much less service instances than the full deployment. When the proposed algorithm adopts the minimal instance policy, it can commit the condition of network latency while it uses the least service instances compared with the other algorithms and the minimal latency policy, as shown in Figure 9. This simulation proves that the proposed algorithm is really effective for reducing the impact of user mobility on the network latency of service delivery and the number of service instances used for chasing the minimal network latency.

On the other hand, the simulation result shows that when the edge cloud servers are centralized into 10 cloudlets, the HAF algorithm is better than the random algorithm for the reduction of network latency between mobile devices and edge cloud servers while it is worse than the DBC algorithm. Although the DBC algorithm performs better than the proposed algorithm with the minimal instance policy in some time intervals, it is worse than the proposed algorithm in most of time intervals. The main reason for the result is that the HAF and DBC algorithms cannot dynamically adapt the locations of cloudlets in each time interval while the proposed algorithm can dynamically change the locations of service instances based on the user locations in each time interval. When edge cloud servers are distributed over 50 cloudlets, the HAF and DBC algorithm are as good as the full deployment for reducing the network latency between mobile devices and cloudlets. Although these two algorithms perform a little bit better than the proposed algorithm with the minimal latency policy, they use much more service instances than the proposed algorithm. As previously discussed, the proposed algorithm is more effective and economic than the others for reducing the network latency of edge cloud services.

5. Conclusions and Future Work

In this paper, we proposed a location-aware service deployment algorithm based on K-means for cloudlets and did a number of simulations for evaluating the performance of the proposed algorithm in different environment parameters. Based on the simulation results, careful service deployment is really necessary and important for reducing the network latency of delivering services from edge cloud servers to mobile devices especially when the area of service development is large or the number of available service instances is small. On the other hand, our simulation has shown that the proposed algorithm indeed can reduce the network latency of service instances to mobile devices and can prevent edge cloud servers or service instances from being overloaded. What is more important is that it can effectively reduce not only the negative impact of user mobility on the network latency of edge cloud services but also the number of service instances used to meet the condition of network latency. For service providers and cloud resource managers, the proposed algorithm is useful for them to effectively cost down and save resources used for the maintenance of QoS.

In this paper, we focused only on network latency for reducing the service response time of cloudlets. However, a cloud service may consist of computation, communication, and I/O. We will take these factors into account for designing an advanced service deployment for cloudlets in future. On the other hand, we will apply the proposed algorithm to real edge cloud services such as high-quality video streaming, AR, and VR.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: a survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[2] S. Wang and S. Dey, "Rendering adaptation to address communication and computation constraints in cloud mobile gaming," in *Proceedings of the 53rd IEEE Global Telecommunications Conference (GLOBECOM '10)*, pp. 1–6, IEEE, Miami, Fla, USA, December 2010.

[3] W. Zhao, Y. Sun, and L. Dai, "Improving computer basis teaching through mobile communication and cloud computing technology," in *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE '10)*, pp. 452–454, Chengdu, China, August 2010.

[4] C. Doukas, T. Pliakas, and I. Maglogiannis, "Mobile healthcare information management utilizing Cloud Computing and Android OS," in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '10)*, pp. 1037–1040, Buenos Aires, Argentina, September 2010.

[5] F. Mattern and C. Flörkemeier, "From the Internet of the computer to the Internet of things," *Informatik-Spektrum*, vol. 33, no. 2, pp. 107–121, 2010.

[6] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[7] J. Dong, X. Jin, H. Wang, Y. Li, P. Zhang, and S. Cheng, "Energy-saving virtual machine placement in cloud data centers," in *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid '13)*, pp. 618–624, Delft, The Netherlands, May 2013.

[8] I.-L. Yen, T. Gao, and H. Ma, "A genetic algorithm-based QoS analysis tool for reconfigurable service-oriented systems," *Advances in Machine Learning Applications in Software Engineering*, pp. 121–146, 2006.

[9] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in clouds," in *Proceedings of the IEEE 4th International Conference on Cloud Computing (CLOUD '11)*, pp. 660–667, IEEE, Washington, DC, USA, July 2011.

[10] N. T. Hieu, M. Di Francesco, and A. Y. Jääski, "A virtual machine placement algorithm for balanced resource utilization in cloud data centers," in *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD '14)*, pp. 474–481, IEEE, July 2014.

[11] J. T. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," in *Proceedings of the International Conference on Grid and Cloud Computing (GCC '10)*, pp. 87–92, November 2010.

[12] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Transactions on Cloud Computing*, 2015.

[13] O. Krajsa and L. Fojtova, "RTT measurement and its dependence on the real geographical distance," in *Proceedings of the International Conference on Telecommunications and Signal Processing (TSP '11)*, pp. 231–234, August 2011.

[14] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '09)*, pp. 51–60, ACM, Washington, DC, USA, March 2009.

[15] R. K. K. Ma and C.-L. Wang, "Lightweight application-level task migration for mobile cloud computing," in *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA '12)*, pp. 550–557, March 2012.

[16] Y. Cui, K. Nahrstedt, and D. Xu, "Seamless user-level handoff in ubiquitous multimedia service delivery," *Multimedia Tools and Applications*, vol. 22, no. 2, pp. 137–170, 2004.

[17] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.

[18] J. Zhu, "Conversion of Earth-centered Earth-fixed coordinates to geodetic coordinates," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 30, no. 3, pp. 957–961, 1994.

[19] *Great Circle—from MathWorld Great Circle Description, Figures, and Equations*, Mathworld, Wolfram Research Inc, 1999, http://mathworld.wolfram.com/GreatCircle.html.

[20] P. G. Guest, *Numerical Methods of Curve Fitting*, Cambridge University Press, 2012.